CHAPTER 12

TERRY SHINN

WHEN IS SIMULATION A RESEARCH TECHNOLOGY? PRACTICES, MARKETS, AND LINGUA FRANCA

The practices of simulation are highly diverse: In a given domain, and for a single problem, practices are frequently multiple, different from one another, divergent, and more than occasionally contradictory. In view of such acute plurality, how is one to grasp the sense of 'simulation'? While the number and scope of practices embedded in simulation and the magnitude and heterogeneity of the simulation market are synonymous with fragmentation, does fragmentation necessarily prescribe the operation of the simulation community, and if not, what might be the form and function of a said community? Is it reasonable to speak in terms of simulation as a system, and if so, on what grounds? This chapter explores this and related issues. It examines transverse features of simulation that serve as operators of cohesion, which cohesion constitutes a prerequisite for the stabilization of a social/cognitive system. The canvas presented here will include historical, organizational, professional, and epistemological components. The context of the emergence of the C++ general-purpose, multiparadigm, object-oriented simulation language will be explored. It will be suggested that the intellectual and social dynamics of C++ strongly reflect key features of generic instrumentation and research technologies, and that, by virtue of this correspondence, it is reasonable to think of the practices, structures, and market of simulation in terms of a transverse research technology system.

FOUNDATIONS

Many of the cognitive, organizational, and institutional elements constitutive of contemporary simulation were introduced before 1960. With a few notable exceptions, such as visualization techniques and virtual reality, what has happened since is an extension of that early orientation. Recent change has largely occurred within the confines of the historical mold and logic that initially formed today's huge and diverse domain of simulation thought and action.

Questions of simulation emerged for the first time in Germany in the 1920s when H. Roeder took out a patent for devices intended for use in training pilots of submers-

J. Lenhard, G. Küppers, and T. Shinn (eds.), Simulation: Pragmatic Construction of Reality, 187–203.

ibles, balloons, and airplanes. The simulators were designed to represent changes of altitude in three planes of movement, to register commands initiated by pilot trainees, and to readjust altitude correspondingly. The project came to naught, but the effort is important. It connects simulation to the emergence, development, and currency of new forms of technological artifacts. It is an extension of a historically novel form of technical design, materials, and activities. From the outset, simulation was connected to aviation, issues of complex motion, equations that describe complex flows and interactions, and the extension and adoption of new varieties of skills (sometimes entailing new training programs). Finally, the simulation trainers project of the 1920s established important parameters that underpin thinking and action in most simulation ventures: 1) representations of multidimensional environments, 2) interaction between elements figuring in the representation (sometimes including human operators), 3) emphasis on TIME, that often comprises a key dimension beyond the three axes of freedom in space (particularly in virtual reality), 4) testing, and 5) validation.

However, it was not until the 1940s that Roeder's intuition that the components of simulation, aviation, and training comprise an integrated unit became a reality, and when it occurred, it was not in Germany but instead in the United Kingdom and United States. This gap corresponds to a massive growth of a simulation market in the shape of war-driven demand for expanding numbers of increasingly advanced combat aircraft, for attendant pilots, and for quick efficient training. It similarly corresponds to the design, construction, and spread of simulation-relevant technology such as 'fast' analogue calculating devices, capable of coping with elementary fluid flow equations and their translation into simulation dynamics and mechanical outputs adapted to an aircraft control environment. This evolution proved crucial: Today's faster, better, and generalized simulations rely entirely on digitalized calculations! The historical multifront technological advances of the 1930s and 1940s in electronics and calculation provided the mental and material conditions fundamental to simulation. The centrality of technology to simulation cannot be overestimated. Simulation is the technology of technology, of science, and of industrial operations and beyond. In its role as the technology of technologies, simulation represents the most reflexive form of analysis/action yet known to humanity, and broadly practiced in society.

In 1939, Professor L. Mueller, working at MIT, designed and built a fast analogue computer to study the longitudinal dynamics of aircraft motion. Mueller's interest was in the solution of the set of aerodynamic equations and their simulation for design purposes. However, in a postscript to his paper, he mentioned that his simulator could be adapted to flight simulation pilot training. In 1941, an electronic analogue computer was developed at the TRE unit for the radar training program. The TRE group, originated at MIT during the war, combined advanced detection, electronics, and control systems coupled to fast calculators, mainly for military objectives. This device was based on the ideas of F. Williams, famous for his later contributions to digital computers, and used the velodyne – another TRE invention for integration. The first model of this computer had been developed by Dynatron Radio Ltd. in 1941.

The war years saw an increase in the companies involved in simulation as well as in hardware, accessory, and modeling technology. In 1945, a new system was introduced

by K.M. Uttle that incorporated force fields and visual inputs into flight trainers. In Britain, advanced longitudinal dynamics was added to the flight simulation repertory through the efforts of G.M. Hellings and the resulting electromechanical analogue computer. This computer and model was arguably sufficiently general and flexible to correspond to the flight characteristics of any aircraft then in operation. The simulator boasted a pitch motion system that incorporated an endless moving belt. In the United States, the Special Devices Division of the Bureau of Aeronautics developed the Center for Naval Training Equipment. It supported the simulator activities of Bell Laboratories, which constructed the Navy's PBM simulator that included a complete front fuselage, cockpit, accessories, and all instrumentation. The Link trainer was also developed during this period. In 1948, and during part of the 1950s, the Curtiss-Wright aviation firm engaged in simulator design, The company produced a new line of servo devices and what was known as shadow graphics. General Electric entered the simulation race during the 1960s, developing digital systems for space-related operations.

