Agile Projects in High School Computing Education – Emphasizing a Learners' Perspective

Ralf Romeike
University of Potsdam
August-Bebel-Str. 89
14482 Potsdam, Germany
romeike@cs.uni-potsdam.de

Timo Göttel
University of Hamburg
Vogt-Kölln-Str. 30
22527 Hamburg, Germany
tgoettel@acm.org

ABSTRACT
Software projects are seen as a methodology for secondary computing education which is highly appropriate and meets the demands and goals of Computer Science (CS). Yet the majority of models and examples for project-based lessons rely on a traditional software development approach: the waterfall model. In this paper such models are analyzed for their strength, problems, and deficiencies. Based on the results of the analysis a new approach to projects in secondary computing education is presented which uses the concept of didactic transposition to adapt agile software development methods for project organization, management, and implementation in class. The resulting model applies valuable practices of eXtreme Programming and Scrum and provides a set of tools that allow high school software projects to benefit from modern software development methods. By emphasizing dynamic processes and a clear course of action an attractive perspective on CS is promoted.

Categories and Subject Descriptors
K.3.2 [**Computers and Education**]: Computer Science Education

General Terms
Human Factors, Theory.

Keywords
Secondary computing education, agile methods, project-based learning.

1. INTRODUCTION
In secondary computing education, software projects are promoted to provide an appropriate and student-oriented approach to Computer Science (CS) [19, 23, 32, 39]. Yet, most projects in this context are mainly focused on sequential project layouts that resemble traditional software development (SD) methodologies such as the waterfall model. In recent years it became apparent in professional SE that such methodologies often fail to produce high quality products, bring forward delays in delivery, and insufficiently consider customers' needs (e.g. [26]). Analogue issues can be found in school projects: unfinished projects, missing time and motivation for testing, neglected documentation, and teachers' difficulties in managing software projects are just some of the problems reported (cp. [19, 24, 32]). In modern SD, agile methods are promoted to provide a dynamic project management that relies on interaction and short design iterations. Agile methods build upon values and provide practices that are also highly expedient in high school contexts. Therefore, we present a new approach to projects in secondary computing education, which implements the theory of didactic transposition to adapt agile methods for project organization, management and implementation in classroom. Valuable agile practices of eXtreme Programming (XP) [2] and Scrum [39] will provide a set of tools allowing software projects in high schools to reference modern SD by highlighting dynamic processes that help to focus on good results, a clear course of action, and an attractive perspective on professional CS by addressing common problems at the same time. In section 2, research on projects in computing education will be discussed and problems with prevalent models will be analyzed. The findings suggest a missing consideration of the learners' perspective in project models. In section 3, agile methods in professional and educational settings are discussed for their potential of supporting a learners' perspective. By describing the agile model for school projects in computing education common agile practices are characterized and adapted. Finally, the model is discussed in the context of existing models, its potential, and issues in computing education.

Conference'10, Month 1–2, 2010, City, State, Country.

2. PROJECTS IN EDUCATION

2.1 Project Based Learning
Project-based learning (PBL) is an approach to teaching and learning in the classroom aiming for engaging students in explorative and problem-solving activities in authentic contexts. The concept is described to originate from teaching in Zurich and Paris in the 19th century. Since then, the idea has been picked up by various teachers and researchers and enriched with psychological, pedagogical and sociopolitical aspects, e.g. by Dewey und Kilpatrick [11]. Projects are understood as learning processes that draw on interests and demands of the students by striving for a complex result, often a product. This includes planning, problem-solving, analysis of different solutions and the evaluation of the process and its product. PBL is known for increasing students' motivation, for strengthening self confidence, and for fostering satisfaction in process and outcome (cp. [5]). Therefore, it is pro-

moted to foster high-level thinking skills including problem-solving and analysis skills. Thus, it helps to gain a deep understanding of topics and processes (cp. [1, 25]). PBL is known to encourage peer interaction (cp. [25]).

In Germany, Frey [16] elaborated PBL by describing the process along the following steps: A project starts with a project idea, which should be based on the interests of the students. Subsequently the idea is specified in order to find agreement on what the class will be trying to achieve. After planning the necessary activities the project plan is carried out. Milestones and meta-communication serve as tools for supporting the process. This model is commonly accepted within Germany and was fundamental for an adapted model in German computing education (cp. 2.3)

Summarizing, projects are characterized by being problem-focused and interdisciplinary, by allowing students a choice of topic and foster personal responsibility. Projects are generally carried out over a longer time span and are often graded "differently", e.g. by focusing more on processes and applying less pressure.

2.2 PBL in Secondary Computing Education

Because of the above mentioned benefits, PBL is widely used in higher computing education (cp. [6, 13]) and secondary computing education, especially in the context of software projects [39, 41]. Consequently, PBL is recommended as an appropriate approach to computing education in the majority of German curricula. However, there is limited research focusing on methodologies for SD projects in secondary education. Meerbaum-Salant and Hazzan [33] constitute "as far as we know, no general methodology has been developed for software development projects in the high school". Consequently, they propose an approach which focuses on a mentoring model for teachers (cp. 2.4).

In Germany, a model for high school software projects was proposed by Frey [15] and elaborated by Schubert and Schwill [39]. The majority of published school SD projects can be attributed to this model. Therefore the model will be analyzed in the following.

2.3 A Professional Perspective

A majority of publications concerning the use of projects in secondary computing education[1] stems from the 1980s and 1990s, generally analyzing and adopting the professional approach to SD and adopting the waterfall model for the classroom. The idea was that a professional model for SD would provide an appropriate framework for school projects: it offers students a structured learning process and gives insights into professional SD processes at the same time. In comparison to other subjects using PBL only as a teaching method without a connection to the subject matter itself, projects are scientifically anchored in CS [39]. Consequently, there is a large body of practice reports on SD projects. However, by analyzing publications of the major German conferences in computing education of the last 10 years we could not find any publication concerning methodologies for school projects in general secondary education. However, research shows that teachers consider it as important that students get familiar with SD processes, e.g. by running "through the workflows of the waterfall model" [30].

