# Is Software Engineering Training Enough for Software Engineers?

Ivica Crnkovic, Rikard Land, Andreas Sjögren
*Mälardalen University, Department of Computer Science and Engineering*
*PO Box 883, SE-721 23 Västerås, Sweden*
*{ivica.crnkovic, rikard.land, andreas.sjogren}@mdh.se*
*http://www.idt.mdh.se/{~icc,~rld,~ase}*

***Abstract***

*Most software engineering courses focus exclusively on the software development process, often referring to problems related to the complexity of software products and processes. In practice, however, many problems of a complex nature arise in which system engineering and other engineering disciplines are important in the development of systems. In such cases software engineers may have difficulty in coping with the entire problem, in the same way that engineers in other fields may have difficulty in understanding the software part. This suggests that the software engineering education of today is inadequate in certain respects. This paper presents a case study of a software engineering course and discusses the difficulty for computer science students to understand and to develop a system which also requires skills in engineering of a non-software nature.*

## 1. Introduction

During recent years, as the use of software has increased and software has become more complex, there have been many discussions about education in software engineering, today and in the future [15,16]. The main challenge of such education is to prepare students for the "real world" [8,9,17], which is inconsistent, unpredictable, complex and always in a state of change. There are different approaches to meet this challenge; one example being the execution of projects based on real examples from industry [9]. Another approach is to "simulate" the real world by introducing a number of unpredictable obstacles into student exercises, the "dirty-tricks" method [10]. There are approaches which are focused on different areas of software engineering, such as requirements engineering, software design, validation and verification, others focused on the management of the complexity of products and processes. Surprisingly, the relation between software engineering and other engineering disciplines such as electrical engineering, mechanical engineering, different aspects of civil engineering is seldom discussed. In emerging as an independent discipline, software engineering has moved away from other classic engineering disciplines. The use of many classic engineering methods has been decreased or even removed from the software engineering curricula. While we can argue that this process is natural since there is no immediately apparent need for knowledge of classic engineering subjects for software development, we must also note that the frequency of project failure due to problems in software is significantly greater than that due to problems in other engineering disciplines. In too many cases the causes of failure originate in a misunderstanding of requirements, mismatches in system design and implementation, unrealistic expectations, and bad project planning [14]. The question is how much these problems are the results of a lack of formal engineering methods and procedures, and from the inability to manage the development of complex systems without using a systematic and analytical approach.

We believe that software engineering education should also include the training of future software developers in understanding and using (at least to some extent) the methods of

classical engineering. We also believe that software engineers will increasingly encounter problems which are not of a pure software character, the software being only one part of a complex system. To understand the entire system, knowledge from pure software engineering will not be sufficient. Finally, we suggest that it is important to develop a culture of a systematic and analytical approach to the solution of problems, widely absent in the software community. For this reason we have updated our software engineering course for computer science students with additional elements: an analysis and modeling of a non-software system [1]. This gave us the possibility of analyzing the student's ability to cope with software and non-software related problems. This paper describes the experience from the course and suggests the importance of the inclusion of elements from other engineering disciplines in software engineering.

## 2. System Engineering vs. Software Engineering

In many products of today which embody the knowledge and skills of engineers from different fields, software is becoming the major and the most important part. Software increasingly controls the functionality and the overall behavior of the entire product. To develop these products successfully, extensive knowledge of software and of other engineering disciplines is required of the project team. However, understanding system requirements and their relation to software requirements remains one of the main difficulties in the development of computer-based systems [11]. It is not only because of the different nature of the requirements and characteristics of systems and software but also because of the different approaches to the development process [6]. A standard approach in a system development is a top-down approach; the system design beginning with the specification of the system architecture, the architecture defining the components and their interactions. Technologies and engineering methods from different engineering fields are then applied to the specification, design and development of components. While classical engineering disciplines follow this strict approach, software development process may be significantly differently. The top-down approach is often replaced by a bottom-up approach, sequential development processes are often experienced as non-appropriate models, and evolutionary models (incremental, rapid-prototype, spiral model, etc.) are used. Software developers have no strong culture of exact and formal specification. This incompatibility of approaches causes the mismatching of requirements, specifications and finally in the products. Our opinion is that this difficulty can be alleviated to a degree by making "system thinking" more explicit in software engineering courses, as we describe.

