Maria Deslis

csci499 - Rader - spring 2013

Reflections Paper

Our first task this semester was to learn about constraints and problems in programming and solve in a skillful and creative way. One of the biggest flaws that was pointed out to me was that I all too often jump straight into the problem without much planning and end up panicking over it when I get stuck, and thus end up wasting hours and hours in front of the computer screen getting nowhere. The three basic problem solving skills that I have gained and have benefited me greatly are hand tracing, divide and conquer, and stepping back.

Learning about hand tracing was the first big improvement in my skills as a programmer. As a visual learner, it allows me to *literally* see what I want to program, what I want to code, and how the computer will think about it. It is also versatile, in that I can hand trace nearly anything and everything - data structures, algorithms, and other ideas of what I want in a program. The “Solving Problems with Arrays” and “Solving Problems with Pointers and Dynamic Memories” sections of the course was where I got the most practice in hand tracing and allowed me to refine this skill. With this refinement, I was able to better understand how a computer thinks and my understanding of these basic dynamic structure is now stronger. By taking a pen and paper, I am able to take some of the most difficult tasks given to me and understand them at a deeper and more fundamental level.

The second tool I learned this semester, divide and conquer, is as it sounds, simply taking the problem and breaking it into smaller pieces that are easier to manage. The general steps I follow using this tool are the following:

1. Make a list of all constraints stated in the assignment
2. Make a list of all stated possibilities/hints in the assignment (if any)
3. Break down the problem from the big picture into smaller parts
4. Hand trace each small part individually, so as to ensure that you are understanding:
    a. What the computer is thinking
    b. What I wanted the computer to think
    c. That the code was doing what I wanted it to do

Even after I learned how to plan the problems I was faced with, when I ended up getting stuck I would get greatly frustrated and I would begin to feel lost. This is where the third tool of problem solving comes in - stepping back. Instead of wasting hours in front of the computer getting nowhere, stepping back allows me to break the cycle. The most important part about stepping back is *getting away from the computer.* During that time I would take a small break and refresh my brain. Once I wasn’t feeling so drained, I would go back and start hand tracing and breaking down the problem all over again from the beginning, attempting to understand the problem from a different perspective. More often than not, when taking a step back I would realize my mistake - usually something small like missing or misinterpreting a step during hand tracing - and end up solving it without issue.

Another benefit of stepping back was that I learned how to ask for help. Before this class, it used to be that if could not progress any further in my code, I would simply tell others that I was "stuck" or "didn't know what do". Obviously this didn't help me, or anyone else. No one knew what I wanted help with and I was too frustrated to make sense of what I was stuck on. Although I have not gotten it to a point where I can explain my

problems with perfect clarity, I have improved drastically. Instead of just saying "Help, I don't know what to do. What am I missing?" I take the time to read through my code first and think about what I wanted the "broken code" to originally do. Then I try to find out what it is actually doing. If I am still not sure as to why I am getting results that are different from what I want, then I am at least able to tell people "This is where I am stuck. This is what I was trying to do, but I got this instead." or "I have pinpointed the problem to here." As time went on and this happened more and more often, as soon as I encountered a problem that I could not immediately fix, I would step away and go through the hand trace again. With each assignment I improved, and even though I still ended up turning in some assignments late, I was usually only a step or two away. Ironically, by learning how to ask for useful advice, I have helped lessen the need to even ask for help from others and have learned to seek help from myself. In the end, stepping back taught me how to ask for specific help, which helped me save tons of time and energy when working, and increased my skill and confidence.

After learning these problem solving skills and gaining a stronger foundation of data structures, the next step in building my skills was learning and refining how to refactor, test, and encapsulate code.

I found the best way to refactor code was to make use of the Divide and Conquer tool. By programming small parts of the code, I was able to test my code consistently as well as ensure that my code was clean, optimal, and D.R.Y. This way, by the time I was done with the project I knew that it was correct and in the best form it could be.

