

Writing as a Tool for Learning Software Engineering

Alf Inge Wang and Carl-Fredrik Sørensen
Dept. of Computer and Information Science,
Norwegian University of Science and Technology,
Sem Sælands vei 7-9, NO-7491 Trondheim, Norway,
Phone: +47 735 94485, email: alfw/carlfrs@idi.ntnu.no

Abstract

This paper presents an educational method used to improve teaching of tedious topics in software engineering courses that can be difficult for students to comprehend without any reference to own practical experience. The method utilizes the students existing software engineering knowledge to understand new theory, methods, and techniques. The goals of the method are to improve student participation in the lectures and to enable the students to think through the topics on their own before giving answers and explanations. The method allows the students to learn in three different settings: Individually, in groups, and in class. This paper describes experiences using the method, and proposes how it can be used in software engineering courses.

Keywords: *Software architecture, software engineering education, writing to learn, and software architecture tactics.*

1 Introduction

Teaching the software engineering discipline is a balance between theory and practice. The students need to know the theory like methods, techniques, standard practices, and life cycle processes to understand the fundamentals of software engineering. They must also get the opportunity to try out the theory in realistic projects to develop skills and get valuable experience. Many software engineering educational papers have focused on how to prepare the students for the real world after the university [7, 11, 3]. These papers focus mainly on how to carry out student projects where the students get realistic experience that is valuable when working in the IT-industry. Although it is challenging to provide realistic software engineering projects to students, it is also challenging to teach the required software engineering theory. Software architecture (SWA) is a part of the software engineering curriculum where some basic engineering knowledge is required to understand the relationships between the SWA and the resulting quality attributes of a final software system. Ideally, software engineering students should have worked in real-world projects in a company for a year or more before taking an SWA course. This would make it a lot easier to motivate the students by mapping the SWA theory to real-world software project experience. In addition, parts of the syllabus of an SWA course consist of systematic methods, lists of attributes, qualities, and design decisions that can be tedious to learn about if they cannot be mapped to something practical. However, most students have some experiences from various programming and software engineering projects that can be utilized.

In this paper, we present the method “*writing as a tool for learning*” that can improve the learning effect of teaching topics in SWA or software engineering that otherwise can be difficult and tedious for students with limited reference to own practical experience. This method makes use of students’ existing knowledge from prior software engineering courses to understand new theory, methods and techniques. The goal of the method is to improve student participation in the lectures and to enable

the student to think through topics on their own before any answers or explanations are given. The method is very easy to use in lectures by introducing small breaks where the students have to think through a topic and write down everything they know about the topic. Afterwards, all students share their notes first in groups and then with the rest of the class. Finally, the results gathered from the students are compared to the textbook.

The paper describes how the method “*writing as a tool for learning*” was used in an SWA course, with the emphasis on one specific lecture on the topic achieving software qualities. We describe how this lecture was planned, executed and we also describe results from the evaluation of the lecture. Further, we describe how this method can be varied to fit various topics in most software engineering courses based on experiences using the method in several software engineering courses.

2 The Educational Method

The educational method “*writing as a tool for learning*” or “*writing and thinking*” is a pedagogical method with a high learning potential. Firstly, the method “forces” students to think through a topic by themselves, making it possible to discover their own perspective on the topic. This is done by asking the students to individually spend 3-20 minutes (depending on how extensive the topic is) writing down everything they know about a specified topic. Secondly, the method helps the students to get various perspectives of a topic by sharing their results in groups of three to four persons. Here, the students should collect on paper the results from the group. Thirdly, letting each group present their results broadens the perspective even more. Here, the knowledge and skills of the teacher can also contribute to learn more about the topic. The documented results of a session as described above will usually be a bullet list of keywords or short sentences about a specific topic.

