An Agile Constructionist Mentoring Methodology for Software Projects in the High School

Orni Meerbaum–Salant and Orit Hazzan

Department of Education in Technology and Science
Technion – Israel Institute of Technology
Haifa 32000, Israel

{orni, oritha}@tx.technion.ac.il


Abstract

This paper describes the construction process and evaluation of the Agile Constructionist Mentoring Methodology (ACMM), a mentoring method for guiding software development projects in the high school. The need for such a methodology has arisen due to the complexity of mentoring software project development in the high school. We introduce the ACMM and suggest a year-long mentoring template that includes the practices required for the actual mentoring process. The evaluation of the ACMM reveals that the methodology addresses each of the challenges teachers cope with during the mentoring process, which were identified in the first phase of the research.

Categories and Subject Descriptors
K.3.2 [Computers and Education]: Computer and Information Science Education – Computer science education, Curriculum.

General Terms
Performance, Human Factors, Management.

Keywords
Computer science education, agile software development, constructionism, Shulman's teacher knowledge base model

---

1 The research described in this paper is a part of the first author's Ph.D. dissertation conducted under the supervision of the second author.
An Agile Constructionist Mentoring Methodology for Software Projects in the High School

1. Introduction

Software development in industry is a complex, problem-ridden process (Brooks, 1987; Hamlet and Maybee, 2001; Tomayko and Hazzan, 2004). In addition to the difficulties inherent in the software industry, additional difficulties arise with respect to software development processes in the high school environment. The research presented in this paper is based on the recognized need for a mentoring methodology – a defined teaching process that supports teachers in guiding their students in software project development processes – which addresses these difficulties. Based on this recognition, a methodology was constructed that involves a variety of knowledge types, as well as management and pedagogical activities.

Specifically, this article introduces the concept of an Agile Constructionist Mentoring Methodology (ACMM) for software development in the high school. The term 'mentoring methodology' refers not only to software programming, but rather encompasses the entire process of mentoring software project development, including pedagogical and management aspects (e.g., time and project management and social practices). To capture this complexity within a single framework, the ACMM combines and is based on theories borrowed from different disciplines: Shulman's Teacher Knowledge Base Model taken from pedagogical studies (Shulman, 1987), constructionism from learning sciences (Papert, 1991), and agile software development from software engineering (Abrahamsson, Salo, Ronkainen and Warsta, 2002; Highsmith, 2002; Williams and Cockburn, 2003). The integration of theories from different disciplines followed the need to provide teachers with answers related to mentoring processes of software development projects in the high school in general, and with solutions to specific difficulties encountered by teachers in the mentoring process itself in particular (Meerbaum-Salant and Hazzan, 2008).

In Section 2, we present the motivation for the development of the ACMM. In Section 3, the research approach employed in the construction and evaluation of the ACMM is detailed. In Section 4, we describe the ACMM in detail, presenting its sources, values, aspects and practices. In Section 5, we present the evaluation findings and in Section 6 we conclude.
2. Motivation for the Development of the ACMM

Software development is a complex process ridden with many problems, such as time pressure, late delivery, changing customer requirements and lack of teammate communication (Brooks, 1987; Hamlet and Maybee, 2001; Tomayko and Hazzan, 2004).

Brooks (1995) notes four differences between software projects and other kinds of projects, each of which renders the management of software projects more difficult than that of other kinds of projects: invisibility of software products, complexity of software products, difficulty in achieving conformity, and software flexibility. Bullock (2003) adds several differences between software projects and non-software projects: deadlines of some software projects are set inaccurately, software development is carried out in stages so the use of part of the software is possible while other parts are still under development, and use is made of multiple methods and tools during the development process.

In addition to typical difficulties encountered by the software industry, additional challenges arise with respect to software development processes in the high school environment. Fincher and Petre (1998) assert that computer science project development is a complicated problem-solving process that requires dealing with multiple problems simultaneously as well as exhibiting supervision and management abilities. In addition, teachers who mentor software development projects are required to create an investigative and responsibility-fostering teaching environment, and in order to help students deal with the different problems encountered, they must also guide students in their learning of the project's various components (Blumenfeld et al., 1991).

This article focuses on projects in which students develop software projects on a topic of their own choice. Student-developed software projects can be viewed as a pedagogical tool from the perspectives of both the teacher and the student. This assertion is derived from the nature of the process during which both teachers and students deal with complex problems, including logical and technical ones, in a variety of creative ways and analysis approaches.

In this spirit, Meerbaum-Salant and Hazzan (2008) suggest that the mentoring of software development projects in the high school is a complex process that involves
teachers’ difficulties such as time scheduling, management of multiple projects developed simultaneously by many students on different subjects, lack of computer science knowledge and difficulties of project evaluation. These difficulties, which were expressed both by teachers who have no previous experience mentoring software development projects as well as by teachers who have such experience, were categorized by Meerbaum-Salant and Hazzan (2008) into four themes: schedule, required computer science knowledge, students’ individual work, and project evaluation. These four themes are addressed in the ACMM evaluation.

As can be seen, from the teacher perspective, the development process of software projects by high school students is a multi-faceted process. However, in spite of the difficulties teachers encounter when mentoring software development projects in the high school, as far as we know, no general methodology has been developed for software development projects in the high school. This fact reinforces the need for a methodology for mentoring software development projects in the high school that will address the difficulties involved in such a process. The ACMM presented in this paper attempts to close this gap.

Several attempts to support the mentoring process in the high school have, nevertheless, been undertaken. Pollak (2002), for example, describes the development and evaluation of learning materials for high school students that support the development process of software projects in logic programming, during both the project scope definition stage and the development of the project itself. Pollak’s guidance is based on the waterfall development model. The waterfall development model is based on several stages (requirements, design, coding, etc), each of which is based on the output of the previous stage and is well defined before continuing to the next step. One main disadvantage of this process is the difficulties to correct, at later stages, errors that were introduced in its early stages. The literature reports many difficulties and problems associated with this life cycle model, such as time pressure and late deliveries (cf. Van Vliet, 2000).

Accordingly, we decided to address the problems encountered by teachers during their mentoring process of software projects in the high school by adapting another paradigm for software project development – the agile approach for software development – which attempts to overcome disadvantages associated with traditional software development methods. Indeed, over the past decade, the software industry
has gradually transitioned to agile software development, and although there is about ten years of accumulated experience using the agile approach, it is currently being applied as the development paradigm by about 20% of companies in North America and Europe.\(^2\) Data indicate that agile software projects cope successfully with common problems of software projects. For example, according to the State of Agile Development survey\(^3\) conducted by Version One and the Agile Alliance in 2007, 60% of respondents estimated a 25% or greater improvement in time-to-market; 55% reported a 25% or greater improvement in productivity; and 55% reported a 25% or greater reduction in software defects. Additional details on agile software development are presented in Section 4.