[1] Research on this topic is rare. The statement reflects the situation in Germany. However, we are not aware of comparable publications from other countries.

Figure 1 illustrates the project steps of Schubert and Schwill's model [39] with the corresponding output of each phase. The proposed model describes student activities along the software life cycle. However, it does not provide methods or practices of how students can reach the expected outcome of each phase. In the problem analysis phase all important environmental conditions need to be gathered clearly and completely. Furthermore, this phase includes planning activities for time, team and equipment. The resulting requirements definition serves as a contract between teacher and students. In the subsequent system design process a model of the system is specified by dividing the "overall system" into modules. Until here, the activities shall be performed by the full team, which is allowed to split up into subgroups for minor tasks. Then, smaller groups handle a module each under their own responsibility. The remaining phases follow the software life cycle.

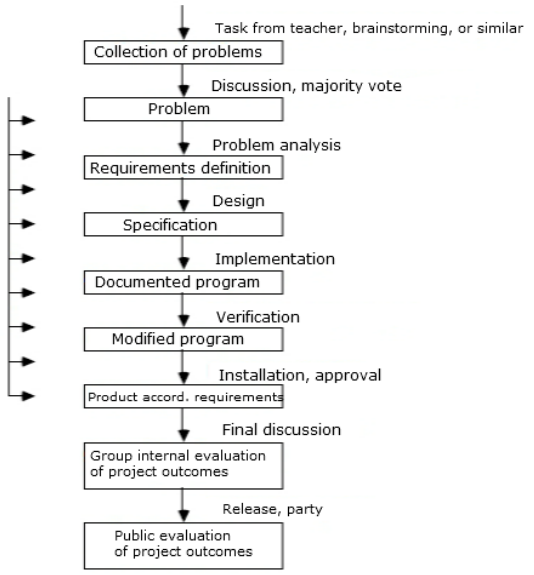

Fig. 1. Project model by Schubert and Schwill [39] .

Concerning the team structure, Schubert and Schwill recognize that the organization of the team cannot follow hierarchical structures as typically used in professional SD teams. Instead, they suggest equal status and responsibility among team members and a "force" for communication and common goals. This goal is addressed by assigning eight different roles for student positions within the project (computer responsible, project supervisor, interface responsible, tester, documentation responsible, butler, session chair, secretary).

Criticism of the methodology points out problems with a perceived bureaucratic overhead: "It is always the same: Students refuse to first plan on paper. Because only small programs are written, these are not documented. The taught principles of software development are hardly noticed" [19]. Often, testing is omitted in the project realization due to a lack of time and a lack of perceived importance, especially if the software does not have a practical use after the project (cp. [24]). Other problems may result from the structure of the project: The time span that needs to be scheduled is up to half a year (Schubert and Schwill [39] suggest to perform one project per semester). This is difficult to plan for, especially if students lack project experience and supportive practices. Additionally, a sequential project layout collides with "project-unfriendly" circumstances of formal lessons such as lim-

ited time, heterogeneous student abilities and lessons spread over several weeks. Humbert [23] even summarizes that the pedagogical dimensions of PBL are not sufficiently considered in such a project model. Additional problems of conducting school SD projects are outlined in the following by pursuing a teachers' perspective.

2.4 A Teachers' Perspective

Meerbaum-Salant and Hazzan [32] analyzed difficulties encountered by teachers in mentoring SD projects in Israeli high schools. Even though the curricular background and objectives are somewhat different[2] than for projects in general educational settings as described in this paper, the results are similar to the problems reported from German teachers. Additional problems were identified in the contexts of scheduling the project, CS expertise of the teachers, considering students' individual performances, and evaluation of the project. Teachers describe mentoring of SD projects as a more complex task compared to traditional teaching.

Meerbaum-Salant and Hazzan [33] express the need for a general methodology for SD projects in high school and address the previously identified problems in a mentoring methodology (ACMM). It is intended to support teachers, who are expected to be confident in a variety of knowledge types [32]. Therefore it describes a set of practices (Pedagogical Class Management Aspect, Social Aspect, Project Management Aspect) that shall be considered by teachers while mentoring a project. The ACMM takes into account the principles of agile software (such as communication, simplicity, feedback, respect), which basically are reflected in the teacher-student interaction.

2.5 A Learners' Perspective

For learning settings, where a team of students is working cooperatively on projects, we see potential for taking the idea of applying agile methods further than it is described in the ACMM: project management can be done by the student team. This can be supported by straightforward and easy to use methods adopted from modern SD. Additionally, students may benefit from experiencing a SD process which also includes management aspects in addition to activities like analysis, designing, coding, and testing, as described in the other models. In the following we demonstrate how *problems* identified by Meerbaum-Salant and Hazzan [32] may be addressed in such a project by considering agile methods as presented in section 3.

Schedule: Teachers may need to catch up with teaching of material during the project. The sequential project approach does not allow for such a teacher's intervention without disturbing the process. However, in an iterative project design, issues and success can be discussed in class regularly.

Required CS knowledge: Some teachers admit a lack of project development knowledge. Students will need help with CS knowledge while solving problems. It meets the ideas of PBL if student teams would be empowered to manage projects themselves. This can be supported by easy to follow practices and strategies which make teacher involvement almost unnecessary. Additionally, clearly defined practices may support teachers' confidence. Heterogeneous student teams stimulate mutual assistance before requiring the help of a teacher.

Students' individual work: Teachers see a need for personal supervision in order to achieve a timely completion of the project and meeting of the requirements. Agile methods allow for transferring this responsibility to the project team, hence relieving the teacher.

Evaluation of project outcomes: Agile practices naturally lead to a variety of documents which can be considered for project evaluation (e.g. estimates in planning poker or burn-down charts). Furthermore mutual assessment within teams may be performed.