In addition to improve the learning potential in lectures, the *writing and thinking* method also develops the student’s skills to write. The writing can be divided into two main categories. In the first category, *writing to learn*, the students write in order to discover various aspects of a topic. This can be called discovery thinking, which focuses mainly on creativity and writing for her/his own understanding. The main focus of this category is to internalize knowledge. In the second category, *writing to inform*, the students need to be more critical to what they should write because the results later on should be shared with other people as well as the teacher. This means that this phase focuses on a more analytical writing where the groups clarify uncertainties and prioritize the results to be presented. In this category the focus is on externalizing (sharing) knowledge.

The method *writing as a tool for learning* can also be used in other ways than described above, e.g., arranging write-intensive seminars with longer writing sessions where the focus will be more on the written result.

The best description of the educational method described in this section is found in the Norwegian book [8]. Related written material in English can be found in [1, 4, 10].

3 Planning a Software Architecture Lecture using the Method

This section describes how an SWA lecture on achieving qualities using the *writing as a tool for learning* method was planned.

3.1 Context

At the Norwegian University of Science and Technology (NTNU), the course *Software architecture* is taught to 4th year master students. The credit of the course is 7,5 student points, which is 25% of the workload of a semester. The course is taught as a 3-hour lecture every week in addition to exercises and an SWA project. Usually, about 50-60 students attend this course every year.

The course syllabus is mainly based on the book *Software Architecture in Practice, Second Edition* [2] by Len Bass, Paul Clements and Rick Kazman, and some additional articles. The book is quite

easy to read and describes SWA in a practical manner. However, to lecture SWA based on the book to students without industrial experience is not so easy. Some topics like *documenting SWA* are quite visual and it is rather easy to engage the students. Other topics like *understanding quality attributes* and *achieving qualities* are more difficult to teach in an engaging and motivating way. The main reason for this is that these topics require the students to learn a long list of qualities, the tactics to achieve these qualities, and the relationships between them. It is important that the students understand how to achieve certain qualities of a system by choosing a certain architectural tactic¹. Here it is important to build on existing software engineering knowledge of the students.

The topic for the lecture presented in this paper was *achieving software qualities*. To understand how to achieve certain qualities in a system, the students have to learn various SWA tactics to achieve the system qualities: Availability, modifiability, performance, security, testability, and usability. Many of these tactics have been taught to the students through prior software engineering courses, but they have not explicitly been described as means to achieve certain quality attributes before. The book *Software Architecture in Practice - Second Edition* describes 51 different architectural tactics that can be used to achieve system qualities in the areas mentioned above.

3.2 Goal of the Lecture

The goal of the lecture was *not* primarily to learn the 51 architectural tactics by heart, but more to understand the various mechanisms that affect the system qualities. Also, it was important that the students got to understand the main strategies for achieving certain system qualities like using data redundancy to achieve higher availability. In addition, after the lecture the students should have an idea of how the 51 architectural tactics in the textbook can be used and how they affect a software system.

Another important goal of this lecture was to improve the student participation and engagement in the course. We wanted the students to see the importance of SWA as a topic and improve their interest in SWA topics. Also, since the SWA course has a 3-hour lecture (including breaks), we wanted the lectures to be more two-way communication to avoid sleepy students and a tired teacher.

3.3 Main Strategy for the Lecture

The strategy for the lecture was to introduce and use the *write to learn* method to go through the topic of the day. Our intention was to see if this method could be useful for teaching parts of the SWA curriculum, and see how much the students learned by using this method. To ensure that the students put maximum effort into the lecture, we wanted to have a group-wise contest where the students had to go through a description of the 51 SWA tactics and try to find the quality attribute the tactic addressed. A prize for the best group was to be given.

