Anecdotes

Anne Fitzpatrick, Editor
Los Alamos National Laboratory

Philip L. Frana's description of the Halloween Problem demonstrates the role of cautionary tales in the history of computing. Laurie Robertson traces the lineage of the DEC software to the present day. Elaborating on the MESM development, Anne Fitzpatrick and Boris N. Malinovsky take a look at Sergei Alexeevich Lebedev's early career and work in the Ukraine.

A well-intentioned query and the Halloween Problem

The history of computer software is replete with cautionary tales meant to impart wisdom or convey a warning from one generation of hackers and programming managers to the next. Famous tales in software development include Fred Brooks' experience with OS/360 development ("Adding manpower to a late software project makes it later ...")[1] and Andy Tanenbaum's shootout with Linus Torvalds over Linux ("I still maintain the point that designing a monolithic kernel in 1991 is a fundamental error. Be thankful you are not my student. You would not get a high grade for such a design ....")[2]. *The New Hacker's Dictionary* abounds with admonitions drawn from cautionary tales.[3]

The cautionary tale is not unique to computer science and allied disciplines. It is a staple of all storytelling, but it certainly has been facilitated by the open-ended nature of this profession, which prizes and finds distinction in nontraditional or alternative methods of communication and education, independence of thought and action, and a perpetual spirit of reinvention.[4] This may also be why the *IEEE Annals* Anecdotes column has remained such a strong and vibrant institution.

Perhaps the most famous cautionary tale in database circles is the Halloween Problem. The Halloween Problem emerged in the context of structured query language optimization in relational database research. The relational database was the brainchild of Ted Codd,[5] an IBM researcher working in the late 1960s who felt that the navigational model of database design had many faults hindering its effective implementation. The navigational model represents information in terms of the presence of connections (or pointer-paths) between individual records. Programs written in specialized languages are necessary to navigate these connections, and exceptional burdens are placed on the database user to precisely specify the query process.[6] This process is further complicated, however, because the algorithm developed by the user in making the query wholly depends on the immediate underlying data structure, which may change as the database grows. This is especially true when a database is shared between many users, as in a time-sharing environment.

In a series of papers published between 1970 and 1972, Codd showed that a more explicit focus on data independence, which his colleague Chris Date succinctly defined as "immunity of applications to change in storage structure and access strategy,"[7] coupled with a high-level relational query language might be just the tonic the database community needed to make the technology more accessible to everyone. The cornerstone of Codd's work is the relation, where each piece of data and every relationship is defined as a record. No other connections are needed. As Date put it, "In a relational database, the only essential data construct is the relation itself."[8]

Codd's ideas came to partial fruition in System R,[9] a prototype relational database management system developed at the IBM San Jose Research Laboratory (later renamed the IBM Almaden Research Center), nestled in Santa Teresa County Park. Many IBMers at the San Jose Lab came to share Codd's passion for user-friendly databases including Ralph Gomory, Jim Gray, and Don Chamberlin. Gomory, head of IBM's Research Division, believed in taking substantial risks despite the potential for failure. Gray would eventually accept a Turing prize for fundamental contributions to transaction processing. Chamberlin later became an ACM Fellow for his role in designing SQL (Structured Query Language) for System R and the popular product DB2.

System R and the multiuser relational model required several new or revised approaches. A theory of normalization was developed to reduce redundancy and logic conflicts in the data. New buffer management algorithms were introduced to control the flow of pages between random access disks and main memory. And new definitions of the underlying properties of transactions had to be formulated: atomicity, consistency, durability, and isolation. The model also inspired serializability, where transactions executed in parallel are executed in the database environment in a particular order.

Most importantly for our story here, System R researchers spent a great deal of time developing the theory and method of query optimization. A query is simply a request for information to be gleaned from the database. For instance, an SQL query to retrieve a list of names of employees making less than $20,000 per year might be written

 1058-6180/02/$17.00 © 2002 IEEE

```
SELECT FIRST_NAME, LAST_NAME
FROM EMPLOYEE [employee database table]
WHERE SALARY < 20,000
```

A relational query optimizer chooses from several alternative strategies in determining an efficient access path to the "right" data set based on the request of the database user. In effect, it automates the expert programmer's responsibility for developing a retrieval plan. Query optimizers make the mapping of connections—a staple of the navigational model—invisible and facilitate the creation of a user-friendly interface where a high-level query language (in this case SQL) replaces the painstaking building of more complex algorithms.

Normally, a query optimizer works by measuring system calls and paging requests and applying heuristics to the entire access path tree. Query optimization was one of the most challenging tasks facing System R researchers at IBM. These experiments with query optimization form the milieu in which the Halloween Problem emerged.

Queries also may be used to update appropriate data sets. In SQL, the command is simply called UPDATE. Thus, the query

```
UPDATE EMPLOYEE
SET SALARY = SALARY * 1.1
```

will raise the salary of each employee by 10 percent.