All suggested practices require a change of the project perspective from the teacher to the learner. In section 3 the mentioned agile practices are discussed in more detail and transferred to classroom settings by the use of didactic transposition.

2.6 Didactic Transposition for Project Methodologies

Didactic transposition describes the process of adapting professional knowledge of a domain for teaching scenarios based on a didactic intent [9]. Hazzan et. al. [21] applied didactic transposition on agile SD methods with the intent to create a teaching framework and a mentoring methodology for software projects.

The model for school SD projects discussed in 2.1 can be attributed to didactic transposition as well. Here, the professional process was adapted under consideration of the underlying principles of PBL. Schubert and Schwill [39] emphasize the advantage of a method which is learning activity and learning content at the same time. However, we see potential for shifting the focus from professional process *knowledge* to modern professional *methods* which may be adapted in a way that they address previously outlined problems in school SD projects.

In the following, we discuss agile methods in professional SD and in education. We applied didactical transposition for developing an agile approach to projects in computing education which emphasizes a learners' perspective.

3. AGILE PROJECTS IN COMPUTING EDUCATION

3.1 Agile Methods in Professional and Educational Settings

Agile methods are popular amongst researchers and practitioners for enabling software developers to create systems that are more likely to be accurate in meeting customers requests, finishing in time, building robust systems, and creating usable/readable code (cp. [22]). Therefore, in industry agile methods are currently replacing waterfall or other linear methodologies that are known for shortcomings in the above mentioned goals of SD. Agile methods are focused on social interactions and dynamic creative processes. Hence, developers in agile teams often report on a strong satisfaction in their work experience and strong confidence in their outcomes (cp. [27, 31]).

The agile manifesto of 2001 [3] clearly presents values contrasting traditional linear methodologies and underlying understandings:

[2] German curricula emphasize computer science concepts in the context of general education which only partly includes algorithmic thinking and programming. In comparison, the underlying curriculum of the study emphasizes foundations of algorithmic thinking and programming [17]. Additionally, we understand projects as teamwork where several students or the whole class are working on the same goal, Meerbaum-Salant and Hazzan [31] describe projects where students work individually and "each student has his or her own project subject".

1. Individuals and interactions over processes and tools
2. Working software over comprehensive documentation
3. Customer collaboration over contract negotiation
4. Responding to change over following a plan

Agile methods are implemented in various frameworks. XP [2] and Scrum [40] are the most prominent implementations applied in industry and academia. Those methodologies define practices to assure compliance with the agile values as described in the agile manifesto. Agile methods are mostly understood to support a development process comprehensively from start to the final stage. However, the individual practices can be classified according to their main targets. Some practices are designed to structure team processes and customer collaboration while other practices focus on the quality of code and outcome.

Recently, several authors report on using agile methods in CS education at university level. Braught et al. [7] promote the use of agile methods because it helps female students to engage in programming tasks through interaction with peers. Nagappan et al. [34] highlight social experiences, learning processes, and quality of code when using agile methods in CS1 courses. However, literature shows that there may be possible barriers hindering an implementation of agile methods in university scenarios. Rico and Sayani [37] present a study where they found that students had already established their own approaches and habits for SD that were opposing practices of agile methods. According to Rico and Sayani it was almost impossible to convince the students to adhere to the introduced practices of agile methods. In this connection, they recommend to introduce agile methods as early as possible. Consequently, a benefit is assumed in the use of agile methods in school contexts. Literature on agile methods at university levels is still discussing the possibility of conducting a fully-fledged project management according to an implementation as XP or Scrum. Some authors promote an almost complete implementation of XP (e.g. [28]), some recommend to use and adopt a subset of practices (e.g. [8]), while others exclusively use one practice (e.g. pair programming) to support computing education (e.g. [7]). Schneider and Johnson [38] reviewed agile methods in computing education and highlight the importance of applying suitable practices according to their goals instead of fulfilling complete implementation of agile methods. Accordingly, Hazzan and Dubinsky [20] present ten reasons to consider agile methods in computing education.

Yet, literature on agile methods in secondary computing education is rare. Weigend [42] introduced elements of XP (user stories, spikes, test driven development, refactoring, and big visible charts) to provide a project-based iterative infrastructure that supports writing of high quality code. However, a sound methodology for connecting the presented elements in projects was not provided.

The work in hand is based on encouraging classroom experiences presented by Göttel [18]. In various educational CS projects, agile methods, such as pair programming, standup meetings, informative workspaces, and user stories supported students in their project work and additionally helped students to discover social aspects of CS. This success provided our basis for developing a comprehensive agile model for school projects in computing education.

3.2 An Agile Model for Projects in Computing Education (AMoPCE)

As discussed above, PBL represents a common teaching and learning method in computing education. However, even if common models suggest a structure and requirements for school software projects, methods are described insufficiently for the individual phases. Agile methods and modern SD principles provide a set of clearly described strategies that seems well suited for an implementation in school contexts. As described in the agile manifesto, they emphasize communication, visualization, teamwork and common goals. In the following, we want to introduce a model for school software projects that builds on the character of agile methods in order to address the problems outlined in section 2. It follows the agile manifesto by focusing on

1. Students and their interactions
2. Rapid success and working software
3. Collaboration in order to strive for a common goal over fulfilling a contract
4. Responding to change and learning progress over following a plan

The individual strategies and tools are illustrated in fig. 7 and will be described below in an agile model for projects in computing education (AMoPCE).

In this description we focus on processes and methods that are central for the agile SD process. Additional pedagogical aspects such as triggering students' motivation or finding agreement in choosing a project topic are not covered.

The process contains various techniques adapted from professional SD practices. They provide clear lines of action that can be followed by the students (e.g. generating user stories, planning poker, defining tasks). However, before applying them in a project we suggest introducing and practicing each method. On the other hand, an explorative learning approach is possible: The methods describe the processes in such detail that appropriate material can be created which allows students to learn and perform the processes independently.