The agile approach is deemed appropriate to serve as the basis for the sought mentoring methodology because, based on teachers' difficulties identified by Meerbaum-Salant and Hazzan (2008), the needed software developing methodology should provide a flexible management framework (that is, agile) to suit different teaching situations, classes, environments, programming paradigms, programming languages, and teacher characteristics. However, since we deal with mentoring processes of software development projects in the high school, it is not sufficient to simply transfer the agile approach from the software industry to the high school environment, and pedagogical and learning theories should support its transition and design in the high school, specifically, Shulman’s Teacher Knowledge Base Model and constructionism, respectively (see Figure 1). The specific form of transition is described in Section 4.

\(^2\) Source: http://www.versionone.com/pdf/AgileMyths_BetterSoftware.pdf
\(^3\) Agile Development: Results Delivered: http://www.versionone.net/pdf/AgileDevelopment_ResultsDelivered.pdf
3. Research Framework

As mentioned, this paper describes the research in which a mentoring methodology for software projects in the high school – the Agile Constructionist Mentoring Methodology (ACMM) – was constructed and evaluated. The details of the research process are described in what follows.

3.1 Research Field

Data was collected in the following settings:

a. High school computer science classes in which students developed software projects. The projects were developed throughout one academic year and included all of the activities commonly carried out in software development projects, i.e. analysis, design, coding and testing.

b. Six teacher workshops on mentoring software development projects in the high school attended by computer science teachers who were mentoring software development projects or who intend to mentor such projects. The workshops were facilitated by the authors of this paper. In the workshops, the available version of the ACMM was presented to the teachers, analyzed, and refined over the
workshops. The ACMM modifications were later examined based on feedback obtained from the research field.

c. The course Methods of Teaching Computer Science taught to prospective computer science teachers for two semesters at the Department of Education in Technology and Science of the Technion – Israel Institute of Technology.

3.2 Research Participants

The research participants were:

a. Ninety computer science teachers who, at the time, were mentoring high school software development projects.

b. Seven computer science teachers who serve as project evaluators and who were assigned by the Ministry of Education to evaluate software projects in different high schools.

c. Twenty prospective computer science teachers who, at the time, were attending the Methods of Teaching Computer Science course at the Department of Education in Technology and Science of the Technion.

d. Eight computer science teachers who participated in a mentoring workshop facilitated by the authors. Seven of the teachers applied the ACMM throughout an entire academic year. Five teachers applied the ACMM on small-scale projects and five teachers applied the ACMM when mentoring large-scale projects. In addition, one experienced computer science teacher, who participated in one of the mentoring workshops but did not implement it, contributed to the ACMM evaluation.

e. Seventy high school computer science students who developed software projects in high school and whose teachers applied the ACMM.

3.3 Data Gathering Tools

In order to increase research validity and reliability, a variety of data collection tools were used, as follows:

**Questionnaires.** Questionnaires, distributed to the 90 teachers and 20 prospective computer science teachers, were used for the examination of responses of a relatively

---

4 Three teachers applied the ACMM both for small-scale and large-scale projects.
large number of prospective and in-service computer science teachers. The responses to the questionnaire formed the basis for the design of the interviews described below. There was also a questionnaire which was distributed to the 70 computer science high school students who developed software projects mentored by the ACMM.

**Interviews.** In order to improve our understanding of the teachers' perceptions of the mentoring process of software project development, semi-open interviews were held with twelve teachers who, at the time, were mentoring the development of software projects in high schools and with seven evaluators of high school software projects assigned by the Ministry of Education. These teachers provided feedback regarding the actual application of the ACMM during its construction process, during the mentoring process, and following the mentoring process for its evaluation.

**Observations.** Observations were conducted in the following settings:

a. Two computer science classes in which students were developing software projects:
   Each class was observed for two hours a week during the entire school year. Teachers of those classes participated in the first computer science teachers' workshop on mentoring software development projects in the high school.

b. Six of the Methods of Teaching Computer Science course lessons: These lessons dealt with the mentoring of software development projects in the high school. The prospective teachers contributed to the construction of the ACMM from their perspective, both as computer science students and as prospective computer science teachers.

c. Six teachers' workshops on mentoring software development projects in the high school, facilitated by the authors: In each workshop, we demonstrated the current stage of the ACMM to the participants and, based on the teachers’ feedback after each workshop, we refined the methodology according to the participants’ perceptions, work and experiences.

d. Two annual conferences of computer science teachers organized by the Israeli National Center of High School Computer Science Teachers: The first observation was conducted at a round table session facilitated by the authors. The second observation was conducted at the authors’ presentation of the ACMM. In both cases, feedback was collected from the audience of computer science teachers.
e. One high school computer science class for four hours per week during the entire school year: Students in this class developed software projects using one of the final versions of the ACMM.

f. Observations were conducted also by six teachers who applied the ACMM in their own classes. The teachers reported on a regular basis about their perceptions, thoughts, and the way in which they used the ACMM.

**Reflective interviews.** Reflective interviews were conducted with teachers, whose classes were observed, following observations of their classes. The goal of the reflective interviews, in which teachers were asked to reflect on special events observed by the researchers, was to improve our understanding of these special events. The analysis of these interviews examined what modifications, if at all, should be made to the mentoring methodology. There were also reflective interviews which were conducted with teachers who participated in the research while applying the ACMM. In the interviews, the teachers were asked to reflect on special events that occurred during their mentoring experiences in their classes.

**Class work and homework assignments of prospective and in-service computer science teachers.** These completed assignments were collected from the students in the Methods of Teaching Computer Science course and the participants in the teachers' workshops. Analysis of these works helped to refine the ACMM formulation.

**Focus group.** A focus group enables the researcher to ask the group members specific questions about a specific topic after the main part of the research is completed (Merton et al., 1956, cited in Fontana and Frey, 1994). Focus groups are used in project assessments to identify perceptions that are considered to be part of the project's product (Patton, 1990). In the evaluation research described in this paper, a focus group was used to reveal students' perceptions regarding the development process inspired by the ACMM. The focus group consisted of five high school students from one class whose teacher applied the ACMM.

**Researcher diary.** During the study, the first author kept a diary, in which she documented her observations, notes, thoughts, decisions, reflections, and hesitations.