Work surrounding the atomic bomb constitutes a second current of simulation activity. During the latter phase of bomb research, Los Alamos scientists and engineers set out to study the magnitude of their bomb's explosive impact. At the time, nothing was known about nuclear blast extent. To determine this, scientists selected numerous possibly relevant parameters, and assigned a huge variety of values to each. The number of permutations was astronomic. They engaged the newly emergent computational technology of computers becoming available at that time to calculate the likelihood of the selected parameters and to estimate the consequences of each. The resulting calculations indicated a statistical likelihood of bomb effects. The probability-based technique of this project soon acquired the name *Monte Carlo simulation*, presumably suggesting the probabilistic aspects of operations. It rapidly became central to a sweep of simulation endeavors. On a different register, starting in the 1950s with scenarios based on 'if-then' logic, insurance companies used Monte Carlo simulation to work out actuarials, and soon banks and investment firms were using it for investment and client advice. Monte Carlo simulation has similarly become the cornerstone for much risk assessment research and public policy, and it is today a technique deployed by nuclear energy lobbies seeking to quiet public unease about nuclear hazard. The introduction of high-power individual computer technology has even further accelerated the generalization of this form of simulation (e.g., for calculating possible trends in the stock market).

The initial organization of simulation technology was rooted in engineering practice and undertaken by engineers and not by scientists, scientific societies, or universities. One of the first simulation forums to be scheduled and the first simulation organization founded was venued in Europe, and not in the United States, where much of the early simulation work had been carried out. One explanation for this is that part of European postwar reindustrialization was free to build around new technology, the older prewar production capacity and technology having been destroyed by bombing, battle, and sabotage. In 1955, a meeting was convened in Brussels at the Free University attended by researchers, managers, and observers of the simulation laboratories that existed at the time. Many of the laboratories were a by-product of the enormous simulation-related activities of World War II. Participation was

international, with delegates coming from most west European countries, the United States, and Japan. The conference decided there existed a need for a permanent means of communication between members of the emerging simulation community. Participants perceived that extant professional and scientific bodies were too restrictive in composition and outlook to include the diversity of backgrounds, skills, and interests that characterized the multidisciplinary, multisectoral, and multipractice new world of simulation. The result was the creation of the AICA (*L'Association Internationale pour la Computation Analogue*). Under the influence of technological and scientific innovation, the initial scope of the AICA expanded to include more mathematical analysis (particularly numerical operations), mathematical modeling, and digital technology. This body played a crucial role in the introduction of simulation particularly into Europe. Its outstanding successes lay in the domains of chemical engineering, automatic system engineering, and later simulation-based design, specifically in the realm of mechanics. In 1976, the pioneering body took the name *International Association for Mathematics and Computation for Simulation*, to better reflect the broadening technology, uses, and markets of simulation. The original AICA was significant for the establishment of simulation, because it tried to coordinate and combine the analytic practices developing in simulation, which, at the time, was still an outside cognitive corpus and set of practices as well as being socially nebulous.

SIMULATION AT WORK – POST 1960

Most post-1960s simulation activity has focused on engineering-related and industry/service-oriented work, as measured by the focus of simulation societies and journals. John McLeod and Vincent Amico have been pillars of simulation in the United States since World War II. McLeod earned a BS in engineering at Tulane University, and has been associated with the universities of Chicago, Harvard, and MIT. He is an expert in the design and construction of automatic control systems (boasting two patents) in which simulation is the principal tool. He served in the US Navy's Guidance Systems Simulation Laboratory for a decade, acquiring initial simulation-based design experience there. He then went to work for the Northrop Company, moving to General Dynamics between 1956 and 1963 – again in design. McLeod became an independent simulation researcher for a brief period during the 1960s when he designed a heart-lung machine using principles of simulation-driven design techniques. He was an active consultant throughout his career.

Amico earned a BS in engineering from New York University in 1941 and went on to study physics. From 1941 to 1945, he worked on the structural design of missiles and aircraft at the Static Test Laboratory at Wright Field. After 1948, he worked as a civilian for the US Navy as product engineer for flight training equipment. In 1969, Amico became Research Director of engineering of his design unit; and, in 1979, he was appointed Research Director of the Navy's entire flight training program. Throughout, Amico based his endeavors on simulation development and simulators. They comprised the mainstay of his career. Beginning in 1972, he taught at the computer department of the University of Southern California, specializing in simulation techniques.

McLeod and Amico figured importantly in the development and successes of the Society for Computer Simulation that became the United States major organization specializing in simulation publications, and to a lesser degree simulation conferences. Although the Society was set up in 1952, it remained a relatively small and obscure body until the late 1960s, when McLeod helped expand its activities, often assisted by Amico. McLeod turned this American organization into an international body; and under his careful guidance, the Society was rechristened the *Society for Modeling and Simulation International*. McLeod managed to connect the American simulation community to a broader simulation environment based in Great Britain, France, Holland, Germany, Italy, Denmark China, Korea, and Japan. From the 1970s onward, simulator experts and users regularly participated in the annual meetings of the Society for Modeling and Simulation International – the *Summer Computer Simulation Conference* that specializes in continuous event simulation.

The Society for Modeling and Simulation operates a large, prestigious stable of simulation publications. It publishes *Simulation* – the society's flagship review, and the most well-read and respected journal in the field. It also publishes *Transactions for Modeling and Simulation*, *Simulation Magazine*, and *The Journal of Defence Modeling and Simulation: Applications, Methodology, and Technology* (with the US Army and Simulation Office).