The methods will be first discussed from a professional perspective[3] and subsequently transferred in a way that they can be *applied in classroom* (italic text). Examples will illustrate the process.

3.2.1 Preparation

Creating software requires competencies in programming and using tools. It is the responsibility of the teacher to make sure that the students have acquired the competencies needed for the software project ahead or that they will be able to acquire them during the project (e.g. with provided teaching material). Another aspect considers establishing an appropriate infrastructure (see [33] for further elaboration).

3.2.2 Ideas in

The initial steps of a SD process usually are devoted to requirements analysis listing possible features, approaches, and needs of

[3] In order to maintain a consistent presentation, methods and practices described in this paper are based on [35], which may be referred to for elaboration and additional information on modern SD practices.

the target audience. This phase is based on interviews, observations, and brainstorming sessions with the customers.

Building upon ideas and interests of students is a central characteristic of PBL. However, experience shows that students may not easily come up with ideas that can be implemented in such a project, especially at the first time. Therefore, an initial presentation of possible projects and the use of creativity techniques (e.g. brainstorming) are suggested. Resulting ideas shall be written down on individual Post-Its or cards for each activity that the software needs to provide (cp. fig. 2).

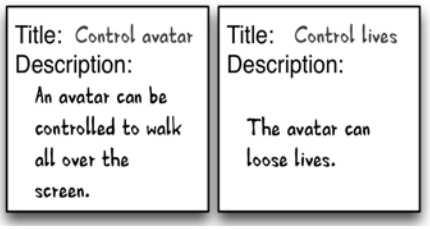

Fig 2. Post-its for recording ideas.

3.2.3 User Stories

User stories briefly describe features of a product that should be available to the actual user. Each user story addresses a specific activity of a user and is derived from the ideas of the requirements analysis. They are written from the perspective of a customer. User stories should easily fit on index cards and also be understood by non-developers. They should provide additional space for an estimation of the work effort.

In combination, user stories specify the entire intended product. A final state and amount of user stories has to be accomplished in agreement with the customer. Thereafter, stories are prioritized together with the customer by sorting user stories according to the importance of each story. Priorities are presented using incrementing numbers by powers of ten from 10 (most important) to 50 (least important).

Furthermore, the customer is asked to pick those stories that should be available in the initially delivered outcome or rather first major release. Consequently, discussion and reprioritizing stories may be necessary considering the basic features wanted for the first release.

User stories are created using various brainstorming techniques and take account of domain specific needs, knowledge, and approaches of the actual users specified in the requirements analysis.

A user story

- covers one activity that needs to be addressed
- represents the perspective of the customer
- is short, i.e. contains no more than three sentences
- does not use technical terms
- does not specify technology or tools

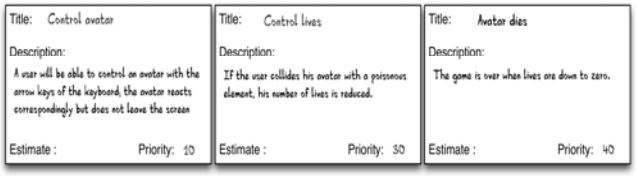

Fig. 3. Cards for user stories holding a title, a description, an estimate for workload, and a priority. Estimates will be added after completing the planning poker.

In professional SD projects one of the most important (and often unsuccessful) tasks is to find out what the customer wants. User stories provide a helpful way for achieving this goal. Since the students are going to implement their own ideas in their software projects, this goal does not apply. However, the team needs to find an agreement on the requirements for the software. These will be represented from a user's perspective: User stories briefly describe how a user interacts with the software. They can be developed from the previously recorded ideas. These need to be analyzed to find out, which interaction is really going to happen. This will be achieved by role playing and observation. Role playing is suggested as an attractive method for computing education to understand processes (cp. [4, 12, 14]). However, some of these examples use role plays in awkward contexts. In contrast, by role playing user interaction with the desired software system, students use a method which is anchored in modern SD processes and helps to identify relevant processes. The rules of the role play are simple: One student pretends to be the software and reacts accordingly. A sheet of paper may be used to illustrate the display. Another student takes the role of the user and instructs the software about what he or she wants to do, according to the previously obtained ideas. The remaining students observe the situation carefully to understand details and constraints of the desired product. The role play should be repeated several times with changing actors until no more new requirements arise. With this experience it should be easy to formulate the requirements from a user's perspective and write down the corresponding user stories (cp. fig. 3). Finally, the user stories receive a value for their priority. Priorities can be determined as a team, since generally agreement is quickly found.

Communication is an essential element of agile SD. Even if a set of user stories will now describe the final goal, questions and changes will appear in the following process. Since there is no customer who can answer questions and make decisions in order to clarify yet open questions, a group member needs to take over this special role: the product owner. This position may be passed around with each iteration.

3.2.4 Planning Poker

Planning poker is a hands-on method helping participants to estimate time needed for the work packages and guarantees a fair and comprehensible approach amongst all team members. Each participant holds a deck of cards to estimate the workload of a user story. There should be cards representing estimates in comprehensible units (e.g. developer-days) and special cards allowing players to indicate a lack of information, a need for a break, and already finished functionalities as shown in fig. 4.

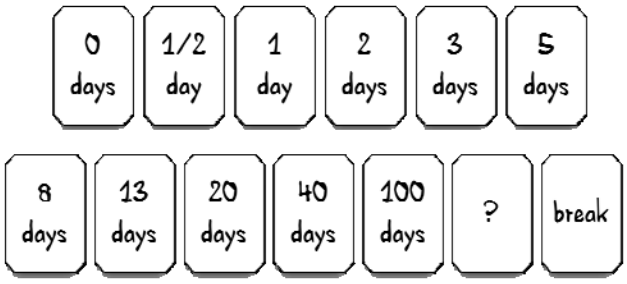

Fig. 4. Deck of cards used for the planning poker.

