Project proposal: the Copenhagen STL

Institution:	Department of Computing, University of Copenhagen (DIKU)
Proposed duration:	1.1.2001 – 31.12.2003
Team leader:	Jyrki Katajainen
Backup person:	Lars Yde
Other programmers:	Christian Boesgaard
	Jesper Dangaard Brouer
	Jacob Gorm Hansen
	Allan Beaufour Larsen
	Bjarke Buur Mortensen
	Christian Worm Mortensen
	Mads Nielsen
Industrial partners:	Jesper Bojesen

Summary. The Standard Template Library (STL) is a library of generic algorithms and data structures that is part of the ISO standard for the C++ programming language. Although existing implementations are efficient, many STL components can still be improved. We propose the design and development of an enhanced edition of the STL which will bring that component library closer to its optimum performance. To enable this, we will first ascertain what that optimum is by carrying out detailed analysis of the STL components and in the process develop tools for automated analysis. We also plan to experiment with novel software for project monitoring during development and thus help document and track our efforts.

Inspiration. "I would like to see components become a dignified branch of software engineering. I would like to see standard catalogues of routines, classified by precision, robustness, time-space performance, size limits and binding time of parameters. ... I do not want the routine to be inherently inefficient due to being expressed in machine independent terms. ... I think there are considerable areas of software ready, if not overdue, for this approach."
—Excerpt from "Mass produced software components", M. Douglas McIlroy, 1968

Background

The need for standardized, well researched, and well documented software libraries was recognized decades ago as the above quotation illustrates. With the increasing complexity of software, such libraries have gained in importance and now feature in most major programming languages. The Standard Template Library (STL), which is part of the ISO standard for C++ [5], is one example of a software library that has found wide acceptance and appli-

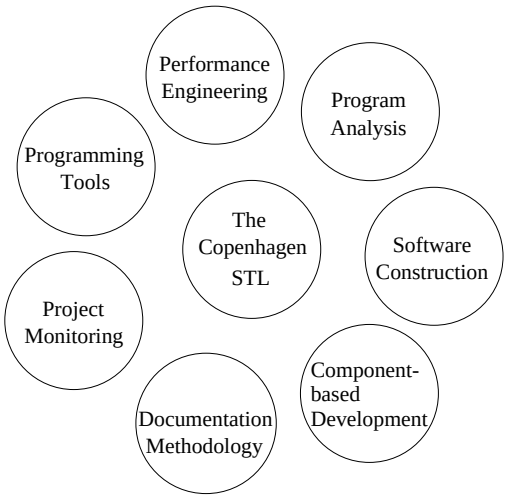

Figure 1. The Copenhagen STL and its satellite areas.

cation. The key reasons for that are its genericity, i.e. broad applicability, its clean, well documented design, and its comparatively good performance. The last item is of particular interest to us as we, under the leadership of Jyrki Katajainen, have accumulated significant experience and skill in the area of performance engineering and have established a laboratory for theoretical and applied research in this field [4]. We believe this puts us in a unique position to further develop and enhance the STL components that are now in use by thousands of developers around the world. Our primary motivations for doing so are:

- A desire to make the STL even more efficient than the implementations available at present and thereby promote library-based software development.
- A desire to complement and enhance existing documentation so as to give developers better grounds for assessing and using STL components.
- A desire to develop and experiment with novel tools and methodologies for program analysis and project monitoring.
- A desire to demonstrate the usefulness of performance engineering practices in an area of great potential benefit to both industry and academia.

The Copenhagen STL

We propose the development of an enhanced edition of the Standard Template Library [5], from here on referred to as *the Copenhagen STL*. The Copenhagen STL will be the centre-piece of our research, but should not be seen in isolation. We have identified several satellite areas of relevance to the core project (illustrated in Figure 1), some of which are already being

explored by members and associates of our research team, and we expect to be active in all of them.

Specifically, we see the following as the most compelling arguments for making our project a reality:

- We will construct a tuned component library for use by industry and academic developers worldwide.
- We will carry out a rigorous analysis of the space and time complexity, i.e. potential resource consumption, of each STL component before and after being performance engineered. We consider this a significant activity in its own right since the C++ standard [5] defines often only asymptotic, i.e. rough, complexity requirements for the number of instructions executed, which may be unsatisfactory for its potential users who are likely to place a premium on run-time performance.
- We will carefully document each performance engineered component, stating the findings of the complexity analysis as well as the modifications and improvements made during design and implementation.
- We will write articles detailing the analysis, design, and implementation of the resulting programs and tools. Given the wide range of disciplines involved in the project work, we foresee interest in a variety of computer science communities and hence good prospects for publication in prestigious scientific journals.
- We expect to gain extensive development and analysis experience which is likely to benefit the project collaborators in their future work and to spawn new initiatives in academic or industrial settings.
- We will design and develop software tools for program analysis that are likely to find application beyond this project.
- We will help promote component-based software development by demonstrating that the performance of generic components can rival — and surpass — that of carefully hand-crafted software.
- We will experiment with novel software for project monitoring currently under development by a member of our team and thus hopefully gain experience of value to the software engineering community.