However, it is not the Summer Computer Simulation Conference that constitutes the foremost venue for the exposition of new work in the field of simulation, but instead the *Winter Simulation Conference*. Whereas the beginnings of the Winter Simulation Conference date back indirectly to a series of small simulation seminars held during the late 1940s, the organization was set up only in 1967. The initial meeting was headed by H.J. Hixson (head of operations systems research analysis with the US Air Force logistics command and program director of the IBM SHARE users group) and by J. Reipman (prominent user of the general purpose simulation system approach in the Nordon division of the United Aircraft Corporation and a leader in the IEEE). It differs from the Summer Computer Simulation Conference in several important respects. The latter deals with continuous simulation, whereas the Winter Conference specializes in discrete event simulation. Discrete event simulation efficiently represents events in which time is a subordinate consideration. In contrast, continuous simulation, based on the solution of differential equations, unceasingly monitors time, but is less attentive to details of complex events. Continuous simulation is used to model activities like continuous flow engineering and aircraft automatic pilot systems. Manufacturing and services are modeled with discrete event simulations. However, as indicated by Küppers and Lenhard (this volume), a closer analysis reveals that matters are becoming increasingly complicated, because even continuous simulation relies on discrete models.

Whereas the Summer Conference has rejected sponsorship by professional engineering bodies, interest groups, and public agencies, the Winter Simulation Conference has multiplied such connections. The Association for Computing Machines, the IEEE, and IBM sponsored the initial meeting. An audience of 225 was expected, but interest ran so high that 401 attended. The proceedings were published by the *IEEE Transaction* in a special issue in 1968 on systems science and cybernetics.

The second Winter Simulation Conference took place in December 1968: Its theme was *Simulation Applications*. In addition to the initial sponsors, this meeting also received backing from Simulations Council Inc. There were twenty-two sessions, and eighty-eight papers were presented on a range of simulation applications. Statistical research, simulation computer language development, and simulation education were included in the program. Attendance jumped to 856. A 356-page digest of conference papers was subsequently published. The 1969 Winter Simulation Conference was sponsored by the American Institute of Industrial Engineers and the Institute of Management Sciences/College on Simulation and Gaming. In 1971, the Operations Research Society of America also became a sponsor. That year, attendance reached over 1,200 – the highest figure ever.

In 1974, the tide turned: The number of participants fell sharply, as did financial support for the program. The future looked bleak for several years, until the National Bureau of Standards intervened, infusing organizational vigor, new ideas and projects, and fresh money. In the 1980s, the Winter Simulation Conference regained its former ascendancy and has maintained it ever since. But, why the collapse in the mid-1970s, and what structural considerations contributed to its newfound energy?

Although the exact circumstances require further research, one can point to several contributory factors: By the 1970s, there existed a plethora of simulation research directions, projects, application niches, and implementations. The field had fragmented considerably. Functional sectors and individual firms were working out their specific simulation solutions. The initial flush of enthusiasm that fuels a new venture had begun to erode. Two key initiatives regalvanized the simulation venture. Intervention by the US National Bureau of Standards provided a measure of stability and coordination that was otherwise lacking. The Bureau pulled together divergent simulation movements. Second, new initiatives in programming language began to emerge. Introduction of a language like C++ (to be analyzed in detail below) helped the simulation community by providing a focus of technological, intellectual, and professional convergence.

In sum and as indicated above, military-related programs lay at the center of the development of simulation work. This was facilitated by the introduction of fast digital computing power and by the swift broad spread of computers. Nevertheless, simulation efforts had begun to thrive on the basis of predigital computation. Slow analogue devices had already permitted simulation to successfully invade a growing range of military-related realms before advanced digital developments. Additionally, the vast majority of simulation endeavors occurred in the narrow engineering/technology sphere.

STRUCTURING SIMULATION – THE BIRTH, EVOLUTION, AND ROLE OF THE C++ GENERAL-PURPOSE, MULTI PARADIGM, OBJECT-ORIENTED PROGRAMMING LANGUAGE

The pages that follow will document the centrality of the C++ computer language to the internal development and point to the diversification and growth of the simulation markets and community since the mid-1980s. It will be further suggested that C++ exhibits many of the key attributes of research technologies. Grounded on these twin

observations, I argue that simulation itself may usefully be perceived as a research technology rich in generic instrumentation that simultaneously provides a stable kernel to the simulation domain and permits diversity, yet transversality, commensuration, coherence, and cohesion.

The C++ simulation-directed, general-purpose, object-oriented, multipurpose language was developed between 1983 and 1985. The programming language's author is Bjarne Stroustrup (1950), a brilliant, energetic, successful, and some would say charismatic, general-purpose language developer. Today, C++ is the most widespread language in simulation. In 2003, estimates range between 1.3 and 3 million users. C++ is a generic technology, fathered, matured, and organized in an interstitial environment. The disembedding of its generic features and their reembedding in specific applications involves intermittent selective boundary crossings. It corresponds to a form of metrology. By virtue of this combination of characteristics, C++ constitutes a research technology (Joerges and Shinn 2001; Shinn and Joerges 2002; Shinn and Ragouet 2005).

Bjarne Stroustrup is currently the College of Engineering Professor at the Texas A&M University Department of Computer Science, and director of the Large-Scale Programming Research Department at AT&T Laboratories. He was born in Denmark, and did his undergraduate work at Aarhus University, taking a degree in mathematics and computer science, before moving to Cambridge University for his doctoral studies. His dissertation adviser was David Wheeler, a well-known programmer who contributed to the Illiac. The Illiac, built in 1952, based on von Neumann architecture and located at the University of Illinois, was the United States' most powerful university computer, even surpassing the capacity of the combined Bell Laboratory machines.

