Spring 2009

Misconceptions and Computer Science

Bradford Larsen
University of New Hampshire

INQUIRY journal
UNH UNDERGRADUATE RESEARCH JOURNAL SPRING 2009

commentary

Misconceptions and Computer Science

—Bradford Larsen (Edited by Travis Taylor)

In any profession there is a great deal of confusion among laymen as to what its practitioners actually *do*. On several occasions through the past few years, I have had run-ins with acquaintances, relatives and family friends who inevitably want to know how I have been spending my life. When I tell them I am studying computer science, they usually respond by telling me about their home computer problems. This is *not* what computer science is about. Alternatively, they ask what I plan to do after I finish my studies. Most people find my response of "professional research" to be as puzzling as the study of computer science itself.

The huge variety of computer fields undoubtedly causes much of this confusion. Many people do work in Information Technology (IT), maintaining and repairing computers. Many people are programmers, who implement pieces of software. Many people are software engineers, who both implement and design software. Other people are computer scientists, who conduct research regarding computers. And then there are innumerable specialties in each of these areas. (Of course, these categories are not so cut–and–dried as I present here, but hopefully they are illustrative.) Undergraduate computer science studies are, for many people, a launch pad into many of these careers.

The author with Fortress Marienberg in the distance.

I want to conduct research professionally for three main reasons: I will almost certainly have more freedom as a researcher; my work will have a higher chance of being influential; and I have great curiosity, which research will help satisfy. This desire was strengthened when I successfully applied for an International Research Opportunities Program (IROP) grant from the University of New Hampshire. I spent the summer of 2007 in Erlangen, Germany, where I worked alongside Ronald Veldema, leader of a research project to make supercomputers easier to program.

Distributed Computing Research

Modern supercomputers are typically made up of thousands of individual computers, not so different from the desktop computers found in libraries and offices, networked together. These are also called *cluster computers*. Effectively programming cluster computers is much more difficult than programming a single computer; and so to address this problem, Ronald and I experimented with an implementation of the Java programming language that would make a cluster computer appear to the programmer as a single computer. (Java is one particular programming language that appeared in the early nineties and quickly gained huge adoption in the industry.)

In order to get computers to do anything useful, they must be given precise instructions. Rather than write these very detailed sequences of instructions directly, programmers usually use higher–level programming languages. The writings produced in these languages are then converted into the very detailed sequences of instructions that computers can execute with other programs called *compilers* or *interpreters*.

Normally, any piece of data in a cluster computer will exist at only one of its individual computers. If a computation occurring on another computer in the cluster needs to access that piece of data, the computation and the piece of data need to be brought to the same place. To create the illusion that the many individual computers are one, our Java implementation would have to take care of all the details of arranging a meeting between computation and data. This is the problem of *remote access*.

My role in the research was to implement a performance optimization: data caching. In certain cases, a program written for our Java implementation would run faster if often used data were copied to several individual computers in the cluster because accessing remote data could be done faster.

Probably Not What You Think It Is

For ten weeks I spent about thirty hours per week in the lab at Friedrich-Alexander University. Understand that a computer science lab typically is quite boring, lacking test tubes, meters, scales, rat mazes, x–ray machines, or any of the interesting things one would expect to find in a science lab. In my case, "The Lab" was a room on the fifth floor of a mammoth building with about ten computer terminals, a whiteboard, and plenty of windows. The specific line of research Ronald and I were conducting could be done without special equipment; all we needed was a cluster computer to test with and a conventional desktop or laptop computer for development.

The building housing most of the mathematics and computer science departments at Friedrich–Alexander University of Erlangen–Nuremberg

It took me about a week in the beginning to become familiar with the existing work of this project. I spent much of that time asking Ronald questions, poking around through the project's source code, and writing some simple programs to test the project. Once I had found my bearings in the project, we tossed ideas back and forth for a couple days, discussing my high–level strategy, and the algorithms and data structures I would use.

The concept of *algorithm* is extremely important in computer science. An algorithm is often compared to a cooking recipe: the recipe gives a sequence of detailed instructions, which, if followed precisely, will result in a certain food. Algorithms are sequences of instructions, which, if followed to the letter, will take input and give output satisfying certain guaranteed properties. Algorithms are specified in such a way that they are independent of any particular programming language, similar to the way recipes are written independently of a particular brand of measuring cup.

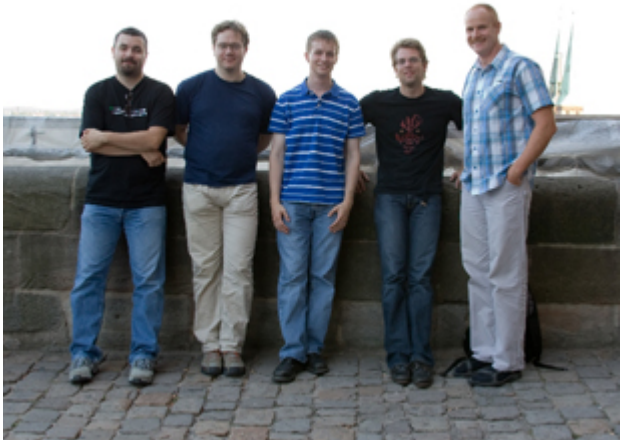
After our initial brainstorming session, I spent a few days programming what we had discussed, stopping only when I ran into an unforeseen issue or when I needed additional guidance. At such points we would brainstorm further. Rather than being a solitary, elusive activity someone did in a dark room with a computer, my work required lots of thought and interaction with others. For several weeks we repeated this process, and I implemented many of the features we had initially discussed. Midway through, I presented my preliminary results to the faculty and graduate students in my group.