**3.4 Research Process**
Throughout the entire research, and in particular during each of the six research iterations in which the ACMM has been constructed, the data collected so far were analyzed, intermediate formulations of the ACMM were refined if needed, and the exact continuation of the research was determined.

Figure 2 presents the full research process, including the identification of teachers’ difficulties in mentoring processes, and the ACMM construction and evaluation.

4. ACMM – Sources, Values, Aspects and Practices

The ACMM aims both at meeting teachers’ needs with respect to the mentoring processes of software development projects in the high school and at providing tools to cope with different difficulties they encounter during the mentoring process. In the spirit of the agile approach for software development, the ACMM is based on specific
practices, and at the same time is subject to change, flexible, and adaptive to specific class situations and is not limited to a specific programming language or paradigm, student level, curriculum or content.

The ACMM is based on three knowledge sources – agile software development, Shulman's Teacher Knowledge Base Model and constructionism. Our analysis indicates that these three sources share five values – communication, simplicity, feedback, courage and respect – adapted from the agile method Extreme Programming (Beck with Andres 2005). To achieve the ACMM targets, these values are implemented by practices, which are categorized into three aspects – Pedagogical Class Management, Social and Project Management. The need for three aspects emerged due to the unique situation of mentoring software project development in the high school, as described in Section 2.

In the continuation of this section, the ACMM is described by outlining its sources (Section 4.1), values (Section 4.2.), aspects (Section 4.3) and practices (section 4.4). In Section 4.5 we illustrate one possible, actual application of the ACMM. This presentation progresses from a high level of abstraction to a low abstraction level. The practices are presented at this stage (see Table 1) to help capture the ACMM essence.

**Table 1. Practice descriptions**

<table>
<thead>
<tr>
<th>Practice</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pedagogical Class Management Aspect</strong></td>
<td></td>
</tr>
<tr>
<td>Infrastructure establishing</td>
<td>The teacher builds the technical infrastructure in the class and establishes the didactical contract between teacher and students, which refers to the behavior and rules in the project development environment. These rules should reflect the ACMM values.</td>
</tr>
<tr>
<td>Project demonstration</td>
<td>During the development process, the teacher demonstrates parts of previously developed projects to illustrate and highlight important issues students should consider at specific stages of the development process.</td>
</tr>
<tr>
<td>Teaching the content knowledge</td>
<td>Teaching the material required for project implementation.</td>
</tr>
<tr>
<td>Individual mentoring</td>
<td>The teacher mentors the students on the individual level to help each student cope with his or her difficulties and proceed in the development process.</td>
</tr>
</tbody>
</table>
### Subject group mentoring

This mentoring activity concerns groups of students with similar project topics who can learn from each other. Mentoring meetings can include explanations given by the teacher or just the students themselves, dealing with the same challenges and difficulties.

### Formative assessment

This assessment is carried out by the teacher for learning purposes, during the entire project development process, with respect to each activity the student performs and takes part in.

### Summative assessment

The teacher performs summative assessment several times during the year, usually at the end of specific stages, to monitor student and class progress.

### Social Aspect

| Mutual assistance and information sharing | The students are required to help each other. This help can be reflected in code and knowledge sharing and in the sharing of ideas. |
| Peer assessment                          | Students are divided into groups in which students mutually evaluate and offer feedback on each others projects. |

### Project Management Aspect

| Project story | Each student defines a project story that describes the project scope, goals and functionality. Students receive feedback from their classmates and from the teacher. The story helps the teacher evaluate whether the project scope and requirements suit the student's abilities and available development period, and meet the requirements of the final evaluation. |
| Analysis      | Each student analyzes the project structure and divides it to modules. |
| Design        | Each student designs his or her project components and determines the needed infrastructure. |
| Programming   | The student programs the project in stages. In each stage, the student first decides how the code to be developed is to be tested, considers problems which might emerge and tries to address them during the programming activity. Then, the student programs the said functionality and checks it. In each stage, the code is integrated and checked. That means that at each stage, the student has a version of a running product to present. This practice enables to deal with problems at early stages of the development process. It also enables to develop software that meets the requirements. |
| Testing       | Testing comprises three testing cycles:  
  a. Before each code programming, the student prepares a testing }
list that includes all problems that he or she predicts might occur.
b. At the end of each code development, the code is tested both by the student and the teacher. Its integration with the existing code is also tested.
c. At the end of the project development, both the teacher and the external evaluator check and evaluate the project implementation.

| Documentation | The student documents the development stages as well as the project's main attributes and features. Documentation is kept to a minimum and it aims at explaining the project rationale and its main development stages. |
| Refactoring | Refactoring activities refer to code improvements without adding functionality. This practice is relevant in the school environment as it is relevant in the industry, since the students develop their projects in stages and improve their understanding of the code structure as they proceed. |
| Individual feedback | Each student provides self-feedback on his or her work and on the way he or she met the planned schedule that the teacher set for the entire class. |

4.1 ACMM Sources

As mentioned, the ACMM is based on three knowledge sources: *Shulman's Teacher Knowledge Base Model* taken from the field of pedagogy (Shulman, 1987), *constructionism* taken from the learning sciences (Papert, 1991), and *agile software development* taken from the field of software engineering (Abrahamsson, Salo, Ronkainen and Warsta, 2002; Highsmith, 2002; Williams and Cockburn, 2003; Hazzan and Dubinsky, 2008).

4.1.1 Shulman's Teacher Knowledge Base Model

Shulman's Teacher Knowledge Base Model focuses on structural knowledge that is common to all teachers. It consists of seven categories: Content knowledge, General pedagogical knowledge, Curriculum knowledge, Pedagogical content knowledge (PCK), Knowledge of learners and their characteristics, Knowledge of educational contexts and Knowledge of educational ends. These categories reflect the interconnection between teachers' cognitive understanding of a subject matter and their teaching methods.
According to Shulman (1987), among the seven categories, Pedagogical Content Knowledge (PCK) is of special interest. This component refers to the special amalgam of pedagogy and content that reflects the teacher's own professional understanding of the subject matter. It includes the organization and representation of the material in ways that render it comprehensible by learners, using analogies, illustrations, examples and explanations.

Since there is more than one appropriate way to teach and explain a certain topic, teachers must be familiar with different teaching approaches, parts of which are based on their own professional experience. According to Shulman (1987), the teachers must understand a subject matter in a deep and flexible manner so they can help students create useful cognitive maps, relate one idea to another, and address misconceptions. In addition, teachers must see how ideas relate to different fields and to practical life. This kind of understanding provides a foundation for PCK that enables teachers to make ideas accessible to others.

