Programming Language Second Essay
- Getting over the inertia of technology

SNUCSE
2005-11767
Park, Ji-sung

Abstract

기술에는 관성이 존재한다. 우리가 쓰는 QWERTY 키보드 입력방식은 매우 흔히 거론되는 예 중의 하나이다. 타자기로 문서를 작성하던 시절 너무 빠른 속도로 자판을 치면 타자기가 엉켜버렸기 때문에, 쓰는데 더 많은 시간이 걸리도록 설계한 것이 지금 쓰는 방식이라 한다. 이후에 더 편리하고 빠른 방식들이 나왔지만, 기술의 관성을 극복하지 못하고 다수에게 외면당한다. 한 번 굳어진 방식은 더 획기적인 발상이 나와도 이미 그것을 사용하고 있는 곳이 많고, 많은 사람에게 익숙하기 때문에 바꾸기가 어려운 것이다. 문제는 이렇게 불합리하게, 뛰어난 기술을 '기술의 관성'이 사장시키는 경우나 불합리함까지 고려하여 설계하도록 만드는 경우가 많다는 것이다. 이전 주류 CPU가 가지던 오류를 그대로 내도록 (어이없게)설계된 CPU라든지, 그 밖에도 혁신적인 기술임에도 불구하고 예전 제품의 interface를 지키기 위해 쏟아 부은 engineer들의 노력은 (쓸데없게 느껴질 양만큼) 정말 많다.

그러한 것들은 공학도의 길을 걷고 있는 나에게는 개인적으로 굉장히 슬픈 일로 느껴지며, 좌절감을 준다. 만약 내가 좋은 것을 개발해도, 이미 시장은 장악되고, 사용자들에게 익숙해져있는 것은 정해져있는데, 결국 그런 것을 따라가며 설계 자체를 해야 하는 것인가? 잘 못된 것은 알지만, '이미 많이 쓰기 때문에'란 이유하나로 계속 그것을 사용해야 하는 것인가? 이런 의문이 계속 생기는 것이다. 프로그래밍 원리나, 프로그래밍 언어 수업에서 scheme, nML, ocaml 등을 배우고 내 프로그램을 설계해보며 Functional language의 우수성과, 이전 언어들의 부족한 점을 많이 느꼈지만 이 부분 역시도 세계의 주류인 C나 C++, Java들을 결국 쓰게 되지 않을까 하는 생각이 들었던 것도 사실이다.

하지만 이번 두 번째 Essay를 위한 글들을 읽으며, 나는 여러 사람이 당연하다고 생각했던 관성을 과감하게 따르지 않고, 그 따르지 않은 자신들의 선택이 옳았음을 증명했던 여러 글들을 보며 깊은 감명을 받았고, 무언가에(?)

통쾌함도 느꼈다. 이 글에서는 그 글들을 요약하고, 느낀 바를 간단하게 정리하는 시간을 가져보자 한다.

Lisp : Is it better? then take it!

세 개의 글 - Beating the Averages, Revenge of Nerd, Toward Programmer Interacivity : Writing Games In Modern Programming Languages - 는 모두 Lisp이란 언어의 위대함과 그를 이용하여 세상의 통념을 깨고 오히려 남들 보다 성공한 사례를 보여주고 있다. 이 한 마디가 모든 것을 요약한다.

"So if Lisp makes you a better programmer, like he says, why wouldn't you want to use it?"

가슴을 치게 된다.

'맞아! 바로 그거야!'

Essay를 준비하기 위해 읽는 글을 보며 이렇게 통쾌함을 느낄 줄이야. Lisp이란 언어를 써보진 않았지만, 그와 유사하다고 알려진 scheme이란 언어를 프로그래밍 원리 시간에 반 학기 이상 쓴 적이 있다. 한창 2학년 무렵에 Java와 C에 대한 자신감이 쌓여가고 있었던 때였다. 실습시간에 처음 Introduction class를 들으며 생각한 것은 '이것을 어디에 쓸꼬?'였다. 무수한 괄호 속에 문법이란 없어보였고, 이런 걸로 내가 자료구조 수업 때 설계했던 프로그램을 짤 수 있을까? 이게 나중에 나에게 도움이 될까? 이런 생각들은 덤이었다. 하지만 계속 수업을 듣고 헐떡거리며 숙제를 하는 동안, (가장 먼저는 자료구조 숙제는 내가 설계한 것이 아니라 덕지덕지 붙인 그야말로 Adhoc Program이었다는 것을 느꼈고,) 과연 내가 이전에 배운 언어로 이런 생각의 프로그램을 표현할 수 있을까? 설사 짤 수 있다 해도 이것처럼 5줄 이내로 짤 수 있을까?(물론 프로그래밍에서 짧은 프로그램이 좋다는 것은 매우 유치한 발상이지만, 여기서는 짧다는 것 보다는 프로그래머의 생각을 효율적으로 표현할 수 있다는 데에 중점을 둔 것이다.) 이런 생각들이 들었다. 거기에 더해서 느낀 안타까움은 이 언어는 매우 좋지만, 아무래도 주류인 C나 Java보다는 나중에 내가 쓸 일이 없겠구나. 이것도 기술적 관성의