Each play round is devoted to one user story. A user story is placed in the middle of the table by the dealer and all participants place a card specifying their estimate face down on the table. All played cards are turned at the same time. The dealer collects the played cards and sums up the estimates trying to set up an average estimate. The dealer should address outliers by asking for reasons explaining fundamental differences in the estimates. Furthermore, the dealer should reflect on average estimates referencing the differences in the played cards. After each round the acquired estimate is written on the card of the user story. Additionally, the individual estimates are written on the back of the user story card to keep track of the decision process.

For a student team it is one of the most difficult tasks to estimate the workload and time demands of a given project due to a lack of project experience. Additionally, very likely not all planning relevant aspects are known at this point, learning processes will happen and changes may be necessary. Planning poker describes a playful way for challenging all students of the team to engage in the planning process by analyzing user stories and tasks, relating, estimating, explaining and defending their calculations, thus practicing their communication skills and ability to give and receive criticism.

For school software projects the same card values can be used as in professional SD. However, since these projects comprise a shorter working time, instead of days, 15 minute-periods seem to be appropriate. Each student estimates the time he or she believes he or she would need to implement the user story in focus. User stories should be presented with decreasing priority. Discussion of very divergent estimates will help resolving unspecified requirements and assumptions. After the planning poker is finished, the total workload for all user stories is divided by the number of programmers or programming teams (if pair programming is used) and compared with the time available. Unlike in professional SD, the priority is not to fulfill all requirements of the software but it is indispensable to keep the time available. If there is a major difference, e.g. more than 20 per cent, amount or complexity of the user stories need to be modified.

Again, for planning poker one team member needs to be the dealer. We suggest for this special role the position of a "teammaster", who leads the planning poker as well as team meetings. In the subsequent process this position also may be passed around with each iteration.

3.2.5 Tasks

After the initial planning poker, user stories are broken down into tasks. Usually each user story can be seen as a collection of tasks. A task is a rough description of a work package that should be done by a single developer. A task should have a unique self-explanatory name and should indicate its priority. The workload of each task should also be estimated by planning poker. Afterwards, task estimates are summed up to double check them with the initial estimates on the corresponding user stories. Thus, these estimates should not differ tremendously.

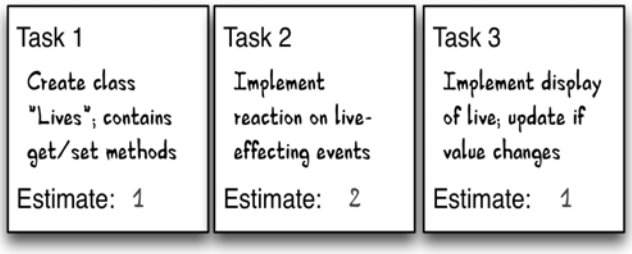

Fig. 5. Tasks for User Story "Control Lives".

While the user stories describe project goals from the perspective of the user, the students now need to change their perspective and look at them from a developer's viewpoint. By dividing user stories into tasks, various design decisions need to be made. Experienced programmers will now benefit from their competencies and known best practices, less confident students at this point can benefit and learn from team members, processes and team discussion.

3.2.6 Iterative Development, Prototypes, and Milestones

Agile processes are designed to provide short iterations that constantly come up with working prototypes that can be used and discussed with users or customers. This allows for rapid feedback loops that help to uncover misunderstandings, to detect issues in using the interface, and to adapt to new requests. An iteration is supposed to be short and has to be balanced according to implementing new features, fixing bugs, responding to change, and considering group dynamics or individual demands. In professional contexts, iterations vary between one week (5 working days) to one month (approximately 20 working days).

Iterations are planned in a team meeting by considering user stories' priorities, estimates, and the intended duration of an iteration. After deciding on the user stories to work on for the next iteration, they are pinned on a project board including associated tasks.

In school software projects, the planning of long development processes is reported to be difficult. Also, teachers report issues maintaining student motivation while they are not getting a grasp of the product until the whole project is assembled. The learning theory of constructionism emphasizes that learning happens especially felicitously in a context where learners are engaged with creating and investigating a personal relevant product (cp. [35]). Iterative development allows for creating a series of prototypes that can be analyzed, examined and played with in a constructionist sense. Also, such a design of the development processes allows for a higher flexibility in team organization and diversification due to more frequently changing tasks. Hence, each iteration is a mini-project containing each phase of the SD processes (requirements, design, code, test), but is easier to handle. This gives students the opportunity to perform the whole process several times within one project, to learn from and reflect on previous experiences and to take over several tasks in the team (in comparison to other project models, where tasks are more strictly divided amongst students). Besides the iterations, which should not be

longer than one to two weeks (which equals 2-4 lessons), milestones are used to structure the process and point out major achievements within the project. We suggest identifying 2-3 milestones for each project, representing versions of the final product with increasing value. However, only the achievement for the next milestone in the development is determined at a time. Milestones can be used for presenting the project progress for the rest of the class or teachers. Also, milestones should be positioned at times when the project pauses and teacher input is planned. Goals for the next milestone and the project progress are visualized at the project board.

3.2.7 Project Board

Project boards visualize goals and status of a current iteration and support target-oriented discussions. They present user stories and tasks in different status areas. Project boards are updated and discussed throughout the entire process. Thereby, it helps team members to keep track of the progress of the design process: the different areas of the board are used to present goals and accomplishments to the whole team. There are three main status areas: to-do user stories with associated tasks for the current iteration, tasks that are in progress, and completed tasks. Furthermore, there is an area to store user stories that need to be reconsidered in a future iteration. To provide a clear view, another area is reserved for finished user stories, allowing to take off corresponding task cards. Figure 6 presents an accordant project board.

Additionally, a burn-down chart is available on the project board. The chart visualizes the working time left in an iteration and work that needs to be done according to the task estimates. The chart is constantly keeping track of the progress by plotting the remaining sum of tasks at the end of a working unit.

