

Writing Assignment 1 (Warm up) “My Transformation”

Please identify and describe a computing concept that transformed the way you see and experience computing. Include in your discussion how you experienced the transformation process. This experience must have left a lasting impression on you. I have included two example “biographies” below: the first one discusses *modularity*, the second *complexity* and *abstraction*.

Specific instructions: write 3-5 paragraphs, as follows:

- Organize your answer using an inductive method (from specific to general).
- In the first paragraph, convey briefly your transformational computing experience and the lasting impression it left you. No generalizations in this paragraph.
- In the middle paragraph(s), expand on your experience, making it evident how the impression was derived. What other computing concepts did you need to understand in order to gain an understanding of this concept?
- In the ending paragraph, draw larger conclusions or generalizations about your experience. How you will approach computing in the future, as a result of having this experience? How did your understanding of this concept affect your understanding of other computing things?

Write your story in plain text or in MS Word, and submit it via email to robert@engr.uconn.edu by the end of Friday, January 25.

The example stories:

Sample Story 1: Discovering modularity

My first transformational computing experience occurred in the eighties when I programmed a sink-a-ship game using the Basic programming language on a borrowed computer. The program was minimally interactive with a user interface that was not very sophisticated. This was one of the first programs I ever wrote, and I did not know much about programming. I discovered how parts of the code could be reused to make my code elegant.

For programming the sink-a-ship game, I only had a language reference that did not tell how to program; it only, more or less, listed the possible keywords in the language. My code became very long and complicated and hard to understand. As that game was represented by matrices and the algorithms searched them all the time in similar ways, I had a feeling that my code was redundant, and that there must be a better way. It was at that point that I discovered the `gosub` command, and I understood how parts of the code could be reused in a much better way, as compared to using the `gotos`. This was the first smart thing in programming that I learned to appreciate. There were functions of course, but it was the `gosubs` that really fascinated me at that time.

Upon incorporating `gosubs` into the code, it became beautiful in a sense and I felt proud of it. It was not just a sequential list of instructions anymore. I felt the passion of programming and first saw the difference between code and elegant code. I learned to always modularize my code and to pay attention so I could avoid repeating the same code.

Now I try to start designing my programs in a modular manner instead of rewriting code. This experience also taught me to pay attention to the subtleties of programming so my future programs would become even more elegant.

Sample Story 2: Abstraction made the light come on

When I first started studying computing, I thought it was all about writing programs, and being really clever about programming. I was cool with it; there were loops, ifs, and functions allowed me to break down my code into segments. But it seemed pretty technical and boring to me. When I learned about analysis of algorithms though, everything changed. I now saw a bigger picture of computer science, saw computer science more abstractly.

While I was programming, I had always thought informally about program efficiency as how long it took a program to run. I knew that nested loops took much longer than a single loop. I discovered that recursive routines sometimes took the same amount of time as a single loop, sometimes took more time. I didn't think much about different solutions for the same problem.

But after learning about complexity, I now used big-O as the measure. For example, an $N \log N$ algorithm instead of an N^2 algorithm is best for sorting a sequence of numbers or strings. And linear recursion is similar in complexity to a single loop, but non-linear recursion takes more computing time. I discovered that I could compare algorithms for efficiency without even having programs.

This was a turning point for me, to be able to see computer science more abstractly. From then on, even though I wrote programs, I thought about the abstract aspects. For example, when I took compilers and I used grammar rules to build a parser, I recognized that they were like the grammars I saw in the automata theory course. These were pretty theoretical and mathematical. I knew if I found other uses for grammars, I would understand how to adapt them to the problem. While I probably will get a job writing code, now I see the big picture. I can use algorithm complexity and abstract many of the fundamental computing concepts so that my code becomes better, richer.