3.4 The Lecture Plan

Here is the scheduled lecture plan that was made based on the strategy described in Section 3.3:

- **Introduction (10 min):** Introduction to software tactics, the terminology and the main concepts. The students are introduced to the “*write to learn*” method, and the contest with a prize for best performance is also announced.
- **System qualities (6x18 min):** The lecture will go through the system qualities availability, modifiability, performance, security, testability, usability as follows (18 minutes each): First, there will be a 2-minutes introduction to how to achieve the given system quality. Second, the students should individually write down every fundamental design decision (tactic) they know to achieve the given system quality in a system (3 min). Third, the students should

¹A tactic is here defined as a fundamental design decisions to achieve certain system qualities.

discuss and summarise their results on paper (3 min). Fourth, the group results are discussed in the whole class and written down by the teacher (5 min). Last, the teacher compares the gathered results with the tactics described in the book (5 min).

- **SWA tactics contest (27 min):** First, the rules and the procedure for the contest are presented (2 min). Then groups of three to four students have 15 minutes to fill in answers in a form (see Figure 1). Finally, the groups swap forms and the teacher goes through the correct answers, and points are counted (10 minutes). The group with the highest score gets a prize.

Architecture Tactics Super Quiz	
A) Availability M) Modifiability P) Performance S) Security T) Testability U) Usability	
<input type="checkbox"/>	1. Specialize access routes/interfaces: Allows capturing or specification of variable values for a component independently from its normal execution.
<input type="checkbox"/>	2. Limited access: Firewalls restrict access based on message source or destination port.
<input type="checkbox"/>	3. Limit possible options: Reduce the possible options for variations for modules.
<input type="checkbox"/>	4. Spare: A standby spare computing platform is configured to replace many different failed components. It must be rebooted to the appropriate software configuration and have its state initialized when a failure occurs.

Figure 1. A part of the form used for the contest

The total time usage for the triple lecture was estimated to be 145 minutes. Normally a triple lecture lasts 135 minutes (3x45 minutes with 2x15 minutes breaks). To be able to go through with this lecture the breaks had to be shortened from 15 minutes to 10 minutes.

4 Carrying out the Lecture

This section presents experiences from two years (2004 and 2005) carrying out the same lecture as presented above. In 2004, the lecture was executed according to the plan as presented in Section 3.4. However, in 2005 we had to skip to use the *write to learn* method for the particular topic testability, because there was not enough time. In 2005, the 3-hour lectures of the course were split in two days, making it impossible to allocate extra time by shortening the breaks between the lectures.

The first part of the lecture (the introduction) was very similar to most lectures where the teacher presents a topic. This part of the lecture was mainly one-way communication with little student participation. During the introduction, the teacher mentioned that there should be a contest at the end of the lecture with a prize for best performance. From this moment on, most students were paying extra attention to everything the teacher said. Before moving on, the teacher said that all textbooks should be placed in the bag. The reason for this was to “force” the students to think for themselves without any given solutions available in a textbook.

The second part of the lecture where the student had to think, write and discuss a topic worked very well. Firstly, the teacher gave a brief introduction to a subtopic before the students individually had to think through the same topic. All the students put a lot of effort into thinking and writing that resulted in useful topic-related discussions in the groups afterwards. The teacher observed that all students were working very hard to think through and discuss the various topics. Later, when the teacher collected the results from the various groups (by typing the results directly on a laptop computer), it was obvious that the students together could contribute with many of the SWA tactics described in the textbook as well as some additional.

For the lecture held in 2004, the results from all the sessions (availability, modifiability, performance, security, testability, and usability) were collected in one document that was made available on the web for all the students of the course. This document contains a list of the 43 software architectural tactics found by the students. For the lecture held in 2005, the teacher presented the

Topic	Book	Match 2004	Extra 2004	Match 2005	Extra 2005
Availability	13	8 (62%)	2	8 (62%)	4
Modifiability	13	9 (69%)	1	10 (77%)	4
Performance	10	9 (90%)	2	8 (80%)	2
Security	8	7 (88%)	3	8 (100%)	2
Testability	4	2 (50%)	1	No data	No data
Usability	3	3 (100%)	4	3 (100%)	2