Likewise, in a school software project all user stories with corresponding tasks are collected and presented at the team's project board. The project board is the central organizational and informative workspace for the entire project and should be available at all times, e.g. by placement at the classroom wall. It is also the meeting point for the regular standup meetings.

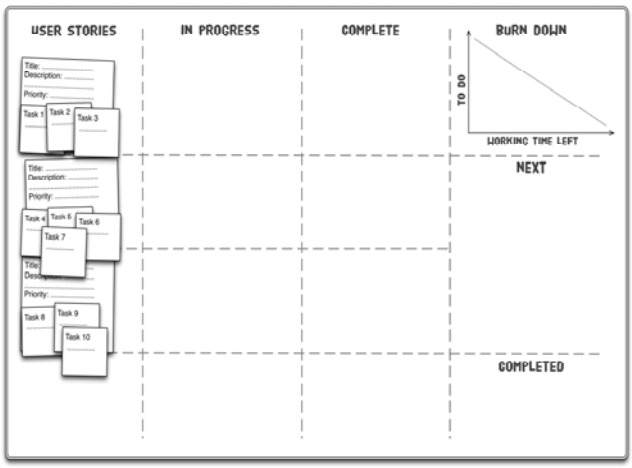

Fig. 6. Project Board including a burn-down chart.

3.2.8 Standup Meetings

Standup meetings provide a recurring fast and short update of the efforts of the team: Each team member has to report on accomplished tasks, possible issues in accomplishing certain goals, and a plan for the work day. Meetings are done while standing to guarantee a fast and goal oriented session kicking off a workday and should not exceed 5 to 15 minutes.

In school projects, standup meetings can provide an elegant way for starting off a lesson or working day within a project by encouraging team communication, sustaining motivation and identifying problems. The team gathers around the project board and recalls the project status, success and problems of the last working session and the goals for the day. After each team member has given a short statement the burn down chart is updated. If non-minor problems are identified, a longer meeting may be scheduled.

3.2.9 Pair Programming

Pair programming ensures an elaborated coding style: A pair of programmers uses a single programming environment for coding. The person using the keyboard and mouse is adopting the role of the "driver". The driver is actually coding and asked to present his or her ideas to the second programmer (the "navigator") verbally. Meanwhile, the navigator questions the coding outcomes, discusses possible misinterpretations, and seeks for alternative solutions that are more straightforward by keeping in focus the overall goals. The roles of driver and navigator are changed repeatedly during a workday. Programming in pairs helps to detect possible slips in the design and architecture of the code at early stages. Furthermore, it helps programmers to build upon social interaction uncovering misinterpretations of relations and intentions of code parts.

In many schools, two students share one computer due to limited hardware availability and hence often program in pairs. The agile method of pair programming supports this practice and adds a framework that encourages attention from both students, mutual learning and a notion of programming as a social activity.

Fig. 7 illustrates the organization of a school software project based on AMoPCE as described above. Other agile methods and ideas may be included in such a school project as well, e.g. test driven development, refactoring or "keep it simple".

3.3 Focusing on Programming Style and Outcome

Several agile practices may be applied in the process to bring forward a high quality outcome. However, since these practices are optional for project organization and partly depend on programming environments, in the following they are only summarized but not adapted for school software projects.

3.3.1 Test Driven Development

Test driven development replaces documentation and provides criteria to evaluate the code solutions: Programmers define intended functionalities by writing automatic tests covering all states and the correctness of the accordant results. First reports of using this method in secondary education have been published (e.g. [10]).

3.3.2 Refactoring

Refactoring introduces the idea that every part of the code should be reconsidered and changed if a more accurate solution can be found: Developers should rewrite parts of code without adding functionality when there is a more straightforward solution available that passes all automatic tests. Emphasizing this aspect may raise students' awareness of efficiency and for evaluating different solutions.