피해자 중 하나이구나. 이런 생각이 들었다. 좋은 데도 쓸 수 없다. 또 한 번 느끼는 무력함과 우울함이었다. 그런데 저자는 그런 나의 생각에 정면으로 묻고 있었다.

'왜 안 돼? 좋으면 쓰면 되지!'

정답이다. 누가 저 말에 반박을 할 수 있단 말인가? 거대한 흐름에 거스를 수 없다고 생각하는 것은 과연 대학생인 내가 가져야할 생각인가? 좋으면 좋은 것을 쓰고, 남들도 그것을 쓰도록 해야 하는 것이 아닌가? 거기다가 성공까지 했다는데,(어쩌면 성공하는 게 당연하겠지만) 반박할 논리가 없다. 실제로 우리가 쓰는 프로그래밍 언어는 그저 Tool이 아니라, 우리가 생각하는 방식을 바꿔놓는다. 감히 잘난 척 할 순 없지만, scheme과 ML language에 익숙해지고 난후, 어떤 문제를 recursive하게 보는 능력이 많이 길러졌다는 것을 느낀 적이 많다. 다른 과목의 숙제를 위해 함수 등을 만들 때, 그런 재귀적 발상이 굉장히 빨라졌다는 느낌이다. C나 Java등도 물론, 세월을 거치며 점점 더 많은 API가 추가되고, Abstracition 기술도 발달하고 하면서 나날이 발전 중이고, 잘 만들어졌으니 지금까지 사용되고 있다는 것은 부인할 수 없지만 분명 더 나은 선택이 존재한다는 것 또한 확실하다. 우리는 무엇 때문에 망설이는가? 그리고 그것은 정말 망설일 정도의 이유가 되는가?

첫 번째 Essay에서, 나는 이렇게 피력했다. 분명 Type system이 있고, 프로그래머의 생각을 잘 표현할 수 있는 언어이므로, "한계가 있지만" 우리는 "사용할 때에 맞춰" 그것을 사용해야 한다고. 한계가 있다. 이것은 어쩌면 기술의 관성에 이미 순응하려하는 나의 모습일 것이다. Computer Architecture나 low level system design에서도 분명, 우리가 경험하는 기술적 관성은 존재한다. lambda calculus에 근거한 수학적으로 입증된 system을 설계하는 것을 꿈꾸다가도 이미 Intel과 Arm이 양분하는 그 시장에서 새로운 CPU가 살아남을 수 있겠는지에 대한 생각은 그런 훈훈하고 설레는 생각을 쏙 들어가 버리게 했던 것이다.

그런데 이 글들을 읽고 그런 '현실적인'고민에 대한 답이 많이 나온 것 같다. 좋으면 써야한다. 잘 못된 것이 있으면 고쳐봐야 한다. 꼭 당위적 문제 때문이 아니라, 그런 것들은 좋은 결과를 내고, 어쩌면 그런 노력들이 있었기에 기술이 잘 발전된 것일지도 모른다는 생각이 들었다.

What is good design?