Table 1. Comparison of number of tactics in the textbook and the number found by students in the lecture

tactics for the topic testability without using the *write to learn* method. For all other software qualities, the results from the students were collected in a document. This document contains a list of 72 SWA tactics found by the students. Table 1 shows the results from comparing the number of architectural tactics described in the book with the number of tactics found by the students in the lecture. The *book* column shows the number of tactics identified by the book. The *match* columns describe how many of the tactics described in the textbook that were identified by the students in the lectures carried out in 2004 and 2005 respectively. The *extra* columns describe how many additional tactics that were identified by the students. Note that of the 43 and the 72 tactics identified by the students (2004 and 2005), some of them had to be combined to describe one tactic described in the textbook. Here are some examples of the extra tactics identified by the students: Validate input of a system to avoid failure (availability), use big name spaces (modifiability), pre-processing of data (performance), self deactivation (security), modularization for hierarchical testing (testability), and localization of GUI (usability). During the process of collecting the SWA tactics from the students, the teacher had to remove many of the students' suggestions that did not fit into the category or was too general or too specific (e.g., bound just to one technology). The main difference between the tactics identified by the students and the ones described in the textbook, was that the book described the tactics with better precision. From Table 1, we can see that the students identified in average above 75% of the tactics described in the textbook. The students' knowledge originates from earlier software engineering courses and previous experiences from software development projects in software engineering courses.

The third part of the lecture was dedicated to an SWA tactics contest. The students were formed into groups of three or four persons. After an introduction they had 15 minutes to map the 51 SWA tactics described in the textbook to the system qualities availability (A), modifiability (M), performance (P), security (S), testability (T), or usability (U). The students only had to fill in one letter per tactic to indicate the system quality (see Figure 1). The tactics given to the students were short descriptions that could not automatically be mapped to the keyword description of tactics they have found in the second part of the lecture. The students had to read carefully through the descriptions, and think before placing the tactics. During the contest, the groups were discussing eagerly to decide where to place the different tactics. After going through the correct answers, we found that for 2004 two groups had the highest score of 45 points (88% correct) and for 2005 two groups had the highest score of 46 points (90% correct). For both years more than 90% of the groups were above 40 points (78% correct). The contest indicated that the students had learned SWA tactics and their relationships to system quality. The prize to the best groups was a bag of assorted chocolates to each.

An oral evaluation was performed after the lecture, where all students expressed that they were very pleased with how the lecture was taught. Most students said that the way the students were engaged in the lecture through writing and discussion was a very effective way of learning this topic. They also said that the contest was very motivating to put extra effort into individual thinking, group discussions, and to paying attention to what the teacher presented. The only negative comment was that this method is quite time-consuming, and it would be hard to teach the whole book in this way. Further, it was commented that this method is probably not particularly useful when the goal

is to give an overview of a specific topic. However, all students agreed that they had learned a lot more using this method compared to the standard “slideshow lectures”. From the teacher’s point of view, it was an engaging experience to see how active the students could participate in a lecture focusing on the goal of the lecture. The main difficulty with the lecture described above was to limit the discussions to stay on schedule and to cover all topics. To be able to follow the time plan (see Section 3.4), a stopwatch was used to measure the time and the students had to stop at once when the time had run out. This meant that the students had to adjust their discussion to the allocated time. As the main objective of the lecture was to give the students an overview of architectural tactics, the time schedule worked well. However, if the lecture goal had been to go deeper into a topic, the students must be given more time to think, discuss in groups, and discuss in class.

5 The Write to Learn Method in Software Engineering Courses

Previous section described how the *write to learn* method was used in one particular lecture teaching the students architectural tactics to achieve specific qualities in a software system. We have also been using the method similarly to teach other topics in the SWA course like usage of different views in SWA, how an SWA can be reconstructed from an available system and how SWA is used to build a system using off-the-shelf components. In all these lectures the write to learn method was used to *discover* a new topic using the students prior knowledge and limited experiences.

