Preprint, please use following information for referencing:
Jari Porras, Kari Heikkinen and Jouni Ikonen, Advanced Technology for Learning (ATL) International Journal, Vol. 4, No. 1, 2007, ACTA press, pp. 43-52, ISSN 1710-2251.

CODE CAMP – A SETTING FOR COLLABORATIVE LEARNING OF PROGRAMMING

J. Porras, K. Heikkinen, J. Ikonen

Lappeenranta University of Technology, Laboratory of Communications Engineering,
P.O.Box 20, FIN-53851
Email: Jari.Porras@lut.fi, Kari.Heikkinen@lut.fi, Jouni.Ikonen@lut.fi

Abstract

Collaborative learning is studied, especially in light of the code camp approach, in this paper. Code camp means a short, intensive collaborative learning event in which participants work in groups. Two code camp settings and one traditional programming course are presented. Students of the courses completed questionnaires and the answers were analyzed to establish the value of the code camp approach. It seems that students like the code camp approach and the results seem to be better than with traditional approaches to learning of programming.

Key Words: Code camp, Collaborative learning, Programming

1. INTRODUCTION

Programming may be the most essential single skill that a computer science student has to master. Thus, special care must be taken when designing and implementing programming skills courses. However, the learning of programming is a very complex process and it takes considerable time. In recent years, *collaborative learning*, has been gaining momentum both as a teaching method and as a core element of active learning. Collaborative learning, for the purposes of this paper, is a learning approach in which the students solve carefully defined tasks together so that critical learning outcomes are met. Collaborative learning is often described, e.g. in [1], as a significant shift from traditional classroom-centric or teacher-centric teaching. Here, it is extended by adding a time dimension (24h or week-long). *Active learning* is a learning approach in which the students solve problems, answer various questions, make their own set of questions, discuss and debate different aspects of their tasks and assignments, and do brainstorming during the teaching sessions. Thus, our code camp can be discussed through active learning definition only roughly. However, through collaborative learning definition our code camp approach can be presented and evaluated.

Preprint, please use following information for referencing:
Jari Porras, Kari Heikkinen and Jouni Ikonen, Advanced Technology for Learning (ATL) International Journal, Vol. 4, No. 1, 2007, ACTA press, pp. 43-52, ISSN 1710-2251.

In [2], *Preston* presents a literature survey of collaborative learning. Preston lists five critical common attributes to successful collaborative learning; i) *Common task or learning activity suitability*, ii) *Small group learning*, iii) *Cooperative learning*, iv) *Interdependence*, v) *Individual accountability and responsibility*. *Johnson and Johnson* [3] have identified the following criteria for a common task to be suitable for collaborative learning; i) *the task is complex or conceptual*, ii) *problem solving is desired*, iii) *divergent thinking and creativity is desired* , iv) *mastery and retention are important*, v) *quality of performance is expected* and vi) *higher-level reasoning strategies and critical thinking are needed*. In our code camp approach the group has a *common task*. The task on a higher level (e.g. to develop a context-aware game), is similar for each group. All of the complexity of the task is not pre-designed, but the complexity and nature of the code is evaluated through demonstration and explanation. Based on Preston's survey [2], collaborative learning research has pointed out the benefits of *small group learning*. Various approaches and group formations have been found to be successful. When considering the role of the faculty in promoting cooperative behaviour, the survey points out gain in learning occurs if the faculty takes an active role in guiding the students. This guidance involves motivation by teachers, providing justification for the tasks and approach, listening to students, showing students how to teach and coach etc. Listening to the students also gives the instructors feedback on how the students have understood the basic aspects of collaborative learning. Students need to be observed in order to evaluate their work and give advice on cooperative model and behaviour. *Interdependence* is a further course development issue and is mostly defined as an approach in which an individual student cannot be successful, if the teamwork is not a success. Thus the students are obliged to cooperate. Based on Preston's survey [2], positive interdependence is reached when the roles and responsibilities within the group are fixed and when the learning activity is clearly structured by the teachers. *Individual accountability*, based on Preston's survey [2], occurs when the team members take individual tests and receive individual grades. For teachers, this is challenging, as the tests and evaluation needs to be defined so that all students are encouraged to acquire the same skill set, possibly through changing roles within the course, or if grading of the course is heavily weighted based on the success of all students of the group. In his study *Preston* uses pair programming as a collaborative learning technique and his analysis forms a basis for recommendations to ensure the success criteria (a-g) are met, namely: a) *Higher quality programs*, b) *Decreased time to complete programs*, c) *Greater understanding of the programming process*, d) *Increased enjoyment of programming*, e) *Decreased dependence on teaching staff*, f) *Improved course completion rate* and g) *Improved performance on exams*. The code camp described in this paper met some of the same success criteria, however, without novel knowledge of collaborative learning and thus our approach is not directly comparable with Preston. *McConnell* discusses about the risks collaborative and active learning possesses. He presents [4] nine dimensions that have both low and high risk. These dimensions are; a) *class time*, b) *amount of structure*, c) *amount of planning*, d) *subject e) controversy potential*, f) *student's knowledge on the subject*, g) *student's knowledge of pedagogy*, h) *teacher's experience of pedagogy and i) interaction*. In our code camp approach some of these dimensions are pre-set. The outline of the students' studies is in very much detail designed, but the implementation, apart from the lectures and hands-on exercises, is not. *Burgess and Hanshaw* [5] introduce an approach in which different learning styles are integrated into teaching delivery within computing classes. However, in our code camp approach this was not the major design issue. *C.A.Sherman* conducted a study [6] into effective techniques to get useful and relevant feedback from the students. In our approach, feedback was direct and constant; teachers continuously monitored the progress of the groups, asking and answering questions, and making suggestions.

Collaborative learning has for some time been popular in Lappeenranta University of Technology as many course assignments have been in the form of group work. The purpose of the group work has been often to decrease the number of assignments which have to be inspected, but also to let students enrich their learning by discussing the topic with fellow students. In the code camp approach this has been taken a step further: students spend intensive time together tackling a problem. Programming camps have been held which range from 24 continuous hours to 5 day camps. This approach fulfils the four first attributes of Preston's definition of successful collaborative learning. Individual accountability is not so clearly addressed, however it should be noted that individual grades are given.

2. CODE CAMP - RECALLING THE JOY OF CHILDHOOD LEARNING

The code camp approach tries to improve the learning process of programming by concentrating on two aspects, efficient time usage and cooperation. The term camp refers to a situation where participants get together and live a while together. The term code refers to coding, i.e. writing programs. During a code camp, participants write programs together, solve problems related to their work together, eat together and even might relax together in a sauna. The intensive time together gives the opportunity to work on ideas without interruptions from other (school) tasks and promotes the possibility to interact with other people working on the same situation. In essence, the code camp is a short term, intensive and cooperative approach for collaborative learning-by-doing of programming skills. By emphasizing the social aspects, learning can be done in more meaningful way. As the camps are usually very intensive, many participants refer to them as experiences.

Typically, on code camps:

- Students solve a challenge as a small group and they have to learn necessary skills to complete the task.

- Completion of the task is a group effort. The group may have students from different levels, but each member is assumed to take some responsibility for the work.
- Students are encouraged to interact with other groups. Evaluation of the course recognizes "the code camp spirit" – helping other groups is preferred.
- Creative thinking is desired as groups have to innovate to complete their task. The students are usually limited by a set of technologies which must be used.
- Tasks are usually complex, because the groups must learn and use many technologies.
- Groups are required to show working realisation at the end of the code camp.

The items presented in the above list have a lot of similarities with Preston's list of critical common attributes to successful collaborative learning, and Johnson and Johnson's enumeration of suitable common tasks for collaborative learning. Johnson and Johnson's list is a bit troublesome for our approach as students are given a lot of freedom to innovate and the time which they have to complete the task is very limited.

Figure 1. (Left-hand side) A snapshot from situation where a group presents their accomplishment on a projector.

Figure 2. (Right-hand side) Intensive work session mixed with coding, conversation and brainstorming.

We have implemented code camps in two flavours; quick and dirty, and a more polished version with a longer running time. Quick and dirty is our initial approach, where students work intensively for 24 hours and must turn the assignment in at the end of the code camp. We are still running annual code camps which follow this approach. The first code camp (C++ coding in the Symbian environment) was arranged 2003 during the 12th Summer School on Telecommunications in Lappeenranta (http://www.it.lut.fi/ssotc) as a practical part of the summer school. In the 13th Summer School the code camp worked with the Nokia MUPE platform (http://www.mupe.net) as the programming environment and emphasized location aware mobile games. The topic at the 14th Summer School was the development of software for mobile devices. Each of these three code camps was successful and feedback from participating students was good. Figure 1 presents a photo shot of a typical code camp situation, where an instructor or participant is showing a coding example with a projector. In this case a group is showing demonstration of their position based game run on a cellular phone emulator. Participation in the code camp was very rewarding for students as they were able to see some concrete results in a very short time. Figure 2 is a casual and common picture from a code camp. A team has been intensively working on a problem and show to another team how to solve it.

These 24 hour code camps have been very useful and taught many things to our students which they could not study otherwise, particularly subjects which might not be available for regular curricula. However, the assignment results were quite unfinished and time to teach a topic was very limited. These 24 hour code camps have been very challenging for instructors and students as topics were chosen so that students had very little experience with the selected topics. This is due to the fact that the summer school should offer something extraordinary for the participating students. The code camps started with a short 1-2 hour introduction to the subject by experts from the selected field. After that the participants moved to the *hands on* phase, where they have given example program codes which the students compiled and run to get to know the environment. Normally the participating students have started from the example codes and made their own realisations by extending and applying them to solving new tasks. As participants are new to the subject, teaching staff and experts from industry are available all the time to answer their questions and help if they have problems. Following to the successful implementation of location aware applications in the 2nd 24 hour code camp, a separate, more polished week-long code camp course on programming was launched. A longer code camp requires students to arrange their personal lives so that they can actively participate in the camp. They also have to plan their work so that they get sufficient rest and nourishment. These things make participation much more complicated for the students. The first week-long code camp (context aware game design and implementation) was especially challenging as the course was held at the same time as other courses and only evenings and nights were reserved for the camp. As duration of the camp was extended, also the number of lectures was increased and students were offered a possibility to pass with a theoretical, practical or combined course option. The course had speakers from the arranging university and multiple companies. Each day consisted of lectures, exercises and time to work on the practical project. The success of the first week long code camp encouraged us to arrange another long code camp. This time *Design of modern www-based information systems with J2EE* was selected as the topic as we had active ongoing co-operation with IBM. The dates of the code camp had to be selected carefully not to interfere with other activities, and we had to use a four day time slot for the camp instead of the desired five days. The course had 2-4 hours of lectures or demonstrations daily and the rest of the time was used for working in small groups. Groups were encouraged to help each other as that is part of the camp spirit. They could also consult local staff and IBM-trainers who were there. In this code camp the groups were given daily objectives and their progress was monitored. The next day's lectures were built on top of the previous day's achievements. Successful completion of the code camp required students to complete the given assignment and write a report about their work.

3. ANALYSIS OF THE CODE CAMP AS OPPOSED TO A TRADITIONAL PROGRAMMING COURSE

So far we have arranged three 24 hour code camps and two week-long code camps. As we have collected extensive data concerning these week-long code camps, we analyse them here and compare the results to similar data from traditional programming courses. First we describe the characteristics of each course under study and then analyse the results.

- Fundamentals of Programming (6 credits)

  This course is a basic programming course targeted to first year university students. The objective of this course is to familiarise the students with the logical thinking needed for programming. Although the C language is used for the exercises, the main emphasis is on fundamental programming skills like algorithms and their implementations, program designs, programming styles etc. Students do not need any prior experience in programming. The course is realised by having traditional lectures, exercises, programming projects as well as a self-learning environment, Viope (http://www.viope.com/). The course is 14 weeks long. Teaching is arranged in a weekly fashion with 2 hours of lectures and 2 hours of exercises every week. In the latter half of the course the lectures are more of demonstration type, thus preparing the students for the practical work required. There are two different courses on the same topic, one for IT students (Course A) and one for students of other departments (Course B). Course B makes great use of the self-learning environment.

- Special Course on Application Programming (5-8 credits)

  The Special Course on Application Programming was the first week long code camp arranged at Lappeenranta University of Technology. MUPE with emphasis on context awareness was select as the basis for the course. This course was based on previous research projects on positioning, context information as well as the programming environment. There was a clear theory component in this code camp so it was possible to pass the course by taking an exam or alternatively by finishing the practical part (or both, in which case students received extra credits). In this paper we concentrate only on the practical part of the course, i.e. the code camp.

- J2EE Programming (3 credits)

  The J2EE Programming course was a special code camp course for J2EE programming with modern tools. This course was arranged together with IBM and the course made use of Rational Software Architecture on top of the Eclipse environment. The course was targeted to advanced, i.e. 4th year, students as an optional course. The main objective of the course was to deepen knowledge of web application programming through the use of the modern programming environment Eclipse. As a code camp course this, course was arranged as an intensive four day course outside the normal lecture period. Students were supposed to spend most of the time during those days on this course. Students formed groups of three and each group had to complete a given task. In this course, the task was to produce a new student database system that can handle several different operations. These operations were defined by the students and implemented using the selected environment, i.e. Eclipse and IBM RSA.

Both of the code camp courses based their work on a well defined programming environment. The role of the environment has been noticed by other researchers/teachers as well. *Depradine and Gay* studied [7] active participation of integrated development environments. Their main results indicate that the software environment plays a pivotal role in understanding the link between the programming and the underlying technology. *Hanks* evaluated [8] Eclipse and its use as an integrated development environment. Hanks found that using Eclipse had a positive overall influence. The students considered that their learning was improved, and it made the classes more interesting. However, students had some difficulties in understanding Eclipse's user interface. *Chen and Marx* also introduced their experiences [9] using Eclipse. They found that using an IDE like Eclipse is not necessarily ideal for an introductory courses but preferable for more enhanced programming skills such as applying programming skills in problem solving of a specific subject. Our experiences verify these results presented previously.

3.1. Student Surveys

Each course was evaluated with a comprehensive questionnaire. Lappeenranta University of Technology uses the Webropol (http://www.webropol.com/) tool to collect feedback. The scope of each course questionnaire is described below:

- Fundamentals of Programming

Questionnaires are completed in the beginning and at the end of the course. The first questionnaire concentrates on the background of the students. Questions concerning previous experience reveal the initial level of the students. At the end of the course two separate questionnaires are completed, one for students that dropped the course and the other for students aiming to pass the course. The first questionnaire tries to discover the reasons why students drop the course while the second questionnaire concentrates on the learning outcomes of the course. As there are two separate groups on the Fundamentals of Programming courses (Course A and Course B) it is possible to compare the learning outcomes of students with different backgrounds.

- Special Course on Application Programming

  In this course the course questionnaire was completed only at the end of the course. Students were asked some 50 questions concerning the background of the student, code camp realisation, contents of the course for both the theory and practical part, as well as the learning outcomes of the course. As this was the first code camp course arranged at Lappeenranta University of Technology emphasis was given to finding out the suitability of the code camp method for teaching programming.

- J2EE Programming

  In this course the course questionnaire was realised at the end of the course. Some 30 questions were asked. The questionnaire included both code camp and content specific questions. The objectives of the questions were to verify the results from the questionnaire of the previous code camp and to improve the code camp process.

3.2 Results

Answers to the questionnaires were analysed and the main results are summarized at Table 1. Table 1 has separate columns for each course. The columns present the key values for each course. The first two rows define the course parameters. The next two lines show the number of students participating in the course and answering the survey. Three main results from the survey are given as the next values. These results are then compared to the parameters of final grading. All courses show improvements in programming skills. With traditional courses the student opinion seems to correlate quite closely with the final grade given to the student. With code camps the difference can be explained by different perspectives. The students evaluate their programming skills on some specific area whereas the course grade consists of other elements as well.

Table 1 Summary of the course questionnaires.

	Course A	Course B	MUPE	J2EE
Type	Traditional	Traditional	Code camp	Code camp
Length	14 weeks 2+2 hours/week	14 weeks 2+2 hours / week	5 days 12-24 hours/day	4 days 12-24 hours/day
Students	170	80	98	33
Answering %	87 %	70 %	60 %	39 %
Study time	-	-	4,8	5,1
Initial self evaluation	2,25	1,875	2,2	1,5
Final self evaluation	3,7	2,4	2,8	2,6
Course grading criteria	Exam Practical work Viope	Exam Viope	Practical work: Idea Implementation Documentation Presentation CC spirit	Practical work: Idea Implementation Documentation Presentation CC spirit
Average course grade	3,5	2,6	4	4,1
Course passing %	41 %	65 %	88 %	100 %

The results of the surveys are further analysed below.

- Fundamentals of Programming

  This course had two options with minor differences in objectives. The broader option (course A) had some 170 enrolled students whereas the narrower option (Course B) had 80 students. Course A was mainly targeted to computer science students and Course B students of other departments. The questionnaire at the beginning of the course was answered by 204 students (148 + 56). These questionnaires revealed that some 50% of students had some experience of programming either at university or earlier in high school. Students on Course A had a willingness to learn programming (66 %) and an understanding that they might need programming in the future (65 %) whereas students on Course B were participating in the course because it was a mandatory course (40%) and because they wanted to know what programming is about (50%). Although motivations for taking the course seem to be quite different with these two groups, in their own view their skills are quite similar. Course A students felt that their initial level was 2,25 (of 5) whereas on Course B students felt that their skills were on level 1,875. At the end of the courses, the questionnaires concentrated on the skills gained on the course. There was a clear difference between the two groups. The Course A students felt that they know and may use the basics of programming almost fluently (3,7 out of 5) whereas the other group had doubts about of their skills (2,4 out of 5). The results can be compared to the exam results; 70 students out of 170 on Course A completed the requirements of the course and were eligible to take the exam (drop-out rate 58%) and the average exam grade for students who passed the course was 3,5. Course B had an average grade of 2,6 but the drop-out rate only 35%. One of the main reasons for difference in the drop-out rate is the fact that Course B has no practical assignment but mainly used the Viope self-learning environment. The main reasons mentioned for dropping the course were that it was too difficult course (Course A) or a lack of time (Course B).

- Special Course on Application Programming

The course had a total of 98 students enrolled. Students had the possibility to select their option, practical or theoretical, while on the course. As we are mostly interested in experiences of practical programming, this study examines only those students that completed the practical part. 16 groups of three persons were formed meaning that 48 students wanted to learn how to program context-aware mobile games. The questionnaire at the end of the course was answered by 59 students. 27 of these students took the practical part. This means that approximately 60% of groups answered the questionnaire. Only two groups out of the total 16 groups did not finish their work. As this course was meant for senior or graduate students, with a strong background in programming. Even though the students had an average of 4.8 years of studies, they mostly had only average programming skills (2,2), in their own opinion. One reason for this result is that the students felt that this question referred more to the topic of the course than programming itself. The questionnaire covered both code camp style of teaching as well as the type of programming (context-aware mobile games). In this analysis we mainly concentrate on code camp aspects. 76% of students considered the code camp an appropriate way of teaching programming. Students felt that the intensive project type of work with social aspects overcomes the problems caused by this kind of total concentration on one topic. The length of the course was suitable by 50% of students and at the same time 40% of the students felt that the code camp should be longer. This is quite surprising as at the same time the students liked the intensive work. Most of the students felt that one week is a good length of time for a code camp but the theoretical work took days so the effective programming time was reduced to three days. Table 2 presents the improvements in programming skills based on the questionnaire. In general, the number of students having good programming skills increased from 45% to 55%. It is hard to verify this change as there is no separate programming exam but students are assessed by continuous evaluations. As 14 groups out of 16 finished their work, there was a passing rate of 88 %. On average, the groups passed with grade 4. Students were pleased with this kind of course as 90 % of the students hoped for more code camp courses and 82 % said they would participate in the next course.

Table 2 Improvements in programming skills.

Programming skills		1	2	3	4	5	Average
General	Before	2	5	15	9	7	3,368
	After	2	2	14	11	8	3,568
Games	Before	14	7	11	5	1	2,263
	After	4	14	9	7	3	2,757
J2ME	Before	20	5	10	3	0	1,895
	After	6	14	9	8	0	2,514
Mobile Games	Before	20	13	4	0	0	1,568
	After	5	18	9	5	0	2,378
Context-aware	Before	18	11	5	3	1	1,895
	After	4	16	12	4	1	2,514
XML	Before	8	8	15	3	4	2,658
	After	3	6	19	4	4	3

- J2EE Programming

J2EE programming course was arranged as a code camp style of course. A lot had been learnt from the previous course and thus this time there was supposed to be no distribution of resources (e.g. no exercises concurrently in multiple rooms). In the worst case this would mean limiting the number of students allowed on the course. In the beginning of the course there were 33 students enrolled in the code camp. Although the course was advertised actively, the actual time frame of the course (first week of the year) was not the best possible. Altogether 31 students started this code camp once, again in groups of two or three persons. This code camp attracted even advanced students as the average study time of the students was 5,1 years (average graduation time is 6,7 years). Once again the participants felt that their background was in the topic under study only moderate even though their average programming course grades were 3,4. Table 3 presents the improvements in programming skills based on the questionnaire.

Table 3. Improvements in programming skills.

Programming skills		1	2	3	4	5	Average
General	Before	0	1	4	3	4	3,833
	After	0	1	4	3	4	3,833
J2EE	Before	8	2	2	0	0	1,5
	After	0	6	4	2	0	2,667
WWW	Before	0	4	4	3	1	3,083
	After	0	2	5	4	1	3,333
Software design	Before	2	4	4	2	0	2,5
	After	1	2	6	3	0	2,917
Use cases	Before	1	3	5	3	0	2,833
	After	1	1	4	6	0	3,25

Improvements in programming skills seem remarkable as 60% of students said that this time the code camp was too short. The results of this code camp were excellent. All groups managed to pass the course thus giving a passing rate of 100 %. Students were evaluated continuously throughout the code camp and ideas, codes, documentations, code camp spirit and presentation were evaluated. 75 % of participants liked the way the code camp approach gives a supportive environment for studying programming. 91 % of the participants felt that the code camp supported the development of programming skills well or extremely well. 92 % would increase the number of code camps and 83 % said they would participate in the next code camp (the rest of the students said maybe).

3.3 Student experiences from the programming courses

The following comments reflecting experiences on the code camp have been collected from the course surveys. The idea here is to emphasise the value of code camp style learning from the student point of view.

- Fundamentals of Programming

Preprint, please use following information for referencing:
Jari Porras, Kari Heikkinen and Jouni Ikonen, Advanced Technology for Learning (ATL) International Journal, Vol. 4, No. 1, 2007, ACTA press, pp. 43-52, ISSN 1710-2251.

"It is hard to be the one who does not understand immediately and need to ask questions", "Exercises are hard to keep as the level of students is so different"

These two comments give a nice perspective to the problems of teaching a fundamental course on programming. Even though Course A is meant only for students that will really need programming in the future, the starting level is quite heterogeneous. With Course B a majority of comments were of this type.

"Viope environment helps the understanding of programming and its concepts"

Students of Course B did not have a practical assignment and thus their skills were tested only with the Viope programming environment. This environment gathered praise from the students. The same environment was in use in Course A but the objectives of the use of this environment were different.

- Special Course on Application Programming

  "In code camp courses it is easy to discuss and learn that way", "People do not need to be afraid of using some other student's code which helps understanding"

  These two student opinions reveal that the objectives set for the code camp course were achieved. These students learnt the value of doing things together. This so-called code camp spirit started to show during the last night but it remained for the following code camps where the supervisors know how to promote collaborative behaviour.

  "Environment had several bugs but luckily there were good supervisors around", "There were too few supervisors around"

  A major mistake with the first code camp was insufficient allocation of resources. For the practical work, students needed to be divided into two different rooms. This caused some troubles with supervising resources. As this was the first code camp, students did not realise the meaning of collaborative work until the last night. Collaborative work, i.e. code camp spirit, really grew during the long hours of the last night.

- J2EE Programming

  "Intensive and collaborative environment for programming", "It was easy to ask help and to give help in this environment"

  This time the location of the course was organized such that all groups were together. Enough supervising resources were allocated and this time also the code camp spirit was around from the start. This is surprising as only 25% of the students had participated in the previous code camp.

  "The competition decreased the code camp spirit"

One of the original ideas of the code camp was to have a competition between the groups. Although it could not be seen during the course, the idea of prizes reduced the code camp spirit somewhat. One way of getting around this problem was to emphasise the meaning of code camp spirit in the evaluation.

Table 4. Code camp evaluation.

Claim	Arguments
Code camp reduces the time to complete programs	As code camp is an intensive approach it really accelerates the development of programs. It could be thought that difficulties of programming could not be solved any faster, but code camp as a collaborative effort has shown that problems can be solved much faster and actually the number of total stoppages decreases.
Code camp offers greater understanding of the programming process	Several days of intensive working helps the understanding of the overall process. When several groups tackle the same problem eventually some groups find the solution and teach the other groups. This has been verified with the two long code camps presented here.
Code camp increases the enjoyment of programming	The nature of code camp is enjoyment. Programming can be fun even though one is not a master. The social aspects of code camp make it easier for those with little experience to come into a group and learn.
Code camp decreases dependence on teaching staff	Code camp teaches students to work together. While in traditional practical exercises the use of each other's code is prohibited, the goal of a code camp is to maximize the use of each other's code. It is possible, and very much recommended, to combine pieces of data to get more functionality.
Code camp improves course completion rate	Traditional courses have a completion rate somewhere around 50%. In the last code camp we doubled this rate. The last code camp was a success in this sense.
Code camp improves grades of students	The improvement in grades can be seen when comparing the two approaches: traditional teaching and code camp. In traditional teaching the grade averages were close to 3,5 (or 2,6) out of 5 and in code camps closer to 4. This is a remarkable improvement.
Code camp allows continuous evaluation of students	Both the completion rate and grade improvements are due to the continuous evaluation of students. This evaluation has two effects. First the supervisor collects more information for grading and at the same time the student becomes more careful in his work.
Code camp inspires students to try their own ideas	The relaxed atmosphere inspires students to come up with interesting ideas. Collaboration with other groups develops ideas and realization possibilities even further. Sometimes the competition requires some original ideas.
Code camp challenges for personal improvement	Code camp creates an atmosphere in which groups playfully compete with each other. This leads to a situation in which students focus on improving their work for their own satisfaction.

3.4 Teacher experiences from the code camps

As teachers of these code camps we need to summarise the findings from the perspective of the teacher. This is done by mirroring claims (by Preston) and their arguments to our own findings (see Table 4). For those who plan to realise a code camp of some sort we give some advice. Realisation of a number of code camps easily gives some insight into what to do and what not to do. A lot of the items stated here are part of standard event arrangements. However, the difference between arranging of a code camp and a lecture course is quite big. Useful tips are presented in Table 5.

Table 5. Advice for code camp arrangements.

Plan carefully. One week full of action is more complex than planning the lectures for one week.
Reserve time. Careful preparation takes a lot of time.
Select charismatic / stimulating leader for the camp. A person who can get students excited about the task is an excellent selection. The formal qualifications of the leading person does not really matter if he understands what kind of things students are interested in and he/she is respected by the students.
Divide responsibilities beforehand among the staff. They are busy (or may have other plans) during the camp.
Select a **schedule** that does not interfere with other teaching or popular events. Remember that students need time for this course. Teachers should **participate** themselves as much as possible. Otherwise students might think that only they should give all their time for the course.
Set a clear **goal** for the code camp. Do not set the threshold too high. Students must feel that they have learned new skills (NOT that they have failed before a huge task). Define the learning outcomes of the course.
Make use of the groups. Select tasks so that a group can work in them and they can complete the task using different approaches. You can mix groups with people who have different skills. If the topic is suitable, you might mix, e.g., engineering and business major students to get multidisciplinary innovations.
Leave space for creativity. Code camp is an excellent opportunity for being innovative.
Make clear instructions. What is expected from the students? How to get started? Where to get help if stuck and especially *when* help is available?
Make **backup** plans for everything: place, hardware, speakers, food, etc.

4. CONCLUSIONS AND FUTURE ENHANCEMENTS TO THE CODE CAMP APPROACH

This paper presented a code camp setting for the collaborative learning of programming. The paper used criteria from the literature to evaluate the code camp's appropriateness and suitability as a teaching method for the collaborative learning of programming. The code camp approach met most of the criteria for successful collaborative learning presented. The code camp setting was studied by introducing several different code camp realisations and comparing these to the traditional way of teaching programming. The study also listed typical elements within any code camp realisation. In addition, an extensive list of recommendations and advice were given for setting up a code camp. These recommendations and advice show that even though a code camp requires a lot of background work and time dedicated solely to course design and the realisation, the actual experiences of both students and teachers are highly rewarding. Code camps truly were held in high regard based on the student surveys. Participants of code camps generally enjoyed the study method. One reason may be the difference to traditional lectures, but of course the inspiring atmosphere, brain teasing challenges and informal tasks give some flavour to the results. According to the surveys, it can be said that code camps are liked by almost all students participating in them. Thus we feel that a code camp can be used as a powerful tool for learning. An important factor in a learning process is to get students excited about things which are done. Learning should not be a dull task. Children learn new languages in many kindergartens just by play. These immersion class techniques can be applied to other subjects also. A stimulating teacher (or teaching) can quite easily encourage students to start working with interesting and challenging projects. In such cases students are not thinking about learning, but solving a problem. They have to do different things to reach their goal and "automatically" learn the required things on the way to completing the task.

Preprint, please use following information for referencing:
Jari Porras, Kari Heikkinen and Jouni Ikonen, Advanced Technology for Learning (ATL) International Journal, Vol. 4, No. 1, 2007, ACTA press, pp. 43-52, ISSN 1710-2251.

The four year history of code camps has taught us a lot. We have learned many lessons what to do and what not to do. We can happily recommend that others try the code camp approach. Currently, it appears that code camps will be arranged as part of future summer schools as well as part of the regular curriculum. However, the code camp approach must be improved by further standardising our practices. At present code camps are extra courses that are not part of the university curriculum, in future code camp courses should *have a more defined status* in teaching, as the results are excellent. Another important aspect is adding the theory of learning itself to the design. As some claim learning is a long process, we could, e.g., run a longer code camp within which we could have mini code camps concentrating on smaller issues. Thus we could integrate the learning pedagogy to our vision of learning-by doing. Further research needs to be carried out by conducting parallel courses on the same topic (one course using traditional method and the other the code camp method) with similar problem setting and similar desired learning outcomes.

References

[1] Y. Wu and J. Wang, Collaborative learning program of IEEE EMBS student branch chapter at BUPT, *Proceedings of the IEEE 31st Annual Northeast Bioengineering conference*, 2005, 123 – 124.

[2] D. Preston, Using collaborative learning research to enhance pair programming pedagogy, *ACM SIGITE Newsletter*, *3*(1) 2006, 16-21.

[3] R. Johnson and D. Johnson, An overview of cooperative learning, In J. S Thousand, et. al. (Ed.), *Creativity and Collaborative Learning: A Practical Guide in Empowering Students and Teachers*, (Baltimore: Paul H. Brookes Publishing, 1994, 31-43).

[4] J.J. McConnell, Active and cooperative learning: more tips and tricks, *ACM SIGCSE Bulletin*, *37*(4), 2005, 34 – 38.

[5] G.A. Burgess and C. Hanshaw, Application of learning styles and approaches in computing sciences classes, *Journal of Computing Sciences in Colleges, 21*(3) 2006, 60-68.

[6] C.A. Sherman, Earning positive evaluations from IT students: effective techniques, *Proceedings of the 6th conference on Information technology education*, 2005, 255 – 259.

[7] C. Depradine and G. Gay, Active participation of integrated development environments in the teaching of object-oriented programming, *Computers & Education, 43*(3), 2004, 291-298.

[8] B. Hanks, Using Eclipse in the classroom, *Journal of Computing Sciences in Colleges, 21*(3), 2006, 118-127.

[9] Z. Chen and D. Marx, Experiences with Eclipse IDE in programming courses, *Journal of Computing Sciences in Colleges, 21*(2), 2005, 104-112.