Stroustrup studied at the Cambridge Computer Laboratory, conducting research on alternatives for the system software of distributed systems. He composed new software from existing systems and tested feasibility and efficiency using simulation techniques. On completing his doctorate in 1979, Stroustrup took a position in the computer science research center in Bell Laboratories at Murray Hill, New Jersey, where he undertook research alongside language specialists like Denis Ritchi, who had recently developed the programming language C. From 1979 to 1983, when Stroustrup set out to build a new language, he was involved in a range of Bell-related tasks. He was notably active in simulation research intended to improve distributed network system operations, and explored applications of this approach. In 1979, Stroustrup set out to analyze the Unix kernel to determine how it could be distributed over a network of computers connected by a local area network. He also worked on improving the low-level language C (Stroustrup 1993).

One way of describing C++ is that it contains many elements of C that have been enriched with Simula and an object-oriented perspective. Stroustrup often says that C++ is three languages in one: a C-like language (supporting low-level programming), an Ada-like language (supporting abstract data-type techniques), and a Simula-like language (supporting object-oriented programming) (Stroustrup 1994: 198). C++ is also organically connected to additional languages (Stroustrup 1994: 198). Algora68 gives to C++ operator overloading and the capacity to declare variables anywhere in a block. BCPL allows comments. Simula gives organization.

Whereas one strength of C is its proximity to computational machines, via the introduction of elements from Simula and from the object-oriented perspective, C++ connects directly to material problems by analysis of language application. C's logic connects with computational machinery, whereas C++'s logic retains this property and adds the property of smooth problem application logic.

Stroustrup's design emphasized three stable features – application, the generic concept of classes, and portability (the latter allowing cross-boundary flexibility and translanguage communication). Stroustrup has always been concerned with promoting solutions to real problems (Stroustrup 1997a,b). While his Denmark training in mathematics was interesting and stimulating, it nevertheless left him uncomfortable, as he is committed to confronting problems. He often insists that his language insights and successes result from thinking about programming with reference to personal problem-solving experience. He generalizes up from problems. C++ has been built to enable real and diverse users to better grasp their problems and treat them computationally. In this sense, the C++ language is application- and user-driven.

Second, C++ was designed to operate in a framework of classes. Initially, C with classes was developed by Stroustrup to allow simulators to be built for research in network design being carried out by Sandy Fraser at Bell. Inclusion of the term 'C' in the name C++ indicates the extent of C's parentage to C++. One frequently asked question is why C++ did not simply emerge as an evolution of C, rather than distinct from it, and as a powerful and eventually victorious competitor. Part of the answer is the centrality of classes in C++. While Stroustrup seriously tried to reconcile the importation of classes into C, as witnessed by his construction of C with classes, the architecture of C limits the full expression of classes. This drove Stroustrup to further diverge from C, as he continued in his project to build a more useful programming language. The relationship between C and C++ is expressed in a phrase submitted by Stroustrup in a 1989 article *As close to C as possible, but no closer* (Koenig and Stroustrup 1989).

According to Stroustrup, classes possess a multifold advantage (Venners 2003; Dolya 2003). Classes promote reasoning in terms of connections crucial to object-oriented representations and work. C++ is not an object-based programming language; it is an object-oriented code, which may be more restrictive. Classes help identify the similarities shared by elements. They facilitate computation between them. They furthermore allow passage from one part of a program to another with a minimum of difficulty, facilitating the work of programmers and users. Classes in a program reduce runtime, making computation efficient. In C++, the combination of classes and static checking helps alleviate the need for garbage collection, which constitutes an important economy in runtime and memory. Stroustrup also suggests that elegance is a desirable quality of a good language, and the use of classes promotes elegance. But, above all, in C++, the true purpose of classes is that they provide a platform for clear reasoning about complex structures.

Classes, in conjunction with other components like static checking, restrictive garbage collection, and multiple inheritance, constitute the generic feature of C++. The centrality of classes in C++, along with the role played by the object-oriented perspective, makes C++ ubiquitous in much simulation work.

While classes comprise the generic anchor of C++ that allows the general purpose code to be adapted and adopted by multiple applications, portability represents the format through which classes and other generic features of the language are vehicled. Portability was crucial to Stroustrup from the outset. He tried to design a language far more portable than C. Portability infers compatibility and flexibility. C++ is not a restrictive language, as it can easily be connected to a range of hardware and software. It operates in innumerable environments – Unix, Windows, and Macintosh. C++ links conveniently to many application tools. It is thus embedded in a huge range of informatic products without restriction, whereas the architecture of other languages limits their partnering. C++ functions as a foundational language tool inside an ever extending variety of application-specific local tools. Most computer languages are written either by language designers for other designers or by users for a particular application; but this is not the case for C++, whose architecture integrates a breadth of perspectives and conveys a multitude of mechanisms felicitous to mobility. This drive for breadth echoes Stroustrup's twin concerns for and experiences in concrete application practice and design involvement.

Portability is the hallmark of boundary crossing. This feature permits the expression of classes in terms of genericity and class reembedding in diverse applications. Without C++'s portability, movement across boundaries would be rare or impossible. Portability spells ongoing communication between evolving C++ design and otherwise isolated C++ niche users. Thanks to portability, C++ can thus stand as a generic language, as a language of application, and also as a reflexive transverse language that permits transapplication exchange.