## 3. Case Study: Software Engineering Course

The software engineering course we analyze here [7] was attended by computer science students. They have attended basic courses in mathematics but very few courses that are parts of traditional engineering curricula, for example physics. The course concerned was presented during one semester and was divided into two parts: the first half of the course consisting of lectures and labs; the second half being devoted to project work. In the project phase, the students worked in independent groups with large grade of freedom, but the milestones and final results were specified by the teachers who also were involved in the project as customers and a steering group.

In the project, the two main elements given specially attention were a system analysis and architectural issues. The goal was to give the students an understanding of the need for a formal specification, based on the engineering methods associated with the system being developed.

### 3.1 The System Engineering Problem

The main idea of the project was to give students a problem not of a pure software character and which it would not be possible to solve by software development alone. The

main assignment was to explain the capsize of the Vasa, a 17th century Swedish warship [2] and to develop a simulation model which would visualize the stability of the Vasa, or a user-configured ship [1].

Stability calculations are derived from basic forces of physics: gravity and buoyancy. Due to limited space, we do not explain the details, but Figure 1 visualizes the forces in action and the interaction between these. *W* denotes the wind force, *G* the gravity force, *B* the buoyancy force, and *h* the horizontal projection of the distance between the centers of gravity and buoyancy. The ship stops heeling in a position at which the forces and torques (leverage effects) are in a state of equilibrium. In case a) shown in the figure the resulting force will keep the ship in a stable position while in case b) the ship will capsize.

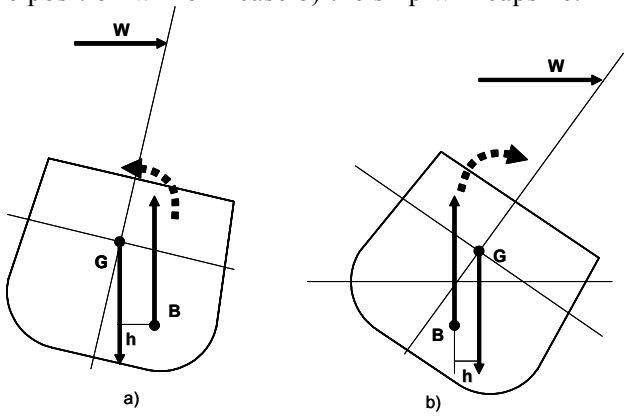


Figure 1. Stable and unstable attitudes of a ship

### 3.2 The Project Assignments

During the project work period the students were required to analyze the problem to be solved, define and process requirements, and design and implement the system. The project work was focused on project planning, process measurement, teamwork, and an analysis of the work performed. In addition to "standard" assignments such as project management and negotiating with a "customer", two specific requirements were added:

- *Modifiability.* The system was required to be designed and implemented in such a way that it would be easy to modify. The students were expected to refine this requirement and describe how they intended to accomplish this, and analyze how well they succeeded.
- *Mathematical Model.* The mathematical model used for stability calculations was to be documented, and its correctness and accuracy analyzed.

The aim of these two additional requirements was to make the students familiar with long-term requirements and with quality issues.

## 4. The Students' Project Work

The project was to serve as an experiment. It was to indicate the ability of computer science students to cope with more complex problems related to a) software engineering and b) other engineering disciplines. The project was to show if the students are able to achieve similar results (i.e. a similar level of quality) in solving problems of a similar degree of difficulty but related to other domains than software engineering. It was also to investigate the student's ability to cope with unfamiliar problems as is very likely in their professional life. We shall concentrate on these additional requirements in the following.

### 4.1 Modifiability

The students discussed modifiability in two senses:

- In a general sense, by using e.g. a design pattern [12] and an architecture [3,5] that are believed to support modifiability, by using an object-oriented approach, or by establishing coding guidelines, structuring the code commenting it well etc.

– In a more specific sense, by assuming particular future changes which are more probable, and preparing their design for these modifications (similar to the way scenarios are used in the Software Architecture Analysis Method, SAAM [3,13]).