Naturally, PCK is considered to be highly importance in complex teaching situations, such as the one explored in this paper – mentoring software projects in the high school.

### 4.1.2 Constructionism

Papert (1991) expands the constructivism learning theory by adding the need for an external concrete construction in learning processes to support mental constructions. Thus, the idea of constructionism is "learning-by-making" and is based on two different senses of "construction": the traditional constructivist approach, which asserts that people learn by actively constructing new knowledge, and the construction of a public entity, whether it is a sand castle on the beach or a theory of the universe. Examples of constructionist learning environments are quantitative measurements while baking a cake, building with Lego and working with the computer programming language LOGO (Papert, 1993).

Seungyeon and Kakali (2007) connect constructionism with problem-based learning (PBL) environments. Since the development of software projects in the high school is an example of PBL in the context of computer science education, we highlight this connection. According to Seungyeon and Kakali (2007), constructionism is expressed in PBL environments by creating a student-centered learning environment and by
placing an emphasis on the creation of an artifact as part of the learning outcome, which is authentic and based on real-life experiences. Hence, learners are allowed to become active builders of knowledge while confronting misconceptions and internalizing content and associated conceptions.

Similar to other PBL environments, the teacher's role in high school software project development processes is different than in situations in which theoretical computer science concepts are taught. In the former situation, teachers are required both to teach the programming paradigm and language, to mentor the project development process and to evaluate the students' projects. In addition, Waks (1997) asserts that the focus of the teacher's role must be modified in PBL environments from teaching activities to learning activities, by establishing conditions that enhance students' curiosity and motivation. The learning environment must be based on experimental investigation-oriented learning that significantly reduces the teacher's criticism and judgment of the students' work. Furthermore, educational mentoring requires the teacher to exhibit authoritative behavior and vast knowledge of different disciplines and of the relations between them. In other words, the teacher must act as a leader and a mentor and less as a decision maker.

The above characteristics of teachers' behaviors in PBL situations yield positive results from the learners' perspective. PBL develops thinking practices, independent learner abilities (Krajcik, Czerniak and Berger, 1999; Barak, Waks and Dopplet, 2000), motivation, security and self confidence (Green, 1998; Frank, Lavy and Elata, 2003; Barak, Waks and Dopplet, 2000; Shepherd, 1998), classmate cooperation (Frank, Lavy and Elata, 2003; Krajcik, Czerniak and Berger, 1999), and an integrative understanding of the content as well as of the process (Krajcik, Czerniak and Berger, 1999; Barak, Waks and Dopplet, 2000). These results are reasonable, since PBL enables to adapt the variety of tasks possible for implementation in PBL environments to each learner's learning style (Krajcik, Czerniak and Berger, 1999). This claim further supports the perspective according to which PBL constitutes a constructionist learning environment.

As can be observed, the teacher's role in mentoring students in PBL is a complex process that involves many difficulties and, as our research indicates (Meerbaum-Salant and Hazzan, 2008), in the case of software development also requires a variety of knowledge areas. Furthermore, in addition to the above general characteristics of
teachers’ behavior in PBL environments, computer science teachers face additional challenges when mentoring students in software project development. This is due in part to the fact that, in addition to the problem-solving process shared by all PBL environments, the solution in a software development process must also be implemented by a software product. These considerations are addressed by incorporating the agile approach for software development into the ACMM to be applied in PBL environment of software project development.

4.1.3 Agile Software Development

In the earliest days of programming, computers were used for computations that took a long time to do manually. This type of development called for code-and-fix or ad-hoc program development. The requirements were relatively stable and the product development cycles were often measured in years (Williams and Shukla, 2002).

The code-and-fix software was written without any substantial plan. As the systems grew, it became more and more difficult to add new features and to fix them. This stage was known as the software crisis. As a result, methods began being formulated to overcome these problems. These methodologies, influenced by the engineering approach of tangible products, caused software development to become more predictable and more efficient. These early development methods are usually referred to as plan-driven methodologies (Fowler, 2005). As it turns out, and as mentioned in Section 2, such methodologies fail to solve most of the problems that characterize software projects.

During the past decade, as a reaction to the problems that still exist with plan-driven methodologies, the agile movement has emerged, offering an alternative approach to software project development (Ilieva, Ivanov and Stefanova, 2004; Fowler, 2005; Hazzan and Dubinsky, 2008). Agile methods offer a compromise between not enough process and too much process. Agile methods are adaptive rather than predictive and, accordingly, they welcome change, which is an integral part of any software development process. Agile methods are also people-oriented rather than process-oriented, which means that the role of a process is to support the development team in its work (Fowler, 2005). Highsmith (2002) asserts that problems characterized by change, speed, and turbulence are best solve by agility. The Agile Manifesto (Table 2), formulated in 2001, captures the main ideas of agile software development.
Table 2. The Agile Manifesto (Agile manifesto at http://agilemanifesto.org)

<table>
<thead>
<tr>
<th>Manifesto for Agile Software Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>We are uncovering better ways of developing software by doing it and helping others do it.</td>
</tr>
<tr>
<td>Through this work we have come to value:</td>
</tr>
<tr>
<td>Individuals and interactions over processes and tools</td>
</tr>
<tr>
<td>Working software over comprehensive documentation</td>
</tr>
<tr>
<td>Customer collaboration over contract negotiation</td>
</tr>
<tr>
<td>Responding to change over following a plan</td>
</tr>
<tr>
<td>That is, while there is value in the items on the right, we value the items on the left more.</td>
</tr>
</tbody>
</table>

The agile approach is applied by different methods, including DSDM (Dynamic Systems Development Method), FDD (Feature Driven Development), XP (Extreme Programming), Crystal, ASD (Adaptive Software Development), and Scrum, all of which share the following principles⁵:

- Customer satisfaction is achieved by early and continuous delivery of valuable software;
- Changes in requirements are welcome;
- Working software is delivered frequently, within short time scale;
- Business people and developers cooperate on a daily basis;
- Projects are built around motivated individuals who are trusted;
- Face-to-face communication in the development team;
- Working software is the primary measure of progress;
- Process promotes constant pace development;
- Continuous attention to technical excellence and good design;
- Simplicity;
- Heavy testing throughout the entire development process;
- Regular adaptation to changing situations.