Another way of using the method is to let the student *elaborate* on a particular problem where the teacher has sketched an outline of a solution. Here the students get time to think through the details of a method or a technique, before the results are compared with the textbook. When the method is used to elaborate a problem, the students must be given more time in order to dive deeper into the problem. The elaborative version of *write to learn* is suitable for teaching topics like design patterns and architectural patterns. For these topics, the teacher presents a problem and gives the students an overview of a pattern that solves the problem. The students will then get time on their own, and later in groups to understand how the pattern solves this problem.

A third way of using the method is for *repetition* of material that has been previously presented in the lecture. Here the students are asked to write down everything that they can remember from a topic presented in the lecture. The results are then discussed and summarized in groups, for later to be presented for the whole class. The method can be used for repetition of any subject and the time duration can be adjusted whether the teacher wants a deep or shallow repetition.

Table 2 identifies some variations of the method that can be used to achieve a variety of educational goals. The columns describe variations in how the knowledge is used in the method, and the rows describe the duration aspect. If the method is used for longer durations, it is required that the students have deeper knowledge of the topic. A short duration is defined from 1-5 minutes, medium is 6-10 minutes, and long is 11-20 minutes. By using this table, it is possible to use the *write to learn* method in most topics in a software engineering course. It is also possible to ask the students to draw figures instead of write keywords if this is more suitable for a particular topic.

From our own experiences we have found that especially the *short repetition* variant of the method suitable to be used for all kinds of topics, and it helps the students in remembering the main parts presented in a lecture.

Even though the *write to learn* method has an improved learning effect for the students and it can be used for any topic, it should mainly be used to spice up lectures. If the method is used too much, the learning effect will suffer. The method must be seen as a useful supplement to other educational methods, which makes it possible to create more varied lectures.

6 Related work

The problem of teaching SWA as described in this paper is the same problem as in teaching software engineering: How to teach students without any reference to real software projects, software

Duration	Discover	Elaborate	Repetition
<i>Short</i>	Introduction to topic	Understand concept overview	Remember outline
<i>Medium</i>	Get overview of topic	Understand concept	Remember most important
<i>Long</i>	Work through topic	Understand concept details	Remember most material

Table 2. Educational goals of variations of the learn to write method

engineering techniques, methods, processes, tools etc.? In this section, we will describe some related work that focuses on software engineering and education.

In [9], Hilburn proposes a conceptual model for software engineering education. The model describes three main areas that are important in software engineering education to improve software engineering practice. The first area is *people*, identifying the need for learning software engineers to interact effectively through learning communication and team skills as well as ethics, and human and social sciences. The second area is *process*, focusing on individual, team and organizational processes. These are processes that can help software developers to develop software in a better way. The third area is *technology*, focusing on traditional science and mathematics as well as computer science and software development. Further, Hilburn proposes a more detailed software engineering curriculum model identifying different courses and in which year these courses should be taught. SWA as a separate course/topic is left out of this model. We believe that SWA is a very useful and necessary course that should be included in the last year of a software engineering study. In [7], Díaz-Herrera and Powell argue that SWA is a very important part of a software engineering curriculum. They claim that to educate industrial-strength software engineers, the students need to be capable of solving problems in relatively large and complex application systems. To enable the students to do so, they propose a model-based software engineering curriculum, where the students must learn abstract models for requirements, architectures, components etc., as well as solution models for domains, design, implementation etc. A main focus in such a curriculum will be an SWA course teaching architectural styles, languages and patterns.

Our paper focuses on how to teach the theoretical material of an SWA course. To really learn software development, the theories must also be tried out in projects or similar. In [6], Dawson presents twenty dirty tricks to train software engineers. The main idea is to introduce events that are likely to happen in real projects in training projects. Often training projects are carried out in an ideal world. Dawson's idea is to allow the students to work in projects where they have to deal with challenges like inadequate specifications, uncertain customers, changing deadlines, introduction of quality inspections etc. These challenges are called dirty tricks, and Dawson has identified twenty such tricks with a belonging lessons learned from experiences facing this tricks.