The break of C++ with C, and its subsequent promotion in 1984–85, proved both problematic and easy. Its possible competitors, Modula-2 and Ada among others, were often regarded as restrictive, entailing awkward problems, or simply did not find dynamic outlets on the United States market. The Bell Laboratory, where Stroustrup worked, had experimented with and contributed to numerous languages, and was thus not irreconcilably committed to any particular one. The research unit was big, leaving room for individual initiatives and maneuver. Furthermore, Dennis Ritchi, who also worked at Bell and, along with Kristen Nigaart, is one of the pioneers of C, never strongly opposed Stroustrup and his endeavors, which increasingly distanced C++ from C. The rapid successes of C++ also owes much to the expansion of the mini- and microcomputer market and to the growth in the number and range of applications. Foremost in these applications was simulation, to which C++ was perceived as appropriate and congenial. The diffusion of C++ was also connected to keen interest in the new language among language designers and programmers. C++ required good compilers and libraries to ensure its spread and effectiveness in different applications (Venners 2003), and much to even Stroustrup's surprise, the computation community responded in record time to his proposed architecture with a number of world-class compilers.

The number of C++ users rocketed. In 1984–85, the emergent language was largely restricted to the Bell Laboratory. In the months that followed, C++ was distributed in a preliminary version to selected universities and a few users (Stanford, University of California, Cal Tech, University of Wisconsin, MIT, Carnegie Mellon, University of Copenhagen, Rutherford Laboratory in Oxford, etc.). The response was

not what was expected. Rather than expanding, demand stagnated. The motive for this was unanticipated. Users loved the new language and wanted nothing better than to use it extensively: but its use in consulting and in public applications required the stabilization and standardization of the language, its public recognition, and the development of adequate compiler architecture. Soon, however, the quick involvement of fresh C++ work, often originating outside of Bell Laboratories, allowed both the size and diversity of the C++ community to expand on an unforeseen scale.

Table 1. Growth in the size of the C++ community

Date	Estimated number of C++ users
1979	1
1980	16
1981	38
1982	85
1983	87
1984	135
1985	500
1986	2,000
1987	4,000
1988	15,000
1989	50,000
1990	150,000
1991	400,000
2002	1.300,000

The following ventures indicate the range of activities associated with C++ in the decade since 1985: Animation, autonomous submersibles, billing systems, bowling alley control, circuit routing (telecom), CAD/CAM, chemical engineering process simulations, compilers, control panel software, cyclotron simulation and data processing, database systems, decision support systems, digital photography processing, digital signal processing, electronic mail, expert systems, factory automation, financial reporting, flight mission telemetry, foreign exchange dealing (banking), search software, hardware description, hospital records management, industrial robot control, instruction set simulation, interactive multimedia, magneto hydrodynamics, medical imaging, missile guidance, mortgage company management, network management and maintenance systems (telecom), network monitoring (telecom), operating systems (real-time, distributed, workstation, mainframe, ‘fully object-oriented’),

programming environments, superannuation, insurance, shock-wave physics simulation, SLR camera software, switching software, test tools, transmissions systems (telecom), transport system fleet management, user interfaces, video games, and virtual reality (Stroustrup 1994: 172).

By the late 1980s and 1990s, C++ had become a major language among computer languages, or perhaps even the foremost. The *C++ Journal* appeared in 1991. *Computer Language*, *The Journal of Object-Oriented Programming*, *The C++ Users Journal*, *Journal of Object-Oriented Programming* (*JOOPS)*, and *Dr. Dobbs Journal* all ran regular articles on the C++ language. To this must be added the score of Internet language publications that frequently feature C++ and present recent C++-related tools and implementations.

C++ owes much to social factors. By the mid-1980s, the computer language and computer programming community had grown greatly. The consequence was twofold: First, numbers of young talented specialists could now loosen their ties with the big powerful computer firms that had earlier exercised considerable influence. Second, conditions permitted some individuals to become independent free-lance programmers. In 1986–87, a movement of independent compilers developed. This was opportune for C++, whose evolution and diffusion required expanding beyond the walls of Bell laboratory, and the participation of many people and inputs from many quarters.

The now famous Santa Fe compilers meeting was held in November 1987. This event marks a turning point for C++. Stroustrup anticipated an attendance of only a few dozen people in Santa Fe, but over 200 showed up! Papers were presented on C++ application, education, environment compatibility, and, strategically most crucial, on compiler development. Building effective C++ compilers was essential to users, as they ease the work of application. For a compiler to be in step with the C++ generic/reembedding/boundary-crossing Stroustrup precepts, it must support many other operating languages. This meeting initiated the design, construction, and diffusion of a spate of C++ inspired compilers and libraries – among others, the 1988 Zortec compiler and the 1990 Borlan compiler. In 1991, Windows marketed its C++ compiler; and in 1992, IBM came out with its version of a C++ compiler. A C++ groundswell ensued. In 1988, the NIH helped sponsor a C++ meeting and began to acquire C++ programs, tools, and other application implementations. In 1988, the second C++ conference was held In Denver, and, since then, there have been C++ conferences on an almost yearly basis.

During the 1980s, why did Bjarne Stroustrup push first to obtain the standardization of C++ by ANSI (the American National Standards Institute) and then by ISO a decade later (ISO/IEC14882)? What does standardization signify for a code, and what form of work is involved? How does standardization impact on a language? What has the standardization of C++ meant to simulation?