⁵ This list is based on Principles of Agile Software at http://agilemanifesto.org/principles.html.
Though, as mentioned previously, the agile approach is implemented by different methods, the ACMM is based primarily on one agile method – Extreme Programming XP (Beck with Andres, 2005) – for two main reasons. First, XP is based on a set of well-defined practices whose application in the school environment can each be examined separately. Second, XP is based on five values – communication, feedback, simplicity, courage and respect – which are implemented by the XP practices. Our analysis revealed that these values are also shared by the other two ACMM sources – Shulman's Teacher Knowledge Base Model and constructionism (see next section). Thus, by basing the ACMM on this value set, the ACMM constitutes a value system whose importance in educational environments is evident.

4.2 ACMM Values

In what follows, we explain how each of the XP values – communication, simplicity, feedback, courage and respect – is expressed by the ACMM sources in general, and by the ACMM in the school environment in particular. The expression of the values by the ACMM is illustrated by the ACMM practices presented in Table 1.

**Communication**

*Agile software development:* According to Beck (with Andres, 2005), communication is one of the most important factors of software teamwork for the creation of effective cooperation. Tomayko and Hazzan (2004) highlight additional important communication channels in software projects, such as those that exist between management and the development team and between the customer and the development team.

*Constructionism:* Communication is expressed by the interaction that occurs during the mental and external construction processes, which fosters creativity, knowledge sharing and problem-solving processes.

*PCK:* Communication is expressed by different pedagogical methods and tools that a teacher employs, such as forums, group work, and content knowledge sharing.

*ACMM:* During the mentoring process, communication between the mentor/teacher and his or her students and between the students themselves is fostered, for example, by group work, collective content knowledge and peer review.

**Simplicity**
Agile software development: According to Beck (with Andres, 2005), simplicity is required in software projects to cope with complex problem-solving situations, which characterize software development environments.

Constructionism: Simplicity is expressed by the acknowledgment that learning processes are gradual and therefore are based on learners’ coping with a gradual increase in problem complexity as well with a gradual complexity of the external construction carried out in parallel to the learning process.

PCK: Simplicity is expressed by the teacher's awareness of the importance of establishing a teaching environment that aims at simplifying and supporting learning processes.

ACMM: Simplicity is expressed in the mentoring process by defining clear and simple rules to create a working environment that supports learners’ development processes. Simplicity is also reflected in the decomposition of the project into modules, which makes it possible to deal with each module separately. In addition, the practice of refactoring renders the code simpler and clearer, and testing helps control the project's continuous development. Simplicity is also implemented in group learning, which fosters information sharing between students, which in turn simplifies the mentoring environment and thus helps the teacher deal with a variety of student projects and with a variety of questions raised during the mentoring process. This is possible since, in such a learning environment, the teacher is not the only knowledge resource.

Feedback

Agile software development: According to Beck (with Andres, 2005), feedback is used to achieve the team's goals. Feedback can be expressed, for example, as an opinion about an idea, as checking of code that implements the idea, or as checking of the idea after its deployment. Feedback is a critical part of communication and it contributes to simplicity - the simpler the system is, the easier and simpler it is to get feedback on it.

Constructionism: Feedback is received from the teacher, the students, and most importantly, from the actual construction, which can be seen as personal feedback. All these feedbacks promote students' construction of knowledge.
**PCK**: Feedback in the school context in general, and within the PCK in particular, refers to the approaches to teachers’ assessment of students' work, which aim to foster learning processes (Shepard, 2000).

**ACMM**: The value of feedback in the ACMM is expressed by several kinds of feedback: peer assessment and peer code review, teacher feedback in the form of both summative assessment and formative assessment, feedback from the code testing and continuous code integration and, finally, feedback from the evaluator on behalf of the Ministry of Education. The teacher uses the feedback to track the progress of the students’ projects and to promote students' learning and understanding.

**Courage**

**Agile software development**: According to Beck (with Andres, 2005), courage is effective in the face of fear. Courage can be expressed by having the patience to deal with a real problem that might emerge and to speak truths – pleasant or unpleasant ideas. Accordingly, courage fosters communication and trust. Furthermore, the courage to forego failing solutions and seek new ones encourages simplicity and the courage to seek real and concrete answers enhances feedback.

**Constructionism**: Courage is expressed by the need to deal with new teaching and learning situations from both the students' and the teachers' perspective. This occurs in constructionist learning environments since it is impossible to a-priori predict the exact path of the learning process and both the teacher and learners must be able to cope with such unknown and unfamiliar situations.

**PCK**: Courage is expressed by teachers, who deal with different teaching styles and by doing so enable their students to experience new learning situations, including coping with failures, information sharing and group learning.

**ACMM**: Courage is applied by the ACMM in that the class learns together as a team and the students learn from each other. In such situations, students are encouraged to admit it if they do not know some material and if they need help and support. Furthermore, the students’ peer review is based on their own courage to give and receive feedback.

**Respect**
Agile software development: According to Beck (with Andres, 2005), the value of courage is based on the idea that no one is worth more than anyone else and that each team member's contribution to the development process should be respected.

Constructionism: Respect is expressed by the acknowledgment and legitimization that the constructionism theory gives to learning from mistakes and to the individual’s self-expression of creativity and problem-solving approach.

PCK: The respect value is reflected in learning environments in which students are encouraged to ask questions, help each other, and give feedback to each other. In such learning environments, the teacher trusts and respects answers presented by students in the class.

ACMM: The ACMM reflects the value of respect in the class environment, in the relationship between the teacher and the students as well as in the relationship among the students themselves. In particular, everyone in the class respects the others’ learning process and promotes the learning, development process and understanding of the other classmates.

4.3 ACMM Aspects

The five values – communication, simplicity, feedback, courage and respect – are implemented by seventeen practices, presented in Table 1. The practices are grouped into three aspects: pedagogical class management, social and project management, each of which includes several practices that are relevant to it. The pedagogical class management and social aspects were added to the project management aspect, which focuses directly on the actual software development, due to the unique situation of mentoring software project in the high school, as described in Section 2.

Figure 3 reflects the contribution of each of the three ACMM sources to the ACMM aspects. As can be seen, while the agile approach contributes only to the project management aspect, constructionism and Shulman's Teacher Knowledge Base Model contribute both to the pedagogical class management and social aspects. This observation explains Figure 1, which reflects the need for learning and teaching theories in the adaptation process of agile software development from industry to the high school learning environment.

Figure 3 also indicates that the constructionism source contributes to the project management aspect which deals with the actual construction of the software project.
This observation can be explained by the perspective according to which software development processes are learning processes in which both customer and developers improve their understanding of the developed application, and therefore, development methods should support these learning processes (Hazzan and Dubinsky, 2003).