과거부터 현제에 이르기까지, '미'라는 것은 가장 오래전부터, 아주 높은 우선순위로 사람들이 추구해온바 이다. 아름다움 이라는 것은 그야말로 시각적으로 예쁜 것으로부터 출발하여, 잘 짜인 구성, 정밀한 조직, 세심한 배려에까지 어디서나 느낄 수 있는 것이다. 요즘 들어 공학에서도 design이란 것은 아주 큰 화두이다. 미끈한 기계의 Body뿐만이 아니라 신선한 user interface, 사용자를 배려한 세심한 기능들까지, 어떻게 설계하느냐에 따라 시장에서 사라지기도 아니면 누구도 넘보지 못할 자리를 차지하기도 한다.
Taste for maker 의 저자는 좋은 design에 대해 말하고 있다. 단순해야하며, 알맞고, 때론 자연을 닮았고, 등등의 조건들은 모두 쉽게 수긍이가고 어떤 것은 그래서 내가 좋게 느꼈구나 하는 생각을 들게도 한다. 컴퓨터 공학에서, 우리의 design에는 어떠한 아름다움이 있을 수 있을까? 간결함, 우리의 사고와 맞는 Algorithm, 무결함 등이 우리가 추구해야할 설계상의 아름다움일 것이고, 그에 추가하여 외적인 아름다움과 사용자들을 만족시키는 창의적 기능 또한 추가해야 할 것이다.

또 하나 느낀 것은 글을 읽으며 나오는 여러 예들을 읽으면서, 분명 이 저자는 (다른 글을 보면)자신의 전공에서도 아주 뛰어나면서도 어떻게 이런 분야에 대해서도 이렇게 잘 알고 있을까? 이런 생각이 들었다. 물론 천재는 여러 방면을 두루 잘 한다는 사실을 몰랐던 것이 아니고, 실제로 주위에도 여러 가지를 다 잘하는 친구들을 못 본 것도 아니지만 이런 사람들을 보면 나의 무력함을 느끼는 게 사실이다. 내가 아주 재미있게 들은 과목에서 어떤 교수님은

'자기의 전공이 아닌 모든 것에 대해서 알아야하고, 자기의 전공에 대해서는 정말 모든 것을 알아야한다.'

라고 하셨다. 컴퓨터 공학은 전분야로 확장해 나아갈 수 있는 분야이기 때문에 더더욱 이 말은 공감이 된다. 좋은 Design이 어떤 조건을 만족해야하는 것인지 알아도, 그 조건이 정말 어떤 방식으로 구현될 수 있을지 상상하지 못한다면 그것은 의미가 없다. 공학도로써 내가 키워야할 소양이 무엇이 더 있을지 생각해보는 좋은 계기를 마련해 준 글이었다.

Growing the language : Let's grow our language together!

WEB 2.0은 예상대로 여러 가지를 바꿔 놓았고, 새로운 것들을 만들어냈다. (물론 몇몇 시대에 뒤떨어진 사람들은 그 위력에 놀라고 혹시나 그것들이 자기 기득권을 해칠까 염려하여 인터넷의 위험성만을 부각시키며 통제하려 들지만)인터넷은 여러 사람의 노력에 의해 기술이 좀 더 멋지게 발전할 수 있게 하고, 지식의 공유의 범위를 전 세계적으로 확장시켰다. 대표적인 예가 Wikipedia이다. "집단지성"의 결정체라고 불리는 이 방대한 백과사전은 여러 전문적인 지식까지 모두가 키워드 하나만으로 검색가능하며 누구든 더 옳은 내용으로 수정하고 보완할 수 있다. 비전문가들의 오합지졸 지식창고가 될 거라는 우려와는 달리 점점 더 보완돼가는 지식은 백과사전을 더 완벽하게 만들어가고 있다.

Growing the language 의 저자는 프로그래밍 언어를 만들어가는 과정에서 성장 가능한 언어를 만드는 것에 대한 이야기를 하고 있다. 여러 사람이 함께 그 언어에 필요한 것들을 생각해보고, 필요한 정의를 추가하고, 그런 것들을 소수의 사람의 최소한의 관리로 제어해나가며 성장시키는 것을 제안하고 있다. 참 멋진 생각이 아닐 수 없다.

'잘 못 설계된 이전의 것만큼 후임자를 괴롭히는 것은 없다.'

이 말도 아까와는 다른 재미있는 수업에서 교수님께서 해주신 말씀이다. 기술이 발전하면서 우리는 수많은 pitfall들을 만나왔고, 심사숙고하지 않은 설계가 끼치는 어마어마한 악영향을 경험했다. 또 하나 알고 있는 것은, 하나보다는 둘이, 둘 보다는 셋이(물론 똑똑하고 호흡이 맞는 사람들이) 더 좋은 설계를 할 수 있다는 것이다. 최소한의 합리적이고 무결한 base에서부터 시작하여 여러 사람이 함께 필요한 것을 생각하고, 잘못된 것을 고칠 수 있는 것은 현대 기술이 이뤄낼 수 있는 가장 바람직한 일 중 하나 일 것이다. 저자의 제안대로 우리가 앞으로 쓸 언어가 그러한 과정으로 만들어진다면 지금같이 '도대체 무엇 때문에 이렇게 설계된 언어인가'라는 의문과 불평을 가지며 밤새 프로그래밍 숙제를 하는 여러 사람들이 많이 없어 지지 않을까 즐거운 상상을 해본다. 왜냐면, 누군가 그것을 정의 할 때는 개발 wiki에 그 목적을 남기고, 검증된 상태일 테니.