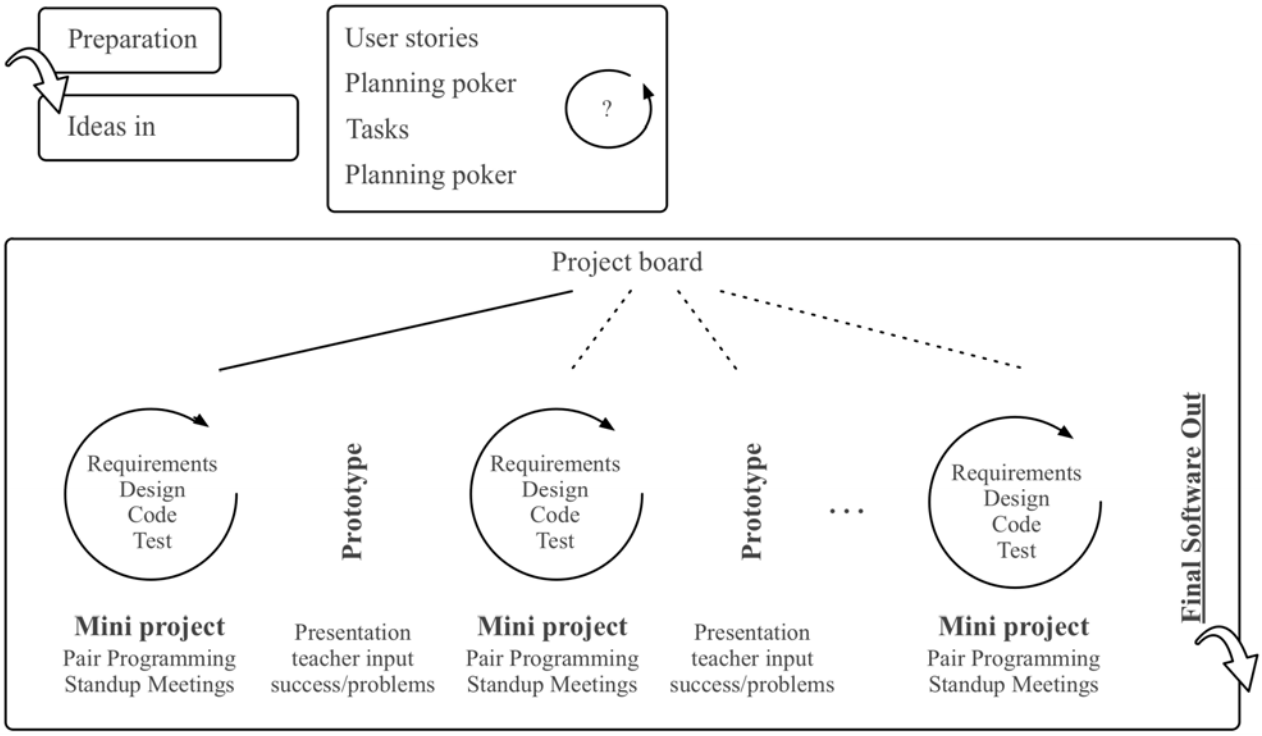

Fig. 7. Agile Model for Projects in Computing Education (AMoPCE).

3.3.3 Keep It Simple

This claim refers to code minimalism: each function should be solved by minimal and straightforward code snippets to ensure an elegant and readable code that is easy to maintain.

4. DISCUSSION

The proposed agile model for SD projects AMoPCE addresses a majority of the previously identified problems. Agile practices fill the learners' gap between requirements and outcome by providing clearly defined strategies for handling difficult planning tasks. Based on the perception of PBL as a team activity, which is in line with modern SD, a team size of 4-6 students is recommended. Dividing the class into several teams, in comparison to having the full class working on one project as proposed e.g. by Schubert and Schwill [39], allows for addressing a broader range of topics, hence it is easier to meet the interest of more students. Also, it should be much easier to find agreement within smaller teams.

Adopting an iterative project design matches the formal circumstances of school projects. In most cases, projects will be worked at along the regular school timetable, sometimes in a so-called project week on several days. In both settings, pedagogical aspects of working in group settings, such as giving curricular input, intermediate project presentations or discussing of common problem solving strategies, can be taken into account easier, since meetings in plenum are part of the model. Correspondingly, projects are divided into mini-projects, which are easier to overview, plan and understand; bureaucratic overhead is reduced. Classroom-management aspects are addressed with professional practices such as standup meetings. This includes recalling of the project status at the beginning of each lesson and quickly planning the individual and group activities for the day.

Within projects, ideas, students' motivation, and skills change over time. Due to the limited time available to work on the projects per week, this is especially very likely for school software projects. Agile methods welcome changes and provide mechanisms to adapt to them with often changing tasks and a straightforward implementation of PBL. Students' confidence is addressed by increasing familiarity stemming from the iterative character of the process.

In comparison to the ACMM, which provides solutions for teachers' difficulties that are grounded in teaching situations, the proposed agile model provides solutions for students' difficulties in learning situations. This in return is expected to relieve the teacher. Giving students clearly defined practices to manage their development processes allows teachers to focus on supporting elaboration and implementation of students' ideas, thus changing the teacher's role from instructor to coach. This better meets the demands of PBL. Additionally, it allows teachers to highlight creativity and social aspects that are rarely seen in connection with computer science (cp. [8, 29]). Applying an adapted SD methodology in school that is also implemented by well known large scale companies may help teachers and students to build an adequate and appealing understanding of computer science relying on creativity, dynamic change, feedback, and soft skills. These attributes may support an attractive notion of computer science, as found in our first experimental settings with agile methods in school projects [18] .

In summary, AMoPCE is suited for supporting the objectives of PBL, for maintaining a professional orientation and for easing the mentoring of software projects. However, in this model curricular aspects such as content and size of a project are not explicitly considered. Nevertheless, it provides the flexibility to fit into a variety of possible scenarios. Evaluation and assessment aspects,

e.g. assessing individual achievement in comparison to group achievement are not determined by the model. However, again practices of SD appear to be adaptable and can be considered: Frequently, agile development teams use self assessments and subsequent peer reviews to verify individual workload and commitment.

As outlined in section 3, there are further practices of agile methods that focus on the quality of the outcome: test driven development, collective code ownership, refactoring, and keep it simple. We acknowledge these practices to be also useful in educational settings but we understand them to be highly dependant on features and methodologies of the used programming environments and tools (e.g. code repositories, automatic testing environments).

The use of agile practices in school SD projects has the potential for replacing the so far predominantly used sequential model. From discussions with teachers we know of the high interest, but a lack of knowledge and resources concerning the use of agile methods. This includes the demand for a revised project model. With AMoPCE, as outlined in this article, we present a model which explains ideas and realization possibilities of agile practices and which can be used as blueprint. In a next step of our research the model will be applied in classroom settings and it will be investigated, to which extend the expected benefits and problem solutions will be approved in practice.

5. REFERENCES

[1] Barak, M., Waks, S. and Doppelt, Y. (2000). Majoring in technology studies at high school and fostering learning. Learning Environments Research 3(2): 135-158.

[2] Beck, K. and Andres, C. (2004). Extreme Programming Explained: Embrace Change (2nd Edition).

[3] Beck, K., Beedle, M., et al. (2001). Manifesto for agile software development. The Agile Alliance: 2002-04.

[4] Bergin, J. (2000). The Object Game. An Exercise for Studying Objects. Online at: http://csis.pace.edu/~bergin/patterns/objectgame.html (visited 01.07.2012).

[5] Blumenfeld, P. C., Soloway, E., et al. (1991). Motivating Project-Based Learning: Sustaining the Doing, Supporting the Learning. Educational Psychologist 26: 369-398.

[6] Blumenfeld, P. C., Soloway, E., et al. (1991). Motivating project-based learning: Sustaining the doing, supporting the learning. Educational Psychologist 26(3-4): 369-398.

[7] Braught, G., Wahls, T. and Eby, L. M. (2011). The Case for Pair Programming in the Computer Science Classroom. Transactions on Computing Education 11: 2:1--2:21.