Figure 3. ACMM's sources and aspects

4.4 ACMM Practices

The ACMM practices are presented in Table 1. Table 3 presents the classification of the ACMM practices into the three aspects. We note that since the practices are based on the ACMM sources, which share the five values as explained in Section 4.2, the practices themselves reflect the ACMM values.
Table 3. The ACMM practices by aspects

<table>
<thead>
<tr>
<th>Pedagogical Class Management Aspect</th>
<th>Social Aspect</th>
<th>Project Management Aspect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infrastructure establishing</td>
<td>Mutual assistance and information sharing</td>
<td>Project story</td>
</tr>
<tr>
<td>Project demonstration</td>
<td>Peer assessment</td>
<td>Analysis</td>
</tr>
<tr>
<td>Teaching the content knowledge</td>
<td></td>
<td>Design</td>
</tr>
<tr>
<td>Individual mentoring</td>
<td></td>
<td>Programming</td>
</tr>
<tr>
<td>Subject group mentoring</td>
<td></td>
<td>Testing</td>
</tr>
<tr>
<td>Formative assessment</td>
<td></td>
<td>Documentation</td>
</tr>
<tr>
<td>Summative assessment</td>
<td></td>
<td>Individual feedback</td>
</tr>
</tbody>
</table>

4.5 ACMM Application, Organization and Illustration

In order to implement the ACMM, it is not sufficient to present the ACMM sources, values, aspects and practices; a time schedule should be offered as well. Table 4 presents one possible way of implementing the ACMM within a given period of time. The timetable presented in Table 4 suits specific kinds of projects intended for development during a single academic year. Similarly, variations can be made to accommodate different teaching and learning conditions.

Table 4. An illustrative implementation of the ACMM

<table>
<thead>
<tr>
<th>ACMM Practices</th>
<th>Aug</th>
<th>Sep</th>
<th>Oct</th>
<th>Nov</th>
<th>Dec</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
<th>June</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pedagogical Class Management Aspect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Infrastructure establishing</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project demonstration</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Teaching the content knowledge</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual mentoring</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Subject group mentoring</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Formative assessment</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Summative assessment</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Social Aspect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mutual assistance and information sharing</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Peer assessment</td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project Management Aspect</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Project story</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Analysis</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Design</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Programming</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Testing</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Documentation</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Refactoring</td>
<td></td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Individual feedback</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

23
5. The ACMM Evaluation

In this section, we present the ACMM evaluation by showing how each of the difficulties encountered by teachers, as described by Meerbaum-Salant and Hazzan (2008), is addressed by at least one ACMM practice. In this respect, we view the evaluation process as a "mathematical proof". From this perspective, the problem was to design a mentoring methodology that answers specific needs. After the methodology was constructed, that is, an object – the ACMM – that meets specific conditions was defined, its evaluation (presented in this section) “proves” that it meets the needs for which it was designed. In other words, from the proof-oriented perspective, the ACMM evaluation shows that it fulfills the conditions for which it was constructed.

Table 5 presents which of the ACMM aspects addresses each teacher difficulty by specifying the relevant practices that address each difficulty.

<table>
<thead>
<tr>
<th>Difficulty Aspect</th>
<th>Schedule</th>
<th>Required CS knowledge</th>
<th>Students' individual work</th>
<th>Projects evaluation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pedagogical class management aspect</td>
<td>• Infrastructure establishing</td>
<td>• Project demonstration</td>
<td>• Individual mentoring</td>
<td>• Formative assessment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Teaching the content knowledge</td>
<td></td>
<td>• Summative assessment</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Subject group mentoring</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Social aspect</td>
<td>• Mutual assistance and information sharing</td>
<td></td>
<td>• Peer assessment</td>
<td></td>
</tr>
<tr>
<td>Project management aspect</td>
<td></td>
<td>• Project story</td>
<td></td>
<td>• Individual feedback</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Analysis</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Design</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Programming</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Testing</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Documentation</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Refactoring</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
In what follows, illustrative evidence for the classification presented in Table 5 are presented. Specifically, for each teacher difficulty, illustrative evidences that relate to the practices that address the said difficulty are presented. These excerpts were expressed by the research participants, both teachers and students who applied the ACMM, in either the interviews, the focus group, the observations or the questionnaires, in most cases without explicit indication of the practices by the interviewer. For the reader convenience, we briefly describe each practice before its illustration. See Table 1 for the full practice descriptions.

5.1 Schedule

*Establishing the infrastructure*: This practice guides the teacher to build the technical infrastructure in the class and to establish a didactical contract between the teacher and students. The following quote highlights the importance a teacher attributes to the ordered schedule that the ACMM lays out:

> I used the model when I was planning my yearly schedule. In my opinion, one of the model's advantages is its template. It is good for us, as teachers, to have a template with which we can work, and we can always test ourselves and check where we are at each point relative to the schedule. I think that the model has great merit that it can always be develop and changed and you don't have to adhere to it strictly, but it gives you the main general guidelines.... This year, .... I finished mentoring the projects at an early stage than in previous years.

5.2 The Required CS Knowledge

*Project demonstration*: This practice guides the teachers to demonstrate in class previously developed projects. The application of this practice is reflected, for example, in the following quote, in which a teacher explains in an interview how she demonstrates projects to her class:

> At each software development stage, I demonstrated projects from previous years to illustrate to the students what they need to do.

It is also reflected in the following quote, expressed by one of the focus group students:
When students from last year came to talk and to demonstrate their projects, they explained to us what we needed to do, and I was less stressed and thought to myself that if they did it, so could I.

**Teaching the content knowledge:** This practice relates to the actual teaching of the material required for the project implementation. In the following quote, a teacher explains in an interview how he taught the material in his class:

*I teach the theoretical material that all students require while implementing the projects. The students are required to study parts of the material that are specific to their project by themselves. They teach each other. I help them find good and helpful resources ... such as specific chapters from books, internet websites, etc. They must write examples that implement the material they have learned so that other students can use it.*

The following quote, expressed by one of the focus group students, highlights this practice from the students' perspective:

*I think that the learning process was very helpful for me. We learned parts of the material by ourselves and each of us developed his or her own software in different fields... We helped each other understand the new material and thus we learned the software, how to work with it and how to use it. The teacher also used a presentation with explicit explanations while he was teaching, and I really liked it because it was accessible for us and we could use the presentation each time we needed it.*

**Subject group mentoring:** This practice guides the teachers to mentor activities that concern a group of students with similar project subjects to deal with similar challenges and difficulties. The following quote, in which a teacher explains in an interview how he mentored activities in his class, addresses this practice:

*After I identified several students who were dealing with the same subject, I gathered them together and taught them the specific subject.... I gave each student the specific chapter that relates to his or her project ... and that was very helpful. They were required to learn the material and to write examples explaining the new material they had learned so that the other students could use it in their projects.*

One of the students in the focus group also described the benefits of this practice:
When I realized that there were several other students whose projects related to my project, I knew that I could consult with them. There was material that we studied together and it was very helpful for us in dealing with the difficulties.