Before the standardization of C in the late 1980s, the language counted over 160 dialects. In this instance, standardization neutralized fragmentation and imposed order. The development of libraries and compilers can be used to hijack a language by locking in users. By standardization, a code becomes public and thus cannot be appropriated. Stroustrup deplores the idea of a proprietary language, and above all desired C++ to remain public. His design goal and subsequent strategy entailed that

C++ be an open pathway, not a closed system. He has often declared that C++ is for the average user as well as for the untypical user. It is for everyone! Finally, the documentation that necessarily accompanies standardization allows clarifications – clarifications in the work of design itself, design improvements, and clarification for users who want or need to know more about their code. Metrology thus fulfills the functions of stabilization, transparence, accessibility, and pedagogy.

In the early 1990s, C++ was certified by the *American National Standards Institute*, the *German Institute of Norms*, and by the *British Institute*. This achievement culminated five years of effort. Stroustrup was motivated by numerous considerations during his drive to win certification. Although the growing number of users approved and employed his language, general acceptance demands code homogenization and stabilization. This is necessary for a technology to be perceived as transparent, transferable, and reliable. Stroustrup understood this. By all means, code deviance must be avoided and prohibited. Lock in of parts of the language also had to be prevented! Standardization of C++ at ANSI required several years. ANSI operates in conjunction with code designers, subsequent contributors, tool and implementation designers, firms that develop and market the tools, attorneys of said firms, independent code experts, and users from many sectors. Observers and freelancers are also implicated in the standardization process. Deliberations occur on two levels: Technical committees deal with issues of detail. A general body discusses questions of principle, law, and policy. For C++, deliberations advanced relatively smoothly. At the end of the process, Stroustrup had given up nothing essential to his initial design plan.

The ANSI and ISO procedures affected several important evolutions in C++. Templates were made central to the code. They were connected to libraries, and the template standard (TSL) resulted. Multiple inheritance also became a key feature. Stroustrup had earlier been weary of inheritance, but in the form that it was engineered during standardization and combined with C++, he came to accept it and appreciate its power. The next steps in modifying C++, states Stroustrup, will be its extension to distributive programming that will necessarily introduce threads (Dolya 2003).

The upshot of standardization is that C++ became even more general-purpose/ multiparadigm than before. C++'s compatibility with other codes was enhanced. It is a general user's language, a specialty language, and a programmer's code. It remains a language for high-level designers, as its openness and generic quality continue to make it interesting, challenging, and a turf still sufficiently malleable and open to correctly support future evolutions.

THE INTERSTITIAL ENVIRONMENT

Genericity, reembedding, and boundary crossing are coupled to an interstitial environment, and this environment figures centrally in the development of C++ practices and the simulation community. An interstitial arena emerges in the interspecies between established dominant organizations, such as the university, corporations or small technology-based firms, state technical services, the military, and so forth. While individuals who occupy the interstitial environment may work for a dominant

organization, they nevertheless frequently escape organizational control by movement or by fostering arrangements that connect them to multiple organizations. Multiple connections increase resources and extend margins of maneuver. He who works for everyone is the bondsman of no one. In what ways has Bjarne Stroustrup and the design and evolution of C++ been affiliated with the interstitial environment?

Today, Stroustrup divides his time between contacts with C++ users and markets, code design, education, and acquiring new language skills and experience (Doyla 2003).

Stroustrup's experience is distributed, extending to pure and applied mathematics, language and program writing and use, and simulation. His endeavors bridge theory and application. Some of this output occurs in the framework of universities, state research or technical services (the NIH), private/semi-public services (ISO and ANTI), huge corporations (Bell and AT&T), and small technological companies (Silicongraphics). Concurrent and sequential employment is a rule. While once an employee of Bell Laboratories and still an active employee of AT&T Laboratory, Stroustrup has often simultaneously held other positions.

The interstitial environment respects and maintains divisions of labor inside science, inside technology, between science and technology, and between science and enterprise, while new important permutations are also invented continually (Joerges and Shinn 2001; Shinn and Joerges 2002; Shinn and Ragouet 2005). Stroustrup held to his C++ general-purpose, multiparadigm project. He used the relative autonomy provided by the interstitial environment to focus long-term attention on his objective. The interstitial arena is not an interest group, not a producer, and not a market. As often stressed by Stroustrup, his goal is steadfastly nonproprietary. He has no plan to encroach on the productions or operations of specific user/market niches. Consistent with this neutrality, the interstitial arena thus provides Stroustrup a platform for crossing the boundaries of innumerable organizations, markets, producers, and users, but without affecting the internal division of labor or infringing on their traditions, plans, or autonomy. He reembeds C++'s generic features into niche applications, and conversely uses the application experience of niches to design and enrich C++.

C++ comprises a lingua franca: A lingua franca is the result of genericity, reembedding, and boundary crossing. A generic technology like C++ contains one or several fundamental instrumentation features, for example, an emphasis on classes in combination with abstraction, object orientation, and portability. When the generic technology moves into a particular market niche, several things occur. The generic features of the technology are reembedded in the local technical culture. In the case at hand, applications absorb certain selected features of C++ in accordance with short-term demand. Parts of C++ are reshaped in this process of adoption. Nevertheless, C++'s adaptation does not alter the fact that the generic characteristics of the base technology survive intact. The stamp of the generic base is permanently imprinted on the local technical expression. The lingua franca arises out of this complex concatenation. For example, a specific library accompanies C++ into the separate applications of hydrodynamics research, auto piloting, and so forth. Nevertheless, the presence in C++ of a base code governing the organization of classes and logic of portability persists both in the piloting and science research extensions, and it is these stable, constant transverse features that enable users from both specialties to

communicate effectively about the language and about more substantive issues beyond their fields on a metalevel.