[8] Caspersen, M. E. and Kölling, M. (2009). STREAM: A First Programming Process. Transactions on Computing Education 9: 4:1--4:29.

[9] Chevallard, Y. (1988). On didactic transposition theory: Some introductory notes. In Proceedings of the International Symposium on Research and Development in Mathematics, Bratislava, Czechoslavakia.

[10] Christopher, G. J. (2004). Test-driven development goes to school. J. Comput. Small Coll. 20(1): 220-231.

[11] Dewey, J. and Kilpatrick, W. H. (1935). Der Projekt-Plan: Grundlegung und Praxis, Hermann Böhlaus Nachfolger.

[12] Diethelm, I. (2007). Strictly models and objects first - Unterrichtskonzept und -methodik für objektorientierte Modellierung im Informatikunterricht. Kassel, Universität Kassel.

[13] Fincher, S. and Petre, M. (1998). Project-based learning practices in computer science education. In Proceedings of the IEEE Frontiers in Education Conference (Tempe, Arizona).

[14] Fothe, M. (2007). Algorithmen in spielerischer Form. In Proceedings of the Praxisband der GI-Fachtagung Informatik und Schule.

[15] Frey, K. (1983). Die sieben Komponenten der Projektmethode - mit Beispielen aus dem Schulfach Informatik. LOG IN 3(2): 16-20.

[16] Frey, K. and Schäfer, U. (1982). Die Projektmethode. Weinheim [a.o.] Beltz.

[17] Gal-Ezer, J., Beeri, C., et al. (1995). A high school program in computer science. Computer 28(10): 73-80.

[18] Göttel, T. (2012). The image of Computer Science and social aspects of modern software development processes in school contexts. Hamburg, University of Hamburg.

[19] Hartmann, W., Näf, M. and Reichert, R. (2006). Informatikunterricht planen und durchführen, Springer.

[20] Hazzan, O. and Dubinsky, Y. (2007). Why Software Engineering Programs Should Teach Agile Software Development. SIGSOFT Software Engineering Notes 32: 1-3.

[21] Hazzan, O., Dubinsky, Y. and Meerbaum-Salant, O. (2010). Didactic transposition in computer science education. ACM Inroads 1(4): 33-37.

[22] Highsmith, J. and Cockburn, A. (2001). Agile Software Development: The Business of Innovation. Computer 34: 120-122.

[23] Humbert, L. (2005). Didaktik der Informatik, Teubner.

[24] Koerber, B. (1992). Die Angst des Lehrers vorm Projektunterricht. LOG IN 12(5/6): 3.

[25] Krajcik, J. S., Czerniak, C. and Berger, C. (1999). Teaching children science: A project-based approach, McGraw-Hill Boston, MA.

[26] Larman, C. and Basili, V. R. (2003). Iterative and Incremental Development: A Brief History. Computer 36: 47-56.

[27] Layman, L., Williams, L. and Cunningham, L. (2004). Exploring Extreme Programming in Context: An Industrial Case Study. Proceedings of the Agile Development Conference. Washington, DC, USA, IEEE Computer Society**:** 32-41.

[28] Loftus, C. and Ratcliffe, M. (2005). Extreme Programming Promotes Extreme Learning? Proceedings of the 10th annual SIGCSE conference on Innovation and technology in computer science education. New York, NY, USA, ACM**:** 311-315.

[29] Maass, S. and Wiesner, H. (2006). Programmieren, Mathe und ein bisschen Hardware... Wen lockt dies Bild der Informatik? Informatik-Spektrum 29(2): 125-132.

[30] Magenheim, J., Nelles, W., et al. Competencies for informatics systems and modeling: Results of qualitative content analysis of expert interviews. In Proceedings of. IEEE.

[31] Mann, C. and Maurer, F. (2005). A Case Study on the Impact of Scrum on Overtime and Customer Satisfaction. Proceedings of the Agile Development Conference. Washington, DC, USA, IEEE Computer Society**:** 70-79.

[32] Meerbaum-Salant, O. and Hazzan, O. (2009). Challenges in Mentoring Software Development Projects in the High School: Analysis According to Shulman. Journal of Computers in Mathematics and Science Teaching 28(1): 21.

[33] Meerbaum-Salant, O. and Hazzan, O. (2010). An Agile Constructionist Mentoring Methodology for Software Projects in the High School. Transactions on Computing Education 9: 21:1--21:29.

[34] Nagappan, N., Williams, L., et al. (2003). Improving the CS1 Experience with Pair Programming. ACM SIGCSE Bulletin 35: 359-362.
[35] Papert, S. and Harel, I. (1991). Situating Constructionism. Constructionism. S. Papert and I. Harel. Norwood, N.J., Ablex Publishing.
[36] Pilone, D. and Miles, R. (2008). Head first software development, O'Reilly Media, Inc.
[37] Rico, D. F. and Sayani, H. H. (2009). Use of Agile Methods in Software Engineering Education. Proceedings of the 2009 Agile Conference. Washington, DC, USA, IEEE Computer Society**:** 174-179.
[38] Schneider, J.-G. and Johnston, L. (2005). eXtreme Programming: Helpful or Harmful in Educating Undergraduates? Journal of Systems and Software 74: 121-132.
[39] Schubert, S. and Schwill, A. (2011). Didaktik der Informatik, Springer.
[40] Schwaber, K. and Beedle, M. (2001). Agile Software Development with Scrum.
[41] Tucker, A., Deek, F., et al. (2003). A model curriculum for K-12 computer science: Final report of the ACM K-12 task force curriculum committee. New York, NY: The Association for Computing Machinery.
[42] Weigend, M. (2005). Extreme Programming im Klassenraum. In Proceedings of the INFOS 2005 - 11. GI-Fachtagung Informatik und Schule (Dresden, Germany, 2005). GI-Edition - Lecture Notes in Informatics (LNI).