The expected gains are thus manifold and realizing them will involve a diverse team of collaborators including student programmers, M.Sc. and Ph.D. students as well as the scientific personnel already in or associated with the performance engineering laboratory at DIKU.

Existing research

Our project will draw inspiration from existing research for both its subject matter and methodology. The former by building on the existing STL implementations as documented in [1, 6, 9, 10, 11] and by applying the research and experience documented, for example, in [2] and [3]. The latter by emulating aspects of the design and documentation of the acclaimed LEDA

[8] — an existing component library that maintains high standards we hope to attain to in our efforts.

A concrete example of how we intend to accomplish that can be found in [3], where implementations of various heap-construction algorithms are compared to performance-engineered versions which are shown to out-perform their original counterparts by a significant factor. The results derived are directly applicable to the STL which contains a handful of heap algorithms and therefore give a fine example of the goals we wish to pursue in our work.

Research distribution

We will partition and distribute the research according to the background, experience and available time of the project collaborators. The principal investigator, Jyrki Katajainen, will take as active a part as possible in the research process itself, but his responsibilities in managing this project and teaching at DIKU will most likely put severe restrictions on his participation. We therefore ask funding for programmer assistance to help implement the Copenhagen STL designs and we also ask for minor sums to help cover travel and hosting expenses, as detailed in the budget below.

We have launched the project this autumn as a graduate course at DIKU where we expect to lay the foundations for the Copenhagen STL with the assistance of volunteer graduate students who receive course credit for their efforts and thus require no financial compensation. Furthermore, we have strong reason to believe that a number of graduate students will be interested in writing master theses pertaining to the Copenhagen STL or its satellite projects and thus boost the research efforts significantly.

Management and organization

We believe organizational style should be dictated by project needs, not vice versa, and since the Copenhagen STL is a hybrid in project terms, being based on existing technology yet exploring fresh territory, we intend to organize it accordingly. This means that we will establish a framework for the Copenhagen STL, stating principles for design and development and setting up guidelines for research but at the same time leaving room to maneuver. We will try to engender an atmosphere of inquisitiveness and openness among the project collaborators, while at the same time ensuring progress through an evolutionary delivery policy [7], i.e. by developing the Copenhagen STL incrementally. This approach will, we believe, fuel a virtuous cycle of delivery, feedback and research that will result in an end-product incorporating both the ideals formulated at the outset and the discoveries made under way.

Research budget for 3 years

Designation	Combined expenditure
1 part-time programmer position for 3 years	410.000
Total salary expenditure for 3 years	410.000
Literature	10.000
Hardware	40.000
Travel expenses	50.000
Guests	20.000
Total operational expenses for 3 years	120.000
Salaries + operational expenses	530.000
Administration and overhead (20% of 530.000)	106.000
Total project expenditure	636.000

Note that the programmer position is not intended for uninterrupted employment of the same individual for a three year period. Rather, it will be filled by senior graduate students contractually employed for a period of three to six months which will hopefully stimulate the research environment as well as expose students nearing the end of their studies to project work similar to what they will encounter in their future work as professionals.

References

[1] M. H. Austern, *Generic Programming and the STL: Using and Extending the C++ Standard Template Library*, Addison Wesley Longman, Inc., Reading (1999).

[2] J. Bojesen and J. Katajainen, Interchanging two segments of an array in a hierarchical memory system, *Proceedings of the 4th International Workshop on Algorithm Engineering*, *Lecture Notes in Computer Science* **1982**, Springer-Verlag, Berlin/ Heidelberg (2000), 159–170.

[3] J. Bojesen, J. Katajainen, and M. Spork, Performance engineering case study: heap construction, *The ACM Journal of Experimental Algorithmics* (to appear).

[4] Department of Computing, University of Copenhagen, Performance engineering laboratory, Website accessible at `http://www.diku.dk/research-groups/performance-engineering/` (2000).

[5] International Organization for Standardization (ISO), *ISO/IEC 14882: Standard for the C++ Programming Language*, Genevé (1998).

[6] N. M. Josuttis, *The C++ Standard Library: A Tutorial and Reference*, Addison Wesley Longman, Inc., Reading (1999).

[7] S. McConnell, *Code Complete: A Practical Handbook of Software Costruction*, Microsoft Press, Redmond (1993).

[8] K. Mehlhorn and S. Näher, *LEDA: A Platform for Combinatorial and Geometric Computing*, Cambridge University Press, Cambridge (2000).

[9] D. R. Musser and A. Saini, *STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library*, Addison-Wesley, Reading (1996).

[10] Silicon Graphics Computer Systems, Inc., Standard template library programmer's guide, Worldwide Web Document (1999). Available at `http://www.sgi.com/Technology/STL/`.

[11] L. Yde, Introduction to the STL — with applications, Worldwide Web Document (1999). Available at `http://www.diku.dk/research-groups/performance-engineering/resources.html`.