For me, learning how to test code was like learning how to paint. There is no right way to learn how to do it other than to do it over and over. With time my skill increased and testing code became easier. Not only did it force me to think of multiple possibilities of how my code could break, but I learned it meant that there wasn't necessarily an absence of errors either. No matter how many ways I can currently think of breaking my code, there are still many ways it can break that I simply have not considered yet. There were many times where I thought I had all of the possibilities only to discover the opposite and have to refactor my code. This isn't a skill that I have honed or have mastered yet, but I do know that this is a skill that has improved since I have started it has been a necessary part of almost all the code I create now. Constantly trying to thinking of different ways of how my code could break and then trying to solve it is truly an art form.

Similar to how refactoring my code allows me simplify and clean up code, learning how to encapsulate taught me how to organize code; keeping like code with like. Rather than having a long list of procedural code, encapsulation allows me to rely on the development environment to find the functions for me. This way I can focus on the part I want/need to focus on one at a time, allowing my mind to work with me, rather than being forced to think about things in a way that doesn't work for me. When learning Linked Lists and Binary Trees this semester, encapsulation came in shining. I was able to work on what I felt was the easiest part and work my way up to the harder parts in the code. This allowed to me to apply my problem solving skills and practice testing and refactoring with each part of the code I was working on without any issues.

As time went on with this course, I ended up getting more and more compliments from my professors about my code. I was also able to solve big projects in class on my own *and* get high marks for them. For

example, in Algorithms, our first major project was the Traveling Salesman Problem. When we received the assignment, individual or group, it still terrified me because I had already convinced myself that I didn't have the skills needed to do even remotely well. As a result, in the beginning of the semester and I would keep putting it off the assignment because I didn't want to deal with it. My original thought process was, *I'm not gonna do well anyway, so why bother?* But I reminded myself that if I was going to get any better, I had to start somewhere. So I sat down away from the computer and pulled out my pencil and paper, and proceeded to apply the tools I was learning in this course. As a result I not only ended up finishing the project in record time, but I completely understood what was happening *and* I got an A. For a long time, I didn't think I was capable of this anymore and here it was happening. That's when I began to ask myself, *What else am I capable of? Was I really able to do this the whole time and just didn't know?* Before I knew it, I was applying these same steps to nearly every single coding project and class I have had this semester.

In Principles of Programming, when I was stuck on creating the NIM game, I was able to talk my friends and I through the project by breaking each part of the game into small pieces and hand tracing. Even though I didn't get the code quite right at first, I knew that I was doing well because not only did I understand what was happening in the assignment but because I was able to help my friends understand what was going on as well. In the end, we solved it and we were able to complete the project with all the requirements.

Another example was in my Databases class, where we often had to do projects in languages we weren't familiar with. However, that class also showed us how to make visual representations of what we wanted our database to look like. This reminded me of hand tracing, and once I was able to understand how to make a model that anyone could read and understand, then making complex databases wasn't an issue. With each project, my code improved even when I wasn't using a language I was used to. Not only did I receive compliments from my professor about my code, but he even displayed it to the class and pointed out all the good things and encouraged everyone to follow.

In the words of Edsger W. Dijkstra, "The competent programmer is fully aware of the limited size of his own skull. He therefore approaches his task with full humility, and avoids clever tricks like the plague." Before I started this course, my biggest problem wasn't just that I was missing some fundamental knowledge, it was that I lacked confidence in my abilities. I felt completely helpless in everything I tried to program; each assignment felt like I was starting from nothing. I was too embarrassed to ask for help because I was convinced everyone else already knew these things, and I was left behind. And when I did ask for help, I felt like I was being a burden on my peers and professors. But with time, as I grew in my abilities the feeling of complete helplessness was disappearing slowly but surely. Now I feel like a truly competent programmer, approaching each problem with humility and a proper attitude rather than feeling overwhelmed and being crushed by anxiety. In this course, I gained more than just the necessary foundation I was missing, but also the confidence to build onto that foundation as well.