**Mutual assistance and information sharing:** This practice guides the teacher in encouraging students to help each other. The following teacher's excerpt describes how a teacher encourages the students to do that:

> I have a directory that I made accessible to all students. The directory is called "Examples"... and if, for instance, someone wants to know how to do something, he or she accesses the Examples directory and looks for the right example that can help. Besides the material that I put in the Examples, when the students learned something by themselves, they were required to write examples of what they had learned and to add them to the repository. In this way, the repository grew bigger and bigger and everyone could use it.

The students also observed the advantages of this practice, as is reflected by the following quote, given as a reflection by a student whose teacher used the ACMM:

> I used the Examples directory a lot in order to understand some of the materials and also to write part of the project. The requirement to add examples to the Examples directory, helped me understand things more thoroughly.

**5.3 Individual Work of Students**

**Individual mentoring:** This practice guides the teacher to mentor his or her students on the individual level. This idea is reflected, for example, in the following quote, in which a teacher explains in an interview how he mentored the students in his class individually:

> I also needed more personal mentoring. I truly believe in that personal mentoring helps more than general mentoring or collective mentoring. There must be personal mentoring in order to touch the student's most specific issues. It helps the students understand and also provides them with support and helps them be more focused.
The students also acknowledged the importance of this practice, as is expressed in the following quote, given by a student who participated in the focus group:

*The personal meetings with the teacher advanced me and helped me a lot; I acquired a new way of thinking and more efficient solutions. I learned to "talk" with the computer and to think systematically.*

**Project story:** This practice guides the teacher to request each student to define a project story that describes his or her project scope, goals, and functionality. In the following quote, for example, a teacher explained in an interview how she requested her students to define a project story:

*Writing the "story" makes the students write exactly what they want to develop. It helps them focus their thoughts. Writing down the "story" also helps me by setting limits to the project and it provides a general view about whether the "story" has enough “meat” on it and whether it fulfills the requirements. Writing the story also helps me by adjusting the project to the student's abilities.*

The contribution of this practice to the project is also addressed by the students, as is described in the following quote, given as a reflection by a student:

*Writing the "story" caused me to focus my thoughts and the teacher helped me focus even more. There are things that she [my teacher] omitted because she said it was too much and that it complicates things, and there are things that she [my teacher] suggested that I delve deeper into.*

**Analysis:** This practice guides the teacher to require each student to analyze his or her project structure. In the following quote, for example, a teacher's explains the usefulness of this practice:

*The analysis stage accurately defined the tasks for each student and what exactly he or she has to do, so the students were more focused on the task.*

The importance of this practice is also addressed by the following quote, given as a reflection by a student:

*In my opinion, the experience of writing the project was a little different even though I had experienced it before. We divided the complex project into its parts ... and it helped us understand what we had to do.*
**Design:** This practice guides the teacher to ask each student to design his or her project components. This practice is described, for example, in the following quote, in which a teacher explains in an interview how she used this practice:

*I think that this kind of model [the ACMM] is easy and preferable because it provides the student with a general vision of the design from the very beginning. From the beginning, the student starts to see how it [the software] is developed and can design it in light of the final results.*

The value of this practice is also seen from the students' perspective, as is illustrated in the following quote, given as a reflection by a student.

*The project that I wrote was divided to several stages, which helped obtain the correct design.*

**Programming:** This practice guides the teacher how to mentor the students during the actual programming. The actual application of this practice is described, for example, in the following quote, in which a teacher explains in an interview how he guides the students' programming processes:

*Project development was done according to an "onion skin" model. In other words, they developed a small primary project that works, and based on it, they added new add-ons, examining the requirements and making sure that it works. Then, they continued to grow the project. This method helped me and the students a lot and encouraged them, since they could see a visual product from the very first stage - which is very enjoyable for them, opens up a variety of improvement options for them and allow them to improve and raise the sophistication of the project all the time. They could see it in front of them and they could examine and check each little addition.*

The contribution of this practice to the project's success is also observed by the students as is reflected in the following quote, given by a student in the focus group.

*Dividing the project into stages improves the efficiency of the process. Particularly, when you know that you have a good basis that works, you then build or add something and examine it each time. And then if there is an error ... I don’t have to go back ... because I know that they [Prolog rules] work since I tested them. And I simply know that the new rule [in Prolog] that I added is no good and must be fixed.*
**Testing:** This practice guides the teacher to adapt three testing cycles. The practice of testing is emphasized by the teachers as is illustrated, for example, in the following quote, in which a teacher explains in an interview how she fostered testing in her class:

*I emphasized the development in stages while testing, and it undoubtedly proved itself. They had to test each part, implement it and test it again. I paid a lot of attention and emphasized it and it worked beautifully. It was very easy to follow and control the stages and the required tasks.... The tests will save you thought in the future. ... It means, the more you test while programming, ... surely not afterward ... it develops their way of thinking ... and so, their project is also developed faster.*

The contribution of testing to students' progress in general, and to their morale in particular, is expressed for example in the following quote, given by a student in the focus group:

*Each time we wrote just a part of it [the project] and only after it worked did we expand it. This method really helped me since I always knew that I just have to test the last part I added. The fact that there is always something that works gave me a good feeling that I already had part of the project which actually works.*

**Documentation:** This practice guides the teacher to teach the students to document the development stages as well as the project's main attributes and features. The actual implementation of this practice is presented, for example, in the following quote, in which a teacher explains in an interview what documentation style he requires from his students:

*The truth is that I hate to document by myself. Sometimes it is too much and it is not needed, so I paid attention that there should be minimal documentation of the important actions and variables.*

The students, as it turned out, grasped the essence of the needed documentation, as is reflected for example in the following quote, expressed by a student in the focus group.

*We were requested to add minimal documentation that explains, in general, the main purpose of each part and the meaning of each variable.*
**Refactoring**: This practice guides the teacher to request the students to make improvements without adding functionality. The following quote expresses a teacher's explanation of the rationale of refactoring:

*When the students finish their modules, I have always taught them that they can refactor the code to be more simple, clear and more efficient.*

The students also addressed the practice of refactoring, as is expressed, for example, in the following quote, given by one of the students in the focus group.