The reembedded generic technology vehicles a particular metrology in the form of standards or units of measurement, a specific vocabulary, form of imagery, methodology, or even a new paradigm. This residue is deposited in the local niche technology during reembedding. Although the product of reembeddings in diverse markets results in heterogeneous artifacts, the generic element remains uniform, a kind of technical fingerprint. However, C++ is expressed in the innumerable applications it serves; C++'s underlying signature persists. Through 'assimilating' the metrology, methods, vocabulary, or images of the hub generic instrument, niche practitioners come to share familiarity with, and competence in, a particular syntax and semantics. This common language becomes an integral feature of the practitioner's niche language. The language is associated with a set of local, efficient, robust practices. The technology, practices, and outcomes associated with the local techniques enjoy the status of 'truth' – in the sense of 'practical truth.' The lingua franca of C++ is the generic residue of the reembeddings of C++ in a multiplicity of fragmented market niches. It is that part of the C++ hub technology that transcends the transformations occurring during the process of adaptation and adoption. Since there persists a transverse stable kernel used in the discourse of market practices, when practitioners who come from diverse economic or disciplinary sectors and have different functions (and come from different nations and have even different cultural horizons) meet, they can nevertheless communicate with reference to a technical field and generate intelligibility. The common parlance available through this generic-driven lingua franca promotes cognitive, artifactual, and organizational transversality that somewhat neutralizes the otherwise often disruptive effects of today's rampant intellectual and social differentiation and fragmentation. Witness to the existence and efficiency of C++ as a lingua franca in simulation can be found in two venues. The annual Summer and Winter Simulation Conferences draw users from scores of applications. They communicate through C++ about C++, and also use the medium of C++ to communicate about their respective different and sometimes divergent simulation applications.

By virtue of the fact that C++ represents a kind of research technology connected to many and diverse audiences and functions and highly amenable to transversality in the form of boundary crossing and commensuration, and by virtue of the fact that C++ constitutes a dominant code in simulation practice, one may reasonably establish the operation of a link between simulation practices and markets on the one hand, and research technology on the other. Simulation's stability and strengths owes much to atributes drawn from research technology. Research technologies provide simulation (here demonstrated through analysis of C++) with open-ended techniques, representations, codes, and language that make it applicable in a miriad spheres, and it simultaneously offers a solid, self-referencing platform that gives definition, meaning, and direction to the more general, overall, quasi-universal simulation enterprise.

WHEN IS SIMULATION A RESEARCH TECHNOLOGY?

Simulation is a system whose architecture emphasizes three features: First, simulation involves a remarkably large number of markets, and the number continues to

expand. Second, the particularities and demands of simulation niches are diverse and even divergent. Many markets require specific simulation practices, representations, robust results, and efficiency. Engineering demands predictability and information that promotes risk avoidance. Science counts on precision and understanding. These demands on simulation are very different. Third, traversing all markets and practices, there exists a kind of common denominator of expectations. Practitioners demand 'truth.' The relevant form of truth might be described as 'practical truth.' Practical truth, as distinguished from the epistemological truth of philosophy, refers to dependable matter-ground individual and group satisfaction based on perceived reliable material inputs and outcomes. What counts here as 'satisfactory' is connected both to individual experience and to the expectations and norms of society. Practical truth is hence simultaneously concretely personal and the fruit of collective routines. Upon determining that technical objects and protocols yield the anticipated result, practitioners gradually come to equate a technical ensemble as 'true' to their understanding of the technical system's properties and ensuing yield. Indeed, stable controlled technical 'yield' comprises the key facet of pragmatic truth.

Heterogeneity of environment, form, and function are constitutive of the underlying logic and action of simulation. In view of the extremes of heterogeneity, what gives the simulation system substance, stability, and continuity? Part of the answer lies in research technology.

Several of the principal ingredients of research technology have been introduced in the above analysis: genericity, the interstitial environment, markets and technical niches, divisions of labor, relative autonomy, boundary crossing, reembedding, commensuration, practical truth, and lingua franca. Simulation is stretched between two imperatives: its multiple practices and markets; and the need to preserve a stable and standard kernel. Simulation necessitates unchanging standards in order to convince users that outputs are effective and comparable, and to provide ground rules for internal community communication and the further development of simulation techniques.

An intellectual and organizational formula must be evolved capable of ensuring transversality across practices and markets while preserving technical and community cohesion. Research technology contributes to this end. In the case of simulation, the emergence of a generic, general-purpose, multiparadigm code specifically designed with objects in mind, like C++, provides a balance between centrifugal and centripetal action. Spawned along tenets of research technology, C++ has given simulation the power to expand into ever more applications and markets through implementation tools (classes, portability, compilers, and standard template libraries). The ANSI and ISO standardization of the code has ensured uniformity and continuity, urgently called for by clients. The standardization process has allowed simulation practitioners and design specialists to further work out generic foundations. Finally, simulation boasts an interstitial environment, witnessed in the complex cognitive and organizational trajectories of Bjarne Stroustrup, John McLeod, and Vincent Amico.