Type : Keep it sound in our common sense.

처음 만난 강력한 Type system은 nML이었다. 나중에 만난 ocaml도 마찬가지지만, 내가 정말 놀랐던 것은 이 변수가 integer라고 생각하고 프로그래밍을 하면(어떤 선언도 없이) 이 언어들은 '그것이 integer군요?'하고 알아채고, type 유추를 한다. argument의 type만 선언했던 데로 들어오는지 check했던 언어들만 써왔던 나로서는 정말 신기한 일이 아닐 수 없었다.

프로그래밍 언어 수업에서, 그런 type checking이 어떠한 원리로 설계되고 왜 맞는지에 대한 증명과 방법론들을 보면서, 고안 해낸 사람들에 대해 정말 대단하다는 존경심과, 이미 내가 쓰는 언어에 구현되어 있다는 사실에 대한 놀라움, 그리고 이런 것도 이제야 알게 됐구나하는 부끄러움이 생겼다.

ML을 쓰는 대부분의 사람들이 느끼는 감정이겠지만, type system은 처음 쓰는 사람을 당황스럽게 한다. 우리들의 상식에서는 type을 선언해주는 것이 너무나 당연하고, 그렇게 하지 않으면 이 바보 같은 컴퓨터는 잘 알아듣지 못할 것이라고 생각하기 때문이다. 자꾸만 반복되는 type checking에서 실패하는 경험은 역시 구관이 명관이구나 하는 C가 최고라는 이상한 생각을 들게도 한다. 하지만 type system이야 말로, program을 우리 상식선에 머물게 해준다. x + y * z 란 식에서 x, y, z가 수일 것이라는 것은 따로 정의 하지 않아도 우리 상식선에서 알 수 있다. type checker는 그런 이해를 컴퓨터에서 할 수 있도록 만드는 것이다. 실제로 ML로 checker가 통과되기만 하면 알고리즘의 문제가 아닌 이상 이해되지 않는 오류가 나지 않는 다는 것은 아마 많은 사람들이 공감할 수 있을 것이다.

저자는 C에 대해 잠시 언급한 후 ML의 type system을 잘 정리하여 설명하고 있다. 이미 ML의 type system이 상당한 수준에서 구현돼있다는 것을 배워 알고 있었기에, 충분히 공감하며 읽을 수 있었다. 마지막에는 Perl의 type system에 대해 언급하고 있었는데, ML과는 다르게 구현될 수 있는 type system에 대해 설명하고 있었다. 언어를 많이 배우지 않아 잘 모르긴 하지만, 상당히 흥미 있는 부분이었다.

Epilog

이번 Essay를 쓰기 위해 article들을 읽으면서, 또 쓰면서 내가 느낀 것을 한마디로 요약하자면,

"좋은 것을 하자."

이다. 세상은 옳은 방향으로 간다. 합리적이라면 선택받는다. 그런 것이 지금 내가 가져야할 마음가짐이라는 것이라고 생각한다. 물론, 현실적인 부분을 간과할 수는 없겠지만, 배우는 입장에서 하는 그런 생각은 얼마나 유치한가? 라는 생각이 많이 들었다. 그리고 여기저기서 들리는 성공담은 이런 생각에 힘을 더한다.

언제나 그렇듯 이렇게 뛰어난 사람들의 글을 읽거나, 강연을 들으면 정말 내가 해야 할 것이 많다는 것을 느끼게 되고, 가슴 벅참도 느끼지만 한편으로는 나도 이럴 수 있을까하는 의문이 든다. 하지만 분명한 것은 이런 고뇌들이 계속될 것이란 것이고, 한편으론 지금까지 나를 만들어 오는데 큰 도움이 됐다는 것이다. 믿음을 가지고, 앞으로도 많은 글을 읽으며 또 생각을 해가며 나를 발전시켜야겠다는 생각을 했다.