*Sometimes, the teacher read the code and gave us pointers on how to improve the code and write it more efficiently and clearly.*

### 5.4 Project Evaluation

**Formative assessment**: This practice guides the teacher in assessing the students' development processes. The contribution of this practice to the teacher's work is elaborated, for example, in the following quote, in which a teacher explains in an interview how she uses her students' assessment to improve her own teaching:

*At the end of each assignment, I collected recurring issues, that is issues that might be interesting and important for everyone, ... or things that show a basic lack of understanding, or a certain concept that they did not get properly and I explained it again... They handed in homework and also parts of the project, which I constantly checked and commented on.*

The importance of this practice to the students' learning process is illustrated by the following quote, given by a student as a reflection.

*Submitting the project's stages and receiving comments for improvement were very helpful to me and I learned a lot from it, and I knew each time that what I had corrected is surely right and works well.*

**Summative assessment**: This practice guides the teacher to perform summative evaluation several times during the year. The actual contribution of the application of this practice to the teachers' work is outlined, for example, in the following quote:

*I gave the students a deadline and they submitted the document. Then I gave them some feedback and they improved the document and sent it back to me and I graded it. I knew that next time, I would not have to go through it and correct it from the beginning. It really helped me and made my work more
efficient and alleviated the pressure I was under, relative to previous years, in which I had to check each project from the beginning.

The contribution of this practice to students' work is illustrated by the following quote, given by a student on the focus group.

_Handing in parts of the project on specific dates organized my work and forced me not to postpone everything to the last moment. And after each step, we received our grades and we had the opportunity to fix it. Retroactively, it was fun that we finished everything before the chaotic exam period when we would not have had time to do it all._

**Peer assessment**: This practice guides the teacher to divide students into groups, in which they offer feedback on each others' projects. The influence of this practice on class atmosphere is illustrated in the following excerpt from a teacher's interview:

_Everyone evaluated the others, they worked together, corrected each other together, one gave the other the code he needed and they also had each other’s code, each one took the code, played with it, checked it, told him if it was not enough, and so on - and all that was very helpful. In their presentation in class, in front of the rest of the students, receiving their friend’s comments, they thought about it again and maybe corrected and improved it. Everyone enriched the others._

The importance of this practice to the students' learning is also acknowledged by the students as is reflected, for example, in the following quote, given by a student in the focus group.

_My classmates’ consultations and different opinions exposed me to different ways of thinking, to knowing how to receive criticism from someone else, and to using it in order to improve my project and make it better._

**Individual feedback**: This practice guides the teacher to guide each student to provide self-feedback on his or her development process. The actual influence of this practice is reflected, for example, in the following quote taken from an interview with one of the teachers:

_Every student has a yearly schedule and everyone knows where they are on it. We checked where we were each month. Each one had to indicate where he or
she was standing and had to know where it was, and this is very helpful. It put some pressure to continue on those who were already late, and those who had met the schedule felt more comfortable.

The students also realized the importance of this practice as is reflected, for example, in the following quote, given by a student in the focus group.

*I was strict about meeting deadlines and about always handing everything in on time. You constantly see where your friends are and what they have already accomplished, and sometimes you get ideas, add, and improve.*

6. Conclusions and Discussion

This article presents a construction and evaluation study of a mentoring framework – Agile Constructionist Mentoring Methodology (ACMM) – that addresses difficulties encountered by teachers in the mentoring process of software development projects in the high school. The ACMM is based on three sources: agile software development, which addresses the actual software development project; constructionism, which takes learning processes into consideration; and Shulman's Teacher Knowledge Base Model (and its Pedagogical Content Knowledge component in particular), which adds the pedagogical perspective to the mentoring methodology. Our analysis indicates that these three resources share five values – communication, simplicity, feedback, courage and respect – adapted from the agile method Extreme Programming (Beck with Andres 2005). To achieve the ACMM targets, these values are implemented through practices, which are categorized into three aspects – a pedagogical class management aspect, a social aspect, and a project management aspect. The need for three aspects emerged from the unique situation of mentoring software development projects in the high school.

In this paper we show how the three aspects, together with their practices, offer solutions to teachers' difficulties according to four themes emerged from the data analysis: schedule, the required CS knowledge, individual work of students, and project evaluation. Thus, the ACMM has the potential to provide solutions for teachers' difficulties that are grounded in the field. More specifically, the practices, as well as the exact timetable that is constructed for each learning environment, help the teacher in the mentoring process and provide answers to the following three questions: What should I do during the mentoring process? When should I do what?
And how should I do it?. A similar idea is presented by Dubinsky and Hazzan (2005) with respect to software engineering and computer science undergraduate learning environments.

We end the paper by noting that the ACMM illustrates the concept of a didactic transposition, which refers to the adaption process of knowledge used by practitioners for teaching purposes (Chevallard, 1985). The term has been coined in the context of mathematics education, in which it refers to the process by which formal mathematics is shaped to fit school teaching. For example, the introduction of proof with two columns, for statement and reasons, represents "a didactic transposition from abstract knowledge about mathematical proofs" (Kang and Kilpatrick, 1992, p. 3). Accordingly, the ACMM can be described as a didactic transposition of agile software development for several reasons.

First, the ACMM illustrates how a successful methodology applied in the software industry can be adapted to the school environment when the required learning and teaching considerations are applied. These considerations include the construction of the ACMM based on three sources (two of which are pedagogical and cognitive) as well as the categorization of the ACMM practices into three aspects – a pedagogical class management aspect, a social aspect, and a project management aspect – that the need to include them in the ACMM resulted from the unique and complex situation of mentoring software development projects in high school. Second, the project management aspect of the ACMM includes agile practices that the teacher guide the students how to carry out; their mere positioning within a boarder mentoring framework is an apparent indication that the ACMM can be viewed as a didactic transposition. Third, the ACMM evaluation shows how the three aspects of the ACMM, together with its practices, offer solutions to teachers' difficulties according to four themes emerged in the first research stage: schedule, the required CS knowledge, individual work of students, and project evaluation. Thus, the ACMM, as a didactic transposition, has the potential to provide solutions for teachers' difficulties that are grounded in concrete teaching situations.

**Acknowledgments**

We would like to thank the Technion's MANLAM (Executive Vice President for Research) and the S. Faust Research funds for their generous support of this research.
References