Two of Chamberlin's colleagues, Pat Selinger and Morton Astrahan, were using the SALARY index to test a particular optimizer when the Halloween Problem cropped up. The SALARY index essentially is an access path for obtaining ordered data from one field in the EMPLOYEE table.

"If you have an index by Social Security number," explained Chamberlin, "then it is really easy to find a person that has a particular Social Security number, and if you access the data using that index, all of the entries will appear in order by their Social Security number."[10]

The most recent eyewitness account of the Halloween Problem comes from a Charles Babbage Institute Software History Project oral history with Don Chamberlin[10] conducted with support from the National Science Foundation in 2001.[11] Chamberlin's recollection is as follows:

> Morton and Pat [were] working on translating high-level queries and updates into plans that use various access paths including indexes. Well, one example of an update that we were playing around with was giving raises to people. The example said, 'Give a ten percent raise to everybody who earns less than twenty-five thousand dollars.' So Pat said, 'Let's imagine that we are a query optimizer, and we are trying to figure out some way to process this query.' First we need to find all the employees who earn less than twenty-five thousand dollars. Well, a really good way to do this would be to use a SALARY index, because using that index you could scan through the data in salary order and, of course, the first people you come to are going to be the lowest paid ones, so that would be very efficient. So using the SALARY index, we'll go and find the first guy, he's the guy with the lowest salary, and if it's less than twenty-five thousand dollars we'll give him a raise. Let's suppose that he earned ten thousand dollars. So according to this plan, the query processor would find this guy and give him a raise and now he wouldn't earn ten thousand dollars anymore, he'd earn eleven thousand dollars and that would cause him to jump ahead in the index. He would still be in the index, but he wouldn't be in the place he was before anymore, he'd move ahead in salary order. So then, we'd go along and process the next guy and the next guy and pretty soon we'd come to the guy that we gave a raise to already, because now he was ahead in the order, and when we came to him again he'd still earn less than twenty-five thousand dollars, so we'd give him another raise and so on. So this guy might jump ahead in the order of the SALARY index several different times. In fact, everybody that earned less than twenty-five thousand dollars would get as many raises as they needed until they earned more than twenty-five thousand dollars. And you can see what the problem is here. That obviously was not the intention of the update, but that's the way that it might be naively implemented if a SALARY index were being used as the access path.

Another account comes from Pat Selinger, who was responsible for creating the first cost-based query optimizer.[12] She related her version of the story in 1995 on the occasion of the 20th anniversary celebration for System R:[13]

> We had exercised the 'person who earns more than their manager' query to death, and finally got to the point where the optimizer was choosing indexes sometimes to implement this query and it happened to think that the SALARY index was a pretty good index to select for this. And having selected the SALARY index for the first time in us testing out the optimizer, we ended up discovering that this query didn't stop. Because we were using the SALARY index to go after the

> EMPLOYEE table and we were also updating it, Don Chamberlin kept getting more and more raises. Which made him very happy, but it made us optimizer folks a little bit uncomfortable. So Morton and I sat down and discovered this and analyzed what was going on, and came to one of [Chamberlin's] RDS meetings [RDS stood for Relational Data System, at the heart of which was an optimizing SQL processor] and it happened to be on Halloween. So we ended up telling the group about this and consulting the general wisdom to figure out what in the world we ought to be doing about this thing.

No account appears to have been left by the third eyewitness, Morton Astrahan, who died in the mid-1980s.

A good cautionary tale is subject to misattribution, misappropriation, and mystery. The Halloween Problem is no exception. It is sometimes argued that the Halloween Problem is so named because of the unnatural havoc it wreaked on relational databases. This is not the case. Chamberlin has noted,

> Pat and Morton discovered this problem on Halloween .... I remember they came into my office and said, 'Chamberlin, look at this. We have to make sure that when the optimizer is making a plan for processing an update, it doesn't use an index that is based on the field that is being updated. How are we going to do that?' It happened to be on a Friday, and we said, 'Listen, we are not going to be able to solve this problem this afternoon. Let's just give it a name. We'll call it the Halloween Problem and we'll work on it next week.' And it turns out it has been called that ever since.[14]

System R manager Mike Blasgen confirms this view:

> It's interesting because all these odd-ball things had names: there were phantoms, and there were other things, and those had to do with names that were somehow representative of what you were observing, right? So the phantom was because it was something that was sort of there, but not there; the name was descriptive. And this was called the Halloween Problem not because it surprises you, or it's spooky, or trick-or-treat or anything; this is because it happened to be discovered on Halloween day. But I think most people think it's the other; I think most people think it's called Halloween because it's so surprising. But it's not.[15]

The name Halloween Problem has also been misappropriated in describing other nettlesome queries. For instance, the Lost Update Problem, where modifications to a database by one user are overwritten because of changes made by another user accessing the database records simultaneously, is not equivalent to the Halloween Problem (which instead might be thought of as a redundant update problem).[16]

Then there is the mystery, Which Halloween? Selinger and Chamberlin have agreed that the Halloