Nerding for Newbies 2014
A Summer School in Computers & Programming

May-Lill Bagge[a]
may-lill.bagge@student.uib.no

Anya Helene Bagge[a]
anya.bagge@ii.uib.no

Baste Nesse Buanes[a]
baste.buanes@student.uib.no

Anna Maria Eilertsen[a]
anna.eilertsen@student.uib.no

Alf Kristoffer Herland[a]
alf.herland@student.uib.no

Sofija Ivanova[b]
sofija.a.ivanova@gmail.com

[a]Dept. of Informatics, University of Bergen

[b]Knowit AS

Abstract

Recruiting students to informatics can be a challenge, particularly when it comes to recruiting female students. The Department of Informatics at the University of Bergen has had severe problems with both recruitment and retention of female students.

As part of an effort to raise interest in computers and programming in general, and among women in particular, we organised a two-week summer school where students could learn a range of subject including programming, Linux, electronics, statistics and other basic computer geek skills – in short, *Nerding for Newbies*. In this paper, we discuss the organisation of the summer school, our teaching methods and our experiences.

1 Introduction

In the first two weeks of the 2014 school summer vacation, we organised *"Nerding for Newbies"* a summer school in computers and programming at the Department of Informatics at the University of Bergen. The students at the summer school were applicants to bachelor programmes at the Department, as well as other applicants and students from other departments at the Faculty of Mathematics and Natural Sciences.

As this was our first attempt at making a summer school, *Nerding for Newbies* was held at a relatively small scale with regard to the number of students. The experience from this summers event has given some valuable information about any potential future events, and some of these are detailed in this paper.

Our main motivation for organising the summer school was a concern about how departments and schools that offer education in computer science often have problems

This paper was presented at the NIK-2014 conference; see http://www.nik.no/.

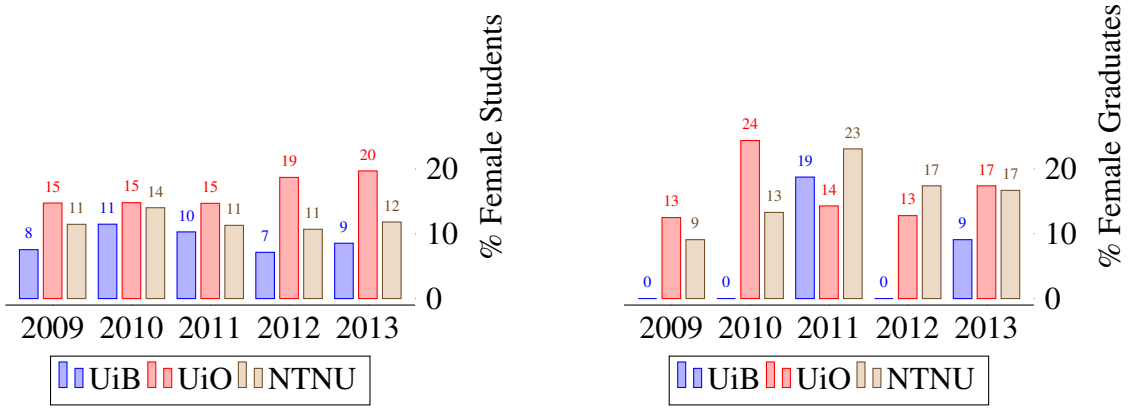

Figure 1: Percentage of female enrolled bachelor students in informatics at UiB, UiO and NTNU (left), and percentage of female informatics bachelor graduates at UiB, UiO and NTNU (right).

in recruiting and retaining female students. Previous experience from the instructors at *Nerding for Newbies*, suggests that courses, teaching methods and valued skills at the Department for Informatics tend to favour male students. The lack of graduating female students and lack female students that go on to master or PhD studies or become members of faculty[1] may be evidence of this.

Figure 1 shows female informatics students and graduates at the Universities in Bergen, Oslo and Trondheim[2]. Bergen has a very low percentage of female students (only 7.5% on average for the past five years); the situation in Oslo and Trondheim is a bit better (around 16.25%), but still low.

Some teaching methods favour male students, while other favour females [6, 4]. Using a mixture of methods that favour males and females, would allow female students to fit in and learn. We feel a summer school based on this principle would help female students to enjoy and complete their studies.

A second motivation was the reported feelings of inadequacy and anxiety that often faces female students of computer science, which hampers their performance [7, 2]. Giving new students a head start, and tasks that give them a feeling of accomplishment might help the female students feel more prepared for their studies.

Third, many of the potential students of computer science, and most other subjects, seems to be bewildered by the amount of choices offered in higher education and it has also become more common to take a year or two to improve grades from high school before starting higher education. A fun and educational summer school could help these potential students to make an informed choice about their future education, and motivate them to actually begin their studies. This would make *Nerding for Newbies* a success also if it clarifies for a potential student that computer science is not for him or her.

Finally, we thought it would be fun for both instructors and students to gather, learn and have some social events.

[1]The department has less than 5% women on the permanent teaching staff – significantly less than Oslo and Trondheim, at 16% and 13%, respectively.

[2]Includes “informatikk”, “datateknologi” and “datavitenskap” bachelors from UiB, “informatikk”, “informatikk 5-årig” and “informatikk: ...” bachelors from UiO, and “informatikk” bachelor from NTNU. Note that the numbers from Oslo include subjects that would be taught at the Infomedia department in Bergen, so the numbers are not entirely comparable.

	Monday	Tuesday	Wednesday	Thursday	Friday
10–12	Raspberry Pi	Raspberry Pi	Raspberry Pi	Raspberry Pi	Raspberry Pi
12–14	Python / Prezi[2]	Python / LaTeX[2]	Python / LaTeX[2]	Python	Python
14–16	Java / Statistics[1] / Electronics	Java / Statistics[1] / Electronics	Java / Statistics[1] / Building computers	Java / Statistics[1] / Electronics	Java / Statistics[1] / Electronics

Table 1: Weekly schedule. The school ran over two weeks, with the Python and Java courses repeating the second week, and the statistics course only running the first week. ([1] first week only, [2] second week only)

The main contribution of this paper is an experience report from our summer school, with a description of the individual courses (section 2), the background and experiences of the students (section 3), as well as a brief outline of the practical organisation of the school (section 4). We also provide a discussion (section 5) of our teaching philosophy and gender perspectives, and a short evaluation.

2 The Courses

The largest courses in terms of numbers of classes given and participants are detailed below; Java course, Python course, Raspberry Pi course, statistics and R course and electronics course. In addition we held some smaller courses in building a computer, LaTeX and the use of applications. We had planned, and prepared, other classes in the use of applications and computers, such as smartphones, security, home networks and the internet, but the courses that ran over several days were put on the schedule at the same time, and these shorter courses lost out. See Table 1 for the exact courses and classes we ended up giving.

Raspberry Pi Course

The *Raspberry Pi*[3] is a credit-card sized single-board computer designed for educational purposes. It's very affordable ($35 for the model we used) yet powerful enough to be used to learn about Linux, programming and simple electronics.

The Pi course ran two hours per day for ten days, and covered basic Linux use, shell programming, interfacing with the Raspberry Pi camera module, and how to use the general-purpose I/O facility (GPIO) to connect to LEDs and buttons.

Giving each student a small computer they could have complete control over had benefits, particularly when teaching them about Linux, since they were free to install new software, mess around with the system and see the distinction between regular users and the root users on their own – things that are not possible on the locked down University computers. They also didn't have to worry about breaking things; if something went wrong, they could just start over with a fresh Linux installation – or even a new Pi if they managed to fry the board.

Additionally, the Pis support the same software and languages used for the other courses, including Python, Java, R and LaTeX.

[3] `http://www.raspberrypi.org/`

The Linux part of the course covered installation and setup of the Raspberry Pi, and basic use of the Linux desktop. We then covered use of the command line, wildcards, the file system hierarchy, software installation, creating/moving/editing/deleting files, job control, I/O redirection and so on. We also did simple shell programming, with conditions, loops and reading/writing input.

The students also experimented with the Raspberry Pi camera module, which allowed them to take pictures and capture videos. We combined this with shell programming, so they could make scripts that did time lapse photography. The camera module was somewhat finicky in use – the connector on the Pi had a tendency to break and the camera itself would sometimes hang. Some of the students who were doing biology made plans for a mobile biology lab with an infrared version of the camera to capture data on photosynthesis activity.

The Pi's GPIO facility allows arbitrary components to be attached to pins on the board which are controlled from software. A simple experiment is to connect a LED to an output pin, and then turn it on and off from Python. The students experimented with blinking one or more LEDs, and then connecting a button and writing a script that would wait for a button press.

Thanks to a lucky find at the on-site recycling station, we also acquired some point-of-sale equipment which we programmed from the Pi: a barcode scanner which we connected to a script that tracked attendance (together with barcoded name badges), and a receipt printer that printed the attendee list.

The Raspberry Pi course used a combination of short whiteboard lectures and lab work, with the main emphasis on lab work. The students could try out everything right away during the lectures, and the lecturer and assistants would move around and make sure that everyone got things to work. We made sure that the assistants were accessible so the students got used to ask for help when needed.

Java Course

The Java course was intended to teach basic programming in Java to absolute beginners in five days, and was held twice on two consecutive weeks. Our goal was to make the students able to write and understand simple Java code, and become aware of abstraction processes. All the students that followed the course had brief experience with programming, even thought that was not a prerequisite. For exercises we used some from `codingbat.com`, and some custom exercises made by the instructors.

The course covered the basic parts of java: primitive types, operators, variables, simplified pointers (explained as "what will this expression evaluate to?"), if-sentences, scopes, for-loops, while-loops, methods, parameters/arguments, tables, objects, constructors, static/non-static, Javadoc, and some predefined classes like `String` and `Scanner`.

Each session was a mixture of lecture and exercises. The lectures were meant to introduce the students to new theory, while the exercises let them explore this theory and develop their understanding of it. Watching them work and encounter problems gave us a much better insight into their misconceptions and how parts of lectures could have been misunderstood, than the students conscious feedback would have given.

In the end everyone was able to read and write simple code, and to give meaningful input on a larger project (a dungeon crawl game) which we wrote together over one and a half session. Some of them had gone from absolute newbies to being able to not only solve basic coding problems, but also to research more theory online themselves.

During the course we used online help and Eclipse for documentation, and we stressed the importance of understanding programming and abstraction over remembering language specific things, which you can look up anyway. This seemed to be helpful both because it is an important skill to develop, and because it is demotivating for the students to discover that they forget these specific things (e.g. `Scanner`-methods, `String`-methods, and so on) before they get used to programming being about understanding instead of remembering.

Python Course

In order to introduce the students to programming in Python we gave them a review of the basic building blocks of Python; variables, lists, loops, `if`-conditionals, methods and a little about classes. They also learned about importing libraries and simple graphical programming.

The primary focus of the classes were problem solving. The instructor explained a type of problem or a concept with examples in a Read-Evaluate-Print-Loop (REPL), where you write code and can see it evaluated immediately. The students had the same REPL and copied the instructor. After each explanation from the instructor, the students were given one or two similar problems to solve with the help of the instructor and assistants. Finally a solution to each problem was presented by the instructor.

After the initial concepts were explained, classes were solely about problem solving. The students were taught to use a simple unit testing framework and were given tests their programs should pass. Their final problem was to make a small game and drawing this game.

The purpose with our choice of topics and problems was to show the students that programming isn't mysterious and difficult, and even a week can have you writing small programs. In order to make this happen we focused on making the students write as much code as possible and neglecting some theory and details that an introductory course at a University necessarily and usually teaches.

Statistics and R course

One of our goals with this course was to introduce students to R so that they would learn some basic concepts and would be able to continue by themselves. We decided to use simple data sets that students would easily understand, and that would be easy to present and analyse. For instance: at the beginning of the course students were asked about their height and gender. This data was gathered and later used in course exercises. Another goal was to make this course interactive, so that students could ask questions and interrupt at any time.

Having that in mind, classes were organised in the following way: The instructor started by telling them what they would learn by the end of the day. She then explained the theoretical background for the visualisation that would be performed (what is an average, histogram, boxplot and so on). Thereafter the instructor gave a demo presentation, where a piece of code was explained line by line. The students had to follow the instructors code on their own computers. After this demo they had to do an exercise that was based on the concepts just explained.

By the end of the course students had learned how to import datasets from files, write scripts that show simple plots, histograms, pie charts, and learned how to import and use other libraries. There was a lot of focus on how they could find new information and continue learning by themselves.

It was clear that the students' background influenced how easily they learned the programming part of the course. Some students were struggling to understand that one can write a piece code that can be run on different datasets at different times, while others were bored because everything seemed so easy for them.

We solved this by preparing exercises at different levels.

Electronics

We focused on soldering of components and construction of starter projects that consisted of ready made circuits and components.

The students learned basic soldering and desoldering of standard components like resistors, diodes, and transistors.

We taught some theoretical concepts, but the students favoured the practical aspect of electronics and we tried to keep the lectures as dynamic and flexible as possible.

Theoretical subjects we covered were amongst others analogue components and digital components like ICs and logic gates.

3 The Students

The students in numbers: 39 students in total, 49% women, and a median age of 24 years. Eight were applicants to the University, 19 were already students and the rest were in the category "other". The other category mainly included students that had not yet finished high school.

We attribute our capacity to attract such a high percentage of female students to several factors. (Although we can not say with any certainty that this is accurate or true.) Our marketing of the summer school was spread evenly to the genders (email invitations went to all potential students, and posters were hung for all to see). However, our matter-of-fact and concrete description of the courses[4] were intended to be appealing to female students. We also had female organisers of the school, which may have made it more attractive to potential female participants.

As female students tend not see themselves as well prepared for computer science courses [2, 7], this may have lead them to see an opportunity to prepare and lower their general anxiety about their studies this fall, to participate more readily. This would apply both to female students that were about to start a bachelor degree in Informatics and to other students that wanted to take a programming course as part of a different degree.

We asked the students to evaluate *Nerding for Newbies* and received replies from 58% of the students. Out of these were 43% women and 57% men.

95% answered that they would participate again and 76% would recommend *Nerding for Newbies* to others, while the remaining 24% would recommend the summer school, but not to everyone.

Many of the students found the courses at *Nerding for Newbies* to be quite educational, as can be seen in Figure 2 and Figure 3 (left).

Quite a few of our students had already started their higher education in other fields, such as biology and physics. When we realised how large a contingent were mainly interested in other fields and saw informatics as a tool or additional subject, we tried to add relevant tasks for other fields in our courses.

[4] `http://bit.ly/inf2014`

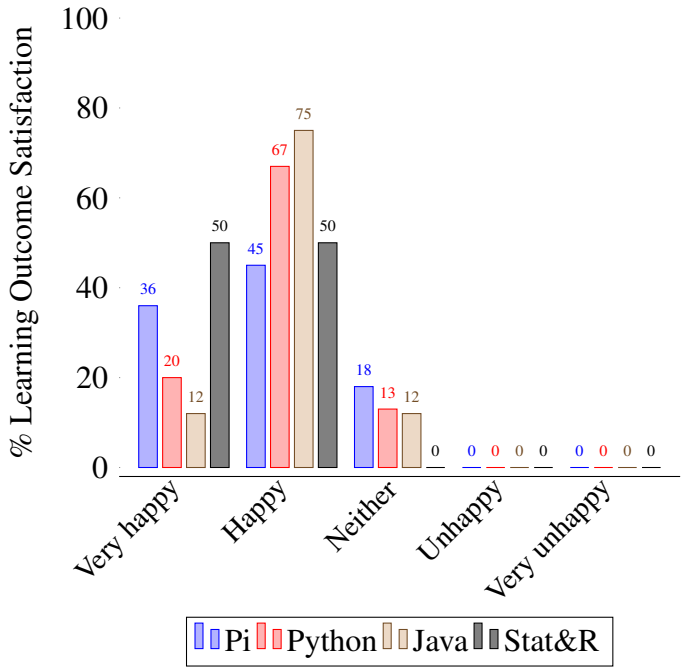

Figure 2: The students' satisfaction with how much they learned in the various courses.

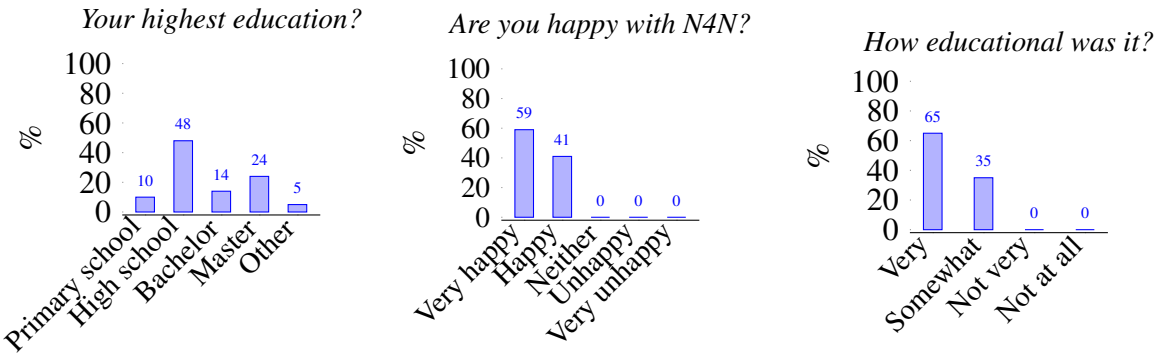

Figure 3: The students highest completed education (left), the students level of satisfaction with Nerding for Newbies (middle) and how educational the students found Nerding for Newbies (right).

Figure 3 shows that the students in general were quite happy with *Nerding for Newbies* and felt they learned quite a bit at the summer school. All in all, we are satisfied with the evaluation our students gave us.

4 Organisation

The first step in making *Nerding for Newbies* was recruiting the instructors. We chose three students that had previously shown promise as educators and asked them to participate in the project. Also, two former students and one faculty member were recruited as instructors.

Each instructor was then asked if they had a course they wanted to teach, and to prepare such a course in keeping with the goals for the summer school; choosing teaching methods that did not only favour male participants, giving the participants an idea of what informatics is, allowing the students to learn a little that prepares them for their first

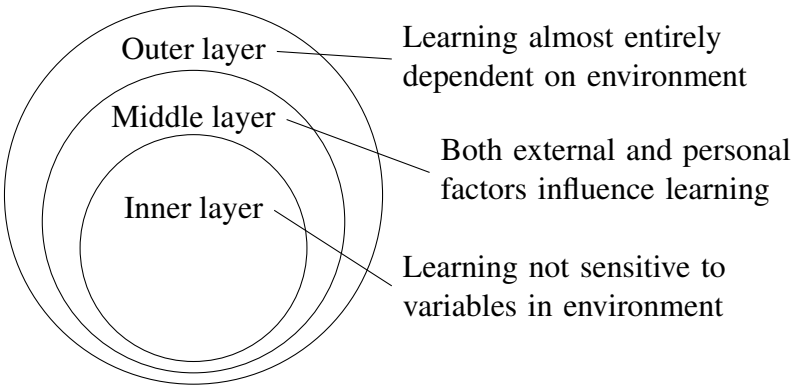

Figure 4: Curry's onion model of learning methods [1].

semester courses and having fun.

Next, we asked some of the local IT companies if they would like to sponsor the event, and got positive responses from three of them. Together with equal opportunity funding from the University, this let us buy equipment for the Raspberry Pi and electronics courses, and partially fund the instructors.

After making a schedule for the classes we now had to offer, we sent out an invitation to all the applicants to the Faculty of Mathematics and Natural Sciences to participate at the summer school. A little later we also sent out an invitation to some of the Bachelor students at the Faculty, as well as to all the local high schools.

In the end, 39 students signed up – half of them women – which was a suitable number for our first foray into making a summer school.

5 Discussion

Teaching Philosophy

When talking about teaching methods there are often assumptions made about the influence of personality versus environment (such as teachers, classmates) on a student's learning. These assumptions can often be implicit.

The extent to which students' personality influences their learning, and the extent to which environmental factors and teaching can have any influence, varies. According to Curry's onion model of teaching methods [1], the inner layer consists of students' solitary activities, such as reading a book. The middle layer consists of activities in groups and labs, with active participation of both the student and his or her environment. The outer layer consists of activities where the student is quite passive, such as a lecture. Figure 4 shows the three layers of learning methods applied by students, according to Curry.

A meta study of the response of female versus male students to various learning styles concludes that there are some differences between the genders in the methods from the middle and outer layer [6]. In the *middle layer* theories (using Kolb's theory on experimental learning [5, 3]) they found a small but consistent difference: abstract conceptualisation was preferred by males, while women tended to prefer concrete learning styles, when both they and their environment was active. In the *outer layer* they found that women often want to learn for the sake of learning, while men wish to take a course for the qualifications they offer. They also found that women often take a surface approach to learning, where they learn to reproduce, while men often take a deeper approach where they learn to solve new problems, when the students are passive participants in their own

learning.

In the light of these gender differences in learning style preference, we wanted our methods and topics to include concrete learning and fewer abstract concepts in the labs, while not ignoring more abstract concepts.

Another gender issue that tends to crop up, maybe especially in computer science and technology, is a lack of confidence and a certain amount of anxiety among female students [7]. This could for instance result in a lack of confidence to ask questions and respond to questions in front of a group or class. We tried to add teaching mechanisms that would counteract such tendencies, as far as our resources would allow.

Of course it may be an issue if instructors are too conscious about these statistical differences between genders in learning methods and apply them to individual students. We only used our findings about gender differences and learning in preparing our classes and what methods to use, and not during the actual classes.

Evaluation and Experiences

As our emphasis was on education rather than running a controlled experiment, it is hard to draw strong conclusions from the data we have – one should rather consider this as an experience report.

Teaching Style: In keeping with the findings about teaching styles that would be appealing to female students, we tried to give the students concrete learning goals. This would in part have a more long term effect on the participating students, but if we succeeded it might also show up in how the students evaluated the course, in that some students may be less happy which such a style.

This does not seem to be the case. All the students replied that they were satisfied, very satisfied or neither satisfied or dissatisfied with the teaching, the help they got during exercises and what they learned from their courses. No one said they were dissatisfied or very dissatisfied on any of these points for any of the courses, and the replies had no typical gender patterns. (There may have been a slight tendency toward male students being more satisfied.) However, we do not have any comparable data to see whether more male-oriented teaching would have left the female students dissatisfied, or if we actually managed to be concrete in our teaching, as our inquiry into female learning styles said would be the female-friendly pedagogy choice.

Hands-On Approach: In order to make our teaching style concrete, we made the students write code themselves as much as possible. We did not use the more traditional divide between theory in a lecture, separated in space and time from exercises and activity from students. We also put our primary teaching resources into the lab with the students.

As the primary learning objectives for the students were programming, which they would practice and mainly learn in the lab, we could not justify a division between theory and practice or leaving the students in the lab with less experienced assistants.

The instructors found this to be a very positive experience. Watching the screen of the student learning to program gave an intimate understanding of the learning process and any problems the students were faced with. This can be socially useful with beginner students that may not yet possess the vocabulary to ask the right questions for their problems and who may need some time to learn how foreign software tools function.

In order to facilitate student-teacher communication in class we used assistants that casually walked among the students during instructors demonstrations and during

exercises. These assistants would monitor any problems encountered and help students that could not follow the classroom demonstration or explanation. If there was a general problem among the students, the assistant could throw the question up on the blackboard, preferably with the use of the students vocabulary. The assistants could also slow down or speed up the demo as the students required.

The students getting used to hearing questions formulated in a manner they could manage to do themselves, seemed to have the desired effect, and we quickly got all of our students to ask plenty of questions.

Happiness and Attitude: 67% of the female students said *Nerding for Newbies* had changed their attitude towards informatics studies, whereas only 9% of the male students reported any change in attitude. How the attitudes had changed and what caused these changes is, however, uncertain. Later informal discussions with some of the female studentes revealed that they feel somewhat more confident about taking informatics courses.

Interestingly, 100% of the female students answered they would participate if we organised *Nerding for Newbies* courses in the evenings during the school year. Only 27% of the male students were wholly positive to this, and an additional 36% of the male students would participate if there was a topic of special interest.

New Students: One of our goals with *Nerding for Newbies* was to prepare new students for starting university or a programming course at the department. This could help students feel less anxious about the courses this fall. We recommended those who were beginning a degree in informatics to take the Java course and all the others to take the Python course. (At the department the beginning course in programming for bachelor degrees in informatics uses Java, while the beginning course in programming for other bachelor degrees uses Python.) Many of them listened to us on this, but some also took both courses.

Another goal was to give an impression of informatics and what they could expect to encounter if they studied informatics at UiB. We expressly stated that it was mainly the programming courses that gave an impression of any future studies at the department, and we think they had a clear idea of what further studies would entail. Again, it is hard to say after such a short time, if we reached this goal.

University Students: During our planning, we did not fully anticipate how many university students we would attract from other fields – almost half of our students were already at the University. For these students, computers and programming are important tools, and some of them (e.g., from physics and nanotechnology) have basic programming on their list of recommended courses. Several of them are now (Fall 2014) taking the "programming for natural sciences" course. One reported that she had been skeptical of programming, but now she had self-confidence with computers and considered also taking more advanced programming courses. Another was already doing IT-studies, but had problems with programming – she is now retaking her introductory programming course with new self-confidence, and now feels better prepared to solve problems on her own.

In addition to the programming courses, these students were also particularly interested in the statistics and LaTeX courses. Several reported having been skeptical of

the Raspberry Pi course at first ("what's the point?"), but found it to be a "fun way to demystify computing" in a "relaxed athmosphere".

Guidelines for Potential Organisers

Based on our experiences, we have the following recommendations for those who consider organising a similar summer school:

- Be careful with having multiple parallel tracks. Some of our smaller courses were cancelled because the students concentrated on the main ones.
- Both teaching and organisation takes time and mental effort. In particular, combining the role of organiser and teacher was difficult – it seems best to have dedicated personell for these tasks, ideally one person for administrative tasks and one with overall responsibility for the teaching.
- Using talented students as teachers works well, and also gives them valuable experience. Obviously, they should be selected on the basis of pedagogic and social skills, in additon to academic skill.
- The Raspberry Pi is an excellent platform for teaching general computing skills, and giving the students the courage to experiment and figure things out for themselves.
- Use lab-oriented teaching as much as possible.
- The students may have widely varying skill levels, so having assistants available to give individual help is a good idea.

6 Conclusion and Future Work

As stated, it is difficult to draw strong conclusions about the effect and impact of our summer school, but it also seems clear that this was a welcome event amongst the students, and the instructors all learned and grew with the experience.

Several of the staff and PhD students at the department have praised the initiative and some have also asked to be instructors at any future events. Study advisers, students, sponsors and staff all seem to take it more or less as given that we will do this again, either next summer or as evening classes during the school year, or both.

We would make a few changes in both preparation and how we implemented the summer school. One change would be to give the instructors more support and mentoring from more experienced instructors at the department regarding their preparation of courses. This in no way reflects any dissatisfaction with their prep work or how they organised or instructed classes. It would simply be more educational for them.

Next, we would make the schedule a bit different, and maybe make tracks for different student groups. That way it could be easier to prepare the right amount of assistants in classes as well. We would need to discuss whether or not to have the students preregister for courses.

It would be better for the students if someone was in charge of organising a few social events in the evenings. They seemed quite interested, but we were not as well prepared as we should have been on this front. Most importantly, we would have started preparation much earlier and also let the potential students know about any events much earlier.

Overall, though, everyone working on this project enjoyed the experience and we feel inspired to organise future events.

Acknowledgements

Many thanks to the study advisers at the Department of Informatics at UiB, our assistants, and Webstep AS, Bouvet AS and Knowit AS for their support.

References

[1] L. Curry. An organization of learning style theory and constructs. In *67th Annual Meeting of the American Educational Research Association*, pages 1–28. Education Resources Information Center, Montreal, 1983. URL `http://eric.ed.gov/?id=ED235185`.

[2] A. Fisher, J. Margolis, and F. Miller. Undergraduate women in computer science: Experience, motivation and culture. In *Proceedings of the Twenty-eighth SIGCSE Technical Symposium on Computer Science Education*, SIGCSE '97, pages 106–110. ACM, New York, NY, USA, 1997. ISBN 0-89791-889-4.

[3] R. R. Hayden and M. S. Brown. Learning styles and correlates. *Psychological Reports*, 56:243–246, 1985.

[4] K. B. Hopkins, A. V. McGillicuddy-De Lisi, and R. De Lisi. Student gender and teaching methods as sources of variability in children's computational arithmetic performance. *The Journal of Genetic Psychology*, 158(3):333–345, 1997. ISSN 1940-0896.

[5] D. A. Kolb. *Experiential Learning: Experience as a Source of Learning and Development*. Prentice-Hall, Englewood Cliffs, NJ, 1984.

[6] S. E. Severiens and G. T. Ten Dam. Gender differences in learning styles: A narrative review and quantitative meta-analysis. *Higher Education*, 27(4):487–501, 1994. ISSN 0018-1560.

[7] D. Stoilescu and D. McDougall. Gender digital divide and challenges in undergraduate computer science programs. *Canadian Journal of Education*, 34(1): 308–333, 2011. ISSN 1918-5979.