It was apparently difficult for the students to understand "modifiability" and the groups succeeded to different degrees. All of the groups had a modifiability requirement in their requirements document; however, when refining it, there were quite different approaches. Two groups explained modifiability with both a general description and a description of specific changes which could be used to test whether the system was modifiable.

In the design documentation, we expected to see documentation of the design decisions that were motivated by the modifiability requirements, according to the groups' own definitions in the requirements document. Three groups provided no explanation of how they had achieved modifiability, two groups showed how it was possible to implement the particular scenarios they had specified in the requirements document (although one group omitted their most interesting scenarios – making the system non-web-based and changing calculation model – and instead focused on "lower-level" scenarios, e.g. how to add a new deck), and one group explained that the system was modifiable both through their choice of architecture and through specific scenarios. Only one group discussed modifiability elsewhere than in a separate section, which might imply that they had a slightly deeper understanding of the issue.

The groups' achievements are summarized in Table 1.

Table 1: A summary of how the six groups treated modifiability

| | *Group No:* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|---|
| Requirements Specification | General | x | - | - | x | - | x |
| | Specific scenarios | - | x | x | x | x | x |
| Design Specification | General | x | - | - | - | - | - |
| | Specific scenarios | x | - | - | x | x | - |

We can conclude that the students achieved a certain understanding of modifiability. While they understood the requirements and to some extent their design was influenced by these requirements, they showed less ability in analyzing and explicitly documenting their decisions. The design was more "construction" oriented than property-oriented.

### 4.2 Mathematical Model

The most interesting, and complicated, part of the system was its ability to visualize the stability of a proposed ship. A part of the assignment was to develop the configuration of the ship: a ship can be built with different numbers of decks, different numbers of guns per deck, with or without ammunition and with different amounts of ballast. These configuration parameters and the form of the ship determine the center of mass of the ship. The center of buoyancy is defined by the shape, weight and heel angle of the ship.

Given a certain wind speed, the resulting heel angle of the ship can be determined. At some wind speed, the ship will capsize. The students thus had to develop and implement a mathematical model of the physics involved. To help the students, we gave a separate lecture describing the model and the basic physical principles such as methods for calculating the center of mass and the equilibrium state of an object.

**4.2.1 The Models:** The main difficulty was the determination of the center of buoyancy, which is at the center of the area of the displaced water. The shape of the cross section of that part of the ship under water when the ship is heeling may be quite complicated. However, it is possible to divide this area into a set of simple geometrical forms, and then its center can be calculated from the geometrical centers of these forms. The students thus had to model the ship's shape, take into account the heel angle, and by geometrical means divide the area into a set of simple shapes, find their the geometrical centers and from them calculate the geometrical center of the complete area.

The students specified basically three types of models:

- Four groups approximated the shape of the ship to a rectangle and calculated the center of buoyancy in either of two different ways, as shown in Figure 2A and 2B.
- One group approximated the shape to a rectangle with the corners "cut off". To find the center of the area they approximated the area using vertical bars, as shown in Figure 2C. Using this strategy, it would have been possible to choose a more realistic shape and – with more subdivisions of the polygon – any level of detail of the area of displaced water, but the group chose this very simple shape without further analysis.
- The last group also approximated the area as that specified by a set of points in the coordinate system, but used a simple but elegant method to approximate the shape of the ship: they superimposed a cross-section of the ship on a coordinate system and obtained a number of coordinates (arbitrarily chosen). The area was then divided into triangles, which made it easy to find the center of buoyancy. See Figure 2D.

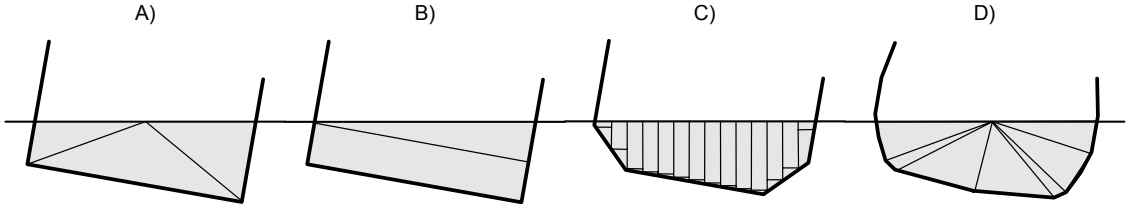


Figure 2. The shapes and the calculation principles of the area of the ship's crosscut.