In [5], Crnkovic et al. describe a case study of a software engineering course where they discovered that the students also required skills in engineering of a non-software nature. In this course, the students had to do a project where they had to develop a simulation model to visualize the stability of an old war-ship. In addition, it was required that their system should be easy to modify. This problem could be attacked by using architecture or design patterns. The experience from this course showed that the students were able to deal with non-software domains like the model of the boat, but had a lot of difficulties to make a modifiable system. This shows that it is important to teach software engineering students the foundations in SWA.

7 Conclusion

In this paper we have described a method that can be used to teach SWA theory as well as other software engineering theory. By allowing the students to think through topics and write down their thoughts, it is easier for them to participate in topic related discussions. This is made possible by letting the students use their existing software engineering knowledge to understand new topics. In addition, the students will learn more from the lectures using this method than traditional lectures

because they have to work with the topics by themselves. It is common knowledge that it is easier to remember things that are written down and formulated on your own. Further, the students get multiple perspectives on the topics being taught from thinking and writing on their own, discussions in small groups, discussions in large group, comparison with the textbook, and shared experience from the teacher.

By using variations of the methods as presented in the Table 2 in Section 5, the method can be used to fit most software engineering topic. These variations make it possible to discover new aspects of a topic, elaborate a topic and repeat a topic. In addition, the duration aspect makes it possible to tailor the method to fit into the schedule of a lecture.

The main benefit from using the method is to improve theme-based discussions in the class, improve student participation, and help the students to learn more efficiently. By breaking up lectures allowing the students to think and write, the teacher can avoid long one-way speeches that students rarely remember anything from. Through further exploration, we can build a list of software engineering topics that are suitable to be taught using the method of “*writing as a tool for learning*”.

References

- [1] Linda Allal, Lucile Chanquoy, and Pierre Largy, editors. *REVISION, Cognitive and Instructional Processes*, volume 13 of *STUDIES IN WRITING, International Series on the Research of Learning and Instruction of Writing*. Kluwer Academic Publishers, Boston, January 2004.
- [2] Len Bass, Paul Clements, and Rick Kazman. *Software Architecture in Practice - Second Edition*. Sei series in Software Engineering. Addison-Wesley, 2003.
- [3] M. Brian Blake and Todd Cornett. Teaching an Object-Oriented Software Development Life-cycle in Undergraduate Software Engineering Education. In *15th Conference on Software Engineering Education and Training (CSEET'02)*, Covington, Kentucky, USA, February 25–27 2002.
- [4] P. Creme and M.L. Lea. *Writing at University. A guide for Students*. Open University Press, Buckingham, 1997.
- [5] Ivica Crnkovic, Rikard Land, and Andreas Sjögren. Is Software Engineering Training Enough for Software Engineers? In *16th Conference on Software Engineering Education and Training*, Madrid, Spain, March 20–22 2003.
- [6] Ray Dawson. Twenty dirty tricks to train software engineers. In *Proceedings of the 22nd international conference on Software engineering*, pages 209–218. ACM Press, 2000.
- [7] Jorge L. Diaz-Herrera and Gerald M. Powell. Educating Industrial-strength Software Engineers. In *11th Conference on Software Engineering Education and Training (CSEET'98)*, Atlanta, GA, USA, February 22–25 1998.
- [8] O. Dysthe, F. Hertzberg, and T.L. Hoel. *Skribe for å lære. Skrivning i høyere utdanning*. Abstrakt forlag, 2000.
- [9] Thomas B. Hilburn. Software Engineering Education: A Modest Proposal. *IEEE Software*, November/December 1997.
- [10] A. Nortledge. *The Good Study Guide. The Open University*. Milton Keynes, 1990.
- [11] V.E. Veraart and S.L. Wright. Experience with a Process-driven Approach to Software Engineering Education. In *1996 International Conference on Software Engineering: Education and Practice*, Dunedin, New Zealand, January 24–27 1996.