Short History of STL

Alexander Stepanov

(contributed to *Evolving a language in and for the real world: C++ 1991-2007* by Bjarne Stroustrup)

In October of 1976 I observed that a certain algorithm – parallel reduction – was associated with monoids: collections of elements with an associative operation. That observation led me to believe that it is possible to associate every useful algorithm with a mathematical theory and that such association allows for both widest possible use and meaningful taxonomy. As mathematicians learned to lift theorems into their most general settings, so I wanted to lift algorithms and data structures. One seldom needs to know the exact type of data on which an algorithm works since most algorithms work on many similar types. In order to write an algorithm one needs only to know the properties of operations on data. I call a collection of types with similar properties on which an algorithm makes sense the *underlying concept* of the algorithm. Also, in order to pick an efficient algorithm one needs to know the complexity of these operations. In other words, complexity is an essential part of the interface to a concept.

In the late 70's I became aware of John Backus's work on FP[1]. While his idea of programming with functional forms struck me as essential, I realized that his attempt to permanently fix the number of functional forms was fundamentally wrong. The number of functional forms – or as I call them now – generic algorithms is always growing as we discover new algorithms. In 1980 together with Dave Musser and Deepak Kapur I started working on a language Tecton to describe algorithms defined on algebraic theories. The language was functional since I did not realize at the time that memory and pointers were a fundamental part of programming. I also spent time studying Aristotle and his successors and that lead me to a better understanding of fundamental operations on objects like equality and copying and the relation between whole and part.

In 1984 I started collaborating with Aaron Kershenbaum who was an expert on graph algorithms. He was able to convince me to take arrays seriously. I viewed sequences as recursively definable since it was commonly perceived to be the "theoretically sound" approach. Aaron showed me that many fundamental algorithms depended on random access. We produced a large set of components in Scheme and were able to implement generically some complicated graph algorithms.

The Scheme work led to a grant to produce a generic library in Ada. Dave Musser and I produced a generic library that dealt with linked structures. My attempts to implement algorithms that work on any sequential structure (both lists and arrays) failed because of

[1] John Backus, Can programming be liberated from the von Neumann style?: a functional style and its algebra of programs. *Communications ACM* 21, 8 (Aug. 1978), 613-641

the state of Ada compilers at the time. I had equivalents to many STL algorithms, but could not compile them. Based on this work, Dave Musser and I published a paper where we introduced the notion of generic programming insisting on deriving abstraction from useful efficient algorithms. The most important thing I learned from Ada was the value of static typing as a design tool. Bjarne Stroustrup had learned the same lesson from Simula.

In 1987 at Bell Labs Andy Koenig taught me the semantics of **C**. The abstract machine behind **C** was a revelation. I also read lots of UNIX and Plan 9 code: Ken Thompson's and Rob Pike's programming style certainly influenced STL. In any case, in 1987 **C++** was not ready for STL and I had to move on.

At that time I discovered the works of Euler and my perception of the nature of mathematics underwent a dramatic transformation. I was de-Bourbakized, stopped believing in sets, and was expelled from the Cantorian paradise. I still believe in abstraction, but now I know that one ends with abstraction, not starts with it. I learned that one has to adapt abstractions to reality and not the other way around. Mathematics stopped being a science of theories but re-appeared to me as a science of numbers and shapes.

In 1993, after 5 years working on unrelated projects, I returned to generic programming. Andy Koenig suggested that I write a proposal for including my library into the **C++** standard, Bjarne Stroustrup enthusiastically endorsed the proposal and in less than a year STL was accepted into the standard. STL is the result of 20 years of thinking but of less than 2 years of funding.

STL is only a limited success. While it became a widely used library, its central intuition did not get across. People confuse generic programming with using (and abusing) **C++** templates. Generic programming is about abstracting and classifying algorithms and data structures. It gets its inspiration from Knuth and not from type theory. Its goal is the incremental construction of systematic catalogs of useful, efficient and abstract algorithms and data structures. Such an undertaking is still a dream.

You can find references to the work leading to STL at: www.stepanovpapers.com