It is quite apparent that the calculations needed when using only three triangles (as in 2A) are not significantly less complex than those needed using several triangles (as in 2D), but that the latter approach will give a much more exact result. There were thus no implementation reasons for the students to choose a solution with a simple approximation of the shape of the ship – the calculations needed still required a sequence of calculations of coordinates. This observation leads us to believe that the students had decided that "just choosing a very simple shape of the ship will make the mathematics and the implementation easier" – which is a misconception. Furthermore, there are no signs that they even considered other alternatives. They had learned the basic physics from a lecture and then used the most direct method of approximating the shape of the ship in their determination of the center of buoyancy.

**5.2.2 Documenting the Models:** As the projects proceeded, the steering group and the customer were somewhat surprised that no group documented their model – nor were there any descriptions given of how they approximated the shape of the ship nor of how they divided the area under water into simpler shapes. Therefore, in the middle of the project, we requested documentation of their model, and the performance of some kind of analysis or verification of the accuracy of their model. The most common reaction to this was that they were now assigned an additional task in the middle of the project – which they felt to be unfair. From project documentation and group meetings during the project, there is no sign of any group having begun their project work by even considering that they should evaluate the mathematical model they found or developed – many of them did not understand they had already developed a model. Only one group began immediately the documentation requested; presenting satisfactory documentation the next week; the other groups did not document the model until the very last week of the project.

Three groups at first documented their model by describing the algorithm they had implemented, two groups using flowcharts and one group a pseudo-code. Although this was a good beginning, they did not perceive that they should also document the model as such.

**5.2.3 Analyzing the Models:** We did not expect any major analysis or verification of their models; we considered it sufficient to use one example of a ship fully equipped (one particular set of cannons etc.), using the original Vasa data. The crucial approximation incorporated in the models is that of the ship's shape, which affects the center of buoyancy

and the center of mass. One means of verifying the model considered was to model the ship's shape more exactly (for this single case) and to find the center of buoyancy and the torque through some advanced method, analytical or numerical, available in some powerful mathematics program. One would then expect the students to analyze the consequences of a set of heel angles by comparing the torque of their model with that of "reality" – or at least as close to reality as possible. It would then be easy to present the accuracy of their model in a plot such as that schematically shown in Figure 3 in which they would indicate the differences in calculation of their model with a "real" model. Additional analysis would include a study of the difference between the critical angles with different wind speeds and different configurations of the ship.

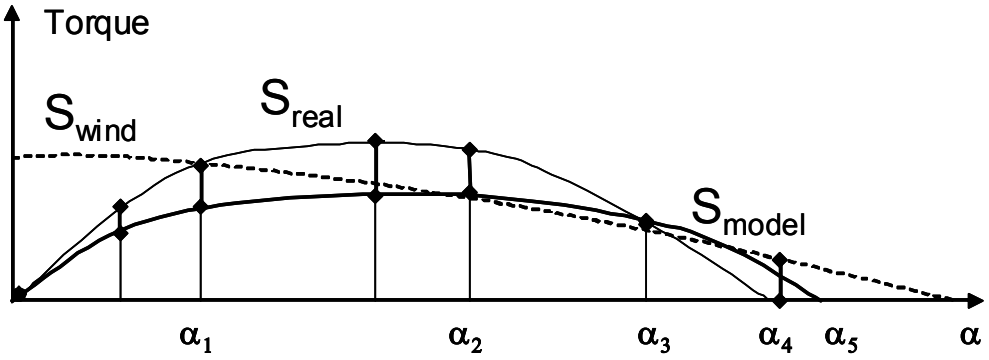


Figure 3.A schematic plot showing an analysis of the model.