Opposing Approaches

One thing I took away from the summer's experience is how different two people's work styles can be. I have a perfectionist streak in me. I find it jarring when my programs and work are inelegant, and I will often go back over things and rework them, even though they might perform functionally. "Obsessive" is not an entirely inaccurate description of my work process. Speaking of a program as "elegant" may seem strange, but it is similar to mathematical elegance, in which a simpler, more concise program is more concordant. (This idea is similar to Ockham's Razor.) Simply put, I don't like clutter in my code.

Rewriting inelegant code can have practical as well as aesthetic considerations. Trudging forward and continuing to work with a needlessly convoluted solution can be much more error prone and take more time than redoing parts from scratch. However, it takes time and experience to realize that a program is convoluted and how it might be better written. While in Germany, I rewrote major parts of my code twice, when the design I had been working with proved to be no longer sufficient for further development.

Ronald, on the other hand, did not share my obsessive perfectionist streak. He would work quickly, produce something that performed functionally; and then maybe in the future go back and redo it, once he was more familiar with the task in question and the earlier prototype was noticeably slowing down further progress. Because he was not so concerned with elegance, he was able to quickly prototype solutions to our problems, producing results sooner. I, in contrast, had to be continually on guard against my natural tendency to try to make everything perfect all at once just to keep up with him.



The author with students and faculty of the Programming Systems computer science group at Nuremberg Castle. From left to right: Dr. Michael Klemm, Dr. Ronald Veldema, Bradford Larsen, Alexander Dreweke, Dr. Michael Philippsen.

He would often sit beside me when we did our work, especially at the beginning when I was getting my bearings, and would often tell me, "You spend so much time formatting!" or "Don't think so much; just type! Try it!" or "Hands on the keyboard!" These urgings went against my natural tendencies, but at the same time kept me focused and helped me to make progress faster than I would have otherwise.

However, occasionally this hands–on–the–keyboard approach slowed development down. About six weeks in, I spent close to two weeks looking for an error in our code that only sporadically manifested itself. It was a subtle bug involving interaction of my data cache and the garbage collector (the subsystem that frees memory when it is no longer used) that resulted in the system crashing. With a slower, more regimented methodology, this problem could have been avoided, saving me two weeks. Balance is hard to strike.

A Happy Medium

In programming and computer science, attention to detail is critical. It's quite a binary distinction: your code/algorithm/system/etc. is either completely correct, or it isn't. In many cases, if it is not completely correct, it is catastrophically wrong. Computer scientists must work with great concentration and focus, or they produce rubbish.

I learned from my experience in Germany that there is a balance to be struck between elegance and progress: If one tries to create a masterpiece right from the beginning, it will take forever to show any results. On the other

hand, if one is never concerned with elegance, it can result in unforeseen problems and incoherency later. My work style leaned much more towards the former; Ronald's was somewhat right of center.

Ultimately we published a paper on our work at an international conference. I went through an entire iteration of the computer science research cycle with my experience in Germany: I collaborated with others, applied for grant money, performed experiments to test my hypotheses, and shared the results. This cyclical process is central to computer science research. Now I just need to work on my small talk so I can offer a better answer when asked what I have been doing.

I would like to thank Philip Hatcher for his mentoring and contacts in Germany; Ronald Veldema for providing an interesting project and overseeing my work while I was in Germany; Michael Philippsen for providing a spot in his department and giving feedback on our work; the students of Informatik II, the students of Rommelwood, particularly Michael Dlugosch, for their friendship outside the lab; Georgeann Murphy for facilitating the IROP process; and, finally, my family for being supportive of me throughout this entire experience.

Author Bio

Bradford Larsen*, originally of Nashua, New Hampshire, is a first year master's student in computer science. During the summer of 2007, with the aid of an International Research Opportunities Program grant from the University of New Hampshire, Brad conducted research in Erlangen, Germany, where he worked on a project to streamline the effectiveness of supercomputers. The project, which was large in scope and complex in its implementation, provided more than just practical experience in the field of computer science. Based largely on the work he produced while in Germany, Brad and the project's leader collaborated to publish a paper that took the "Best Paper Award" in the category of "Distributed Information and Systems" at the 2008 International Conference on Parallel and Distributed Computing and Systems sponsored by the International Association of Science and Technology for Development. Brad continues the pursuit of his graduate degree with his eyes set on the Ph.D., recognizing that his summer research experience and the resulting publication will be important in his future study and grant opportunities.*

Mentor Bios

*Professor **Philip Hatcher** has worked since 1986 with students in the Department of Computer Science at the University of New Hampshire, where he is currently chairperson. Specializing in the areas of programming languages and compilers, parallel and distributed computing, and bioinformatics, as well as once being a recipient of the "Outstanding Assistant Professor" award, Hatcher is no stranger to mentoring. In his time at UNH he has advised a number of honors thesis students and six candidates for International Research Opportunities Program (IROP) grants. Hatcher describes an IROP project as "a life–changing experience," one that is "always very rewarding" for a student. In Brad's case, Hatcher was particularly happy to see Brad working side by side with "a very intense and very bright foreign mentor," and how exciting it was to see him "rise to the occasion and meet that high standard."*

*Brad's foreign mentor, Dr. **Ronald Veldema**, has been at the University of Erlangen–Nuremberg for the last six years, where he teaches and conducts research in programming language and parallelization. As well as being Brad's foreign mentor and co–author, Dr. Veldema also introduced Brad to a new work methodology, one of fast–paced, hands–on–the–keyboard programming. While this break–neck approach didn't always mesh with*

Brad's work style, it forced him out of his comfort zone and helped him see the values (and shortcomings) inherent in any approach to solving a problem. When asked about his time spent with Brad and the work they performed together, Dr. Veldema replied simply: "Satisfactory, not difficult, fun and interesting."