GEMAS, Maison des Sciences de l'Homme, Paris, France

REFERENCES

Allison, C. (1996). Interview with Bjarne Stroustrup on his reaction to the imminent completion of the ANSI/ISO C++ Standard, *The C/C++ Journal,* October 1996, **14** (10); http://public.research.att.com/~bs/interviews.html (acc. April 13, 2004).

Dolya, A.V. (2003). Interview with Bjarne Stroustrup for the *Linux Journal*, posted August 28, 2003, http://public.research.att.com/~bs/interviews.html (acc. April 13, 2004).

IEEE Computer (1998). "Open Channel" Interview with Bjarne Stroustrup: The Real Stroustrup Interview. "The Father of C++ explains why standard C++ isn't just an object-oriented language", http://www.research.att.com/~bs/ieee_interview.html (acc. April 13, 2004).

IEEE Transaction (1968). Special Issue on Systems Science and Cybernetics, Vol. SSC-4, No. 4.

Joerges, B. and T. Shinn (eds.) (2001). *Instrumentation Between Science, State and Industry*, Dordrecht, NL: Kluwer Academic Publishers.

Kalev, D. (2001). "An interview with Bjarne Stroustrup", *LinuxWorld.com*, August 2nd, 2001, http://www.itworld.com/AppDev/710/lw-02-stroustrup (acc. April 13, 2004).

Koenig, A. and B. Stroustrup (1989). "C++: As close to C as possible – But no closer", *The C++ Report*, July 1989.

McLeod, J. (1968). Simulation: The Dynamic Modeling of Ideas and Systems with Computers, New-York: McGraw -Hill.

McLeod, J. (1982a). *Computer Modeling and Simulation: Principles of Good Practice*, The Society for Computer Simulation International.

McLeod, J. (1982b). "Simulation" in the Encyclopedia of Science and Technology, New-York: McGraw-Hill.

McLeod, J. and P. House (1977). *Large-Scale Models for Policy Evaluation*, New York: Wiley-Interscience.

Nelson, E. (2004). Interview with Bjarne Stroustrup for *Visual C++ Developers Journal (VCDJ)*, http://www.research.att.com/~bs/devXinterview.html (acc. April 13, 2004).

Pradeepa Siva C. (2004). Interview with Bjarne Stroustrup *for Addison-Wesly Longman* about the third edition of The C++ Programming Language, http://www.research.att.com/~nineteen nineteen bs/3rd_inter1.html (acc. April 13, 2004).

Shinn, T. and B. Joerges (2002). "The tranverse science and technology culture: Dynamics and roles of research technology", *Social Science Information,* **41** (2): 207–251.

ShinnT. and B. Ragouet (2005). *Controverses sur la science: Pour une sociologie transversaliste de l'activité scientifique*, Paris, Raisons d'agir.

Stroustrup, B. (1980). "Classes: An Abstract Data Type Facility for the C Language", *Bell Laboratories Computer Science Technical Report* CSTR-84.

Stroustrup, B. (1981). "Extensions of the C language type concept", *Bell Labs Internal Memorandum*.

Stroustrup, B. (1982). "Adding classes to C: An exercise in language evolution", *Bell Laboratories Computer Science Internal Document*.

Stroustrup, B. (1986) *The C++ Programming Language*, Reading, MA: Addison-Wesley.

Stroustrup, B. (1987a). "The evolution of C++: 1985–1987", *Proceedings USENIX C++ Conference*, Santa Fe, NM.

Stroustrup, B. (1987b). "Possible directions for C++", *Proceedings USENIX C++ Conference*, Santa Fe, NM.

Stroustrup, B. (1989). "Standardizing C++", *The C++ Report,* Vol. 1, No. 1.

Stroustrup, B. (1990). "On language war", *Hotline on Object-Oriented technology*, Vol. 1, No. 3.

Stroustrup, B. (1993). "The history of C++ : 1979–1991", *Proceedings ACM History of Programming Languages Conference (HOPL-2). ACM SIGPLAN Notices*.

Stroustrup, B. (1994). *The Design and Evolution of C++*, Indianapolis, IN: Addison-Wesley.

Stroustrup, B. (1997a). "The object magazine online interview", http://www.research.att.com/bs/omo_interview.html (acc. April 13, 2004).

Stroustrup, B. (1997b). *Design and Use of C++ by Bjarne Stroustrup*, Talk given November 3, 1997 at Computer Literacy Bookstore in San Jose, CA, http://www.research.att.com/~bs/Cbooks_QnA.html (acc. April 13, 2004).

Stroustrup, B. (2000). *C++ Answers From Bjarne Stroustrup*, e-mail interview with slashdot.org in the week of February 25, 2000, http://www.research.att.com/~bs/slashdot_interview.html (acc. April 13, 2004).

Stroustrup, B. (2004). Interview, http://members.shaw.ca/qjackson/writing editing/interviews/Bjarne-Stroustrup.htm (acc. April 13, 2004).

Venner, B. (2003). "The C++ style sweet spot. A conversation with Bjarne Stroustrup, Part I", October 13, 2003, http://www.artima.com/intv/elegance.html (acc. April 13, 2004).

"Modern C++ style. A conversation with Bjarne Stroustrup, Part II", November 24, 2003, http://www.artima.com/intv/modern2.html (acc. April 14, 2004).

"Abstraction and efficiency. A conversation with Bjarne Stroustrup", Part III", February 16, 2004, http://www.artima.com/intv/abstreffi2.htm (acc. April 14, 2004).

"Elegance and other design ideals. A conversation with Bjarne Stroustrup, Part IV", February 23, 2004, http://www.artima.com/intv/elegance.html (acc. April 13, 2004).