The group that had approximated the ship's shape most closely (Figure 2D) used another approach: they verified their method, not their model. They approximated another shape, a hull with a semicircular cross section shaped as a semicircle, with 10 points, and compared the exact area (simple to calculate for a semicircle) with the approximated shape, as a number of triangles. However, they did not analyze the effect of the number of points on the accuracy – how much would 20 points have improved the accuracy? But this example shows a typical engineering approach; if you cannot calculate (or do not know how to calculate), use measurements from a particular case. Similarly, the group using the shape presented in Figure 2C did not analyze the effects of a larger number of coordinates on the accuracy.

Two of the groups had a smarter and more scientific and engineering approach. Using a recognized formula for calculating the torque, given the wind speed [4], they compared the results thus obtained with those they obtained by their own method, one group used two different wind speeds, the other one only (4 m/s, that when the Vasa capsized). This should be considered a respectable analysis, although we would have preferred the analysis of a larger number of heel angles or wind speeds.

One group performed a series of comparisons with different wind speeds, but compared their rectangular cross section hull not with a "real" ship but with a ship with a semi-circular cross section; they seemed to understand that this was not a verification, only a comparison between two crude approximations, but this was the best idea they had. This approach would show how close are the different approximations and assuming that the real shape is something between these two shapes, the correct result would probably lie between these approximations.

The other verifications did not reach this level. Two groups made a drawing of their rectangular ship and of the real ship (from a photo) on a graph paper, and counted the number of squares covered by the ship. Needless to say, this was a very inexact verification – perhaps more inexact than their model? One group performed no analysis or verification at all; they only discussed briefly the difficulty of modeling the ship sufficiently accurately.

Our impression is that this task proved difficult for the students to understand. Not all of them seemed to understand the purpose of verifying the model in this way at all. Others understood the purpose of such an analysis, but had no idea of how it was to be performed.

And we daresay that not until the end did some of the students realize that such an analysis could have been used at the beginning of the project to choose a suitable model.

The groups' models and their verifications are summarized in Table 2.

Table 2: A summary of how the six groups verified their mathematical models.

| *Group No:* | *1* | *2* | *3* | *4* | *5* | *6* |
|---|---|---|---|---|---|---|
| Shape of ship | ⊔ | ⊔ | ∪ | ⊔ | ⊔ | ∪ |
| Checked paper | x | x | - | - | - | - |
| Center of Buoyancy[1] | x | - | - | - | x | - |
| Verification of method | - | - | - | x | - | x |

[1]The center of buoyancy was compared with the formula found in [4].

## 6. Conclusion

The students knew quite well how to design and implement software to fulfill requirements; all groups produced design documents that were at least acceptable. The execution of the project was also satisfactory with very good teamwork. The issue of modifiability was seemingly harder; they understood that software should be easy to modify since one can always expect new requirements or changing environments, but it was hard for the students to know how to make the requirement of the software being modifiable more specific, and to show that they had succeeded. Finally, all groups certainly implemented a model of a ship; but its documentation and analysis was obviously a hard assignment.

Of course, students of computer science have limited experience in physics, so it is perhaps unfair to expect them to perform equally well in the physical modeling as the software modeling. However, although a physics student could be expected to implement a better model or a better numerical algorithm (and probably have problems with the software part), our point is rather that the students easily lost sight of the software as (part of) a system. They happily performed a software design and implemented it, and we had in general no serious remarks on their project work, but they seemed to view the software without reflecting on the result from a system perspective.

What are the lessons learned from the course result? We can discern at least two:

- *Previous education.* Two and a half years of studying computer science or computer engineering at university level had not given them "system thinking"; they focused on the software and the project work.
- *Teachers.* The assistants, at first, neither appreciated the importance of the model nor its documentation; they probably gave the impression that this was indeed a "new assignment" which turned up in the middle of the project. They had thus finished their education without sufficient awareness of system thinking! The main teacher, being also the customer, did not explicitly state the requirements of model and analysis and did not place them within the perspective of computer science students. This was an exemplary case of a misunderstanding between stakeholders due to different assumptions.

Execution of a project is today an established part of most Software Engineering courses [1,7,8,10,14-17]. Such projects usually cover different phases, from the requirements elicitation and specification to the implementation, validation and verification. We can designate these parts as "Software Engineering centric" tasks, as opposed to what we would prefer to call "System-Engineering centric" tasks. We claim that the focus on modifiability (and on other non-functional properties) requires more of a holistic and system perspective. The tasks that require particular knowledge from other engineering or basic disciplines such as physics and mathematical modeling (and analysis) can be denoted as "Domain Engineering centric" tasks. We argue that these parts are even more difficult for software engineers (see Figure 4).

