At first, the lecturer (Guy L. Steele) started by “defining” English terms in a way that one would do in defining a type or object in a programming language. But I noticed that his talk was composed of fairly simple and short.

Then into about 10 min of his lecture did he reveal that

“For this talk, I chose to take as my primitives all the words of one syllable, and no more… if I need to use a word of two or more syllables, I must first define it.”

That was when I realized: he was trying to demonstrate what it is like to use a small programming language and just about how restrictive it would be for one to use a small programming language.

I wasn’t even at the half of the video, but I was already drawn into this talk. As a student of CS320, a course where students learn about the underlying structure of programming language and actually get a chance to make their own language, I was one of those students that faithfully followed the coursework, did all the assignments, did all the reviews.

But I realized that I’ve never asked myself the most important question; ‘What is the meaning in learning this? Why are we learning this?’.

Basically, all the programming languages that I’ve learned/implemented such as *<WAE>*, *<FWAE>*…etc were small languages. As new features are added in the language, I wasn’t making a new language every time: I was “growing” the language to match my need for new features that I want, or features that I thought might be useful for users. But if I were to put myself in perspective of a user, I was growing the language in a specific direction that is the most optimal for my use.

Then he told the audience (and me) of the main thing that he wanted to ask:

“If I want to help other persons to write all sorts of programs, should I design a small programming language or a large one?”

And then he answers his own question right away by saying that neither is the answer: the answer is to design a language that can grow. And now I (partially) understand why he would answer the question as that.

Then I had this question of ‘If the goal of the program designer is not to try to implement everything by his/her own hand, then what should be the goal of the program designer, and how should they do their work of designing a program?’

He then explained on to answer both of the questions, both of which the answers were immensely insightful and satisfying.

For the first part, he started by classifying the growth in a language as changing the vocabulary or changing the rules that say what a string of words means. Then, he suggested two ways of actually doing the growth: either “one person (or a small group) to be in charge and to take in, test, judge, and add the work done by other persons”, or “to just put all the source code out there, once things start to work, and let each person do as he wills” and compared the pros and cons of each methodology. He gave an example of a language that utilized both ways, which was *Linux*. I realized that this is the reason why open-source software was so successful over the last decade.

Then he introduced something called “cathedral” and “bazaar”, which were actually analogies for designing with a “master plan” and designing with a flexible plan. Quoting from his words:

“…the key point is that in the bazaar style of building a program…, the plan can change in real time to meet the needs of those who work on it and use it.”

This was when I started to realize the importance of this course CS320, and the process that I was going through. Unlike the languages that I was familiar with such as C, Java, or Python, there will be a need of features that are scattered throughout different (already-built) languages. If that ever happens, then I would have to build my own language, and the principles that I will be sure to follow is the ones that he’ve mentioned in his lecture.

Summarizing above, he said that

“It is good to design a thing, but it can be far better (and far harder) to design a pattern. Best of all is to know when to use a pattern.”

Overall, this talk was very motivational in some sense. Guy L. Steele did a splendid job in explaining why we should be focused on “growable” language and how we should “grow” a language. Last but not least, I now realized exactly what I was learning in CS320, which is the one take-away that I think is the most valuable lesson learned from this lecture.