When the students leave university, many of them will build software systems that are a part of larger systems – mechanical systems, electronics systems, web sites, etc. So, we arrive at a very serious question: are they prepared for this task? Will they be able to create "good

systems", or will they be limited to creating "good software"? We believe that software engineers and students of computer science should have more insight into engineering disciplines other than pure software engineering.

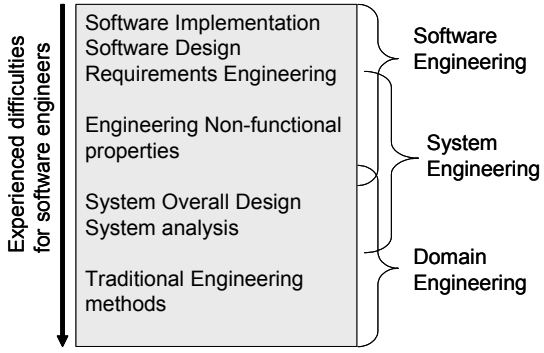


Figure 4. The difficulty of performing engineering tasks for software engineers

## 7. References


[1] *Mälardalen University, Software Engineering Course*, URL: http://www.idt.mdh.se/kurser/cd5360/, 2002

[2] *The Vasa Museum*, URL: http://www.vasamuseet.se/indexeng.html, 2002

[3] Bass L., Clements P. and Kazman R., *Software Architecture in Practice*, Addison-Wesley, 1998.

[4] Borgenstam K. and Sandström A., *Why Vasa capsized*, Vasa Museum, 1995.

[5] Bushmann F., Meunier R., Rohnert H., Sommerlad P., and Stal M., *Pattern-Oriented Software Architecture - A System of Patterns*, John Wiley & Sons, 1996.

[6] Crnkovic I., "Component-based Software Engineering: Building Systems from Software Components", In *Proceedings of 26th Computer Software and Application Conference (COMPSAC)*, IEEE, 2002.

[7] Crnkovic I., Larsson M., and Lüders F., "Implementation of a Software Engineering Course for Computer Science Students", In *Proceedings of 7th Asia-Pacific Software Engineering Conference (APSEC)*, 2000.

[8] Daniels M., Faulkner X., and Newman I., "Open ended group projects, motivating students and preparing them for the 'real world'", In *Proceedings of 15th Conference on Software Engineering Education and Training (CSEE&T)*, IEEE, 2002.

[9] Dawson R.J., Newsham R. W., and Fernley B. W., Bringing the 'real world' of software engineering to university undergraduate courses, *Software Engineering.IEE Proceedings*, volume 144, issue 5, 1997.

[10] Dawson R., "Twenty Dirty Tricks to Train Software Engineers", In *Proceedings of 22nd International Conference on Software Engineering (ICSE)*, ACM, 2000.

[11] Farbman White S., Melhart B. E., and Lawson H. W., Engineering Computer-based Systems: Meeting the Challenge, *IEEE Computer*, volume 34, issue 11, 2001.

[12] Gamma E., Helm R., Johnson R., and Vlissidies J., *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.

[13] Kazman R., Bass L., Abowd G., and Webb M., "SAAM: A Method for Analyzing the Properties of Software Architectures", In *Proceedings of The 16th International Conference on Software Engineering*, 1994.

[14] Pfleeger S. L., *Software Engineering, Theory and Practice*, Prentice-Hall, Inc., 1998.

[15] Shaw M., We Can Teach Software Better, *Computing Research News*, volume 4, issue 4, 1992.

[16] Shaw M., "Software Engineering Education: A Roadmap", In *Proceedings of 22nd International Conference on Software Engineering (ICSE)*, ACM, 2000.

[17] Wohlin C. and Regnell B., "Achieving industrial relevance in software engineering education", In *Proceedings of 12th Conference on Software Engineering Education and Training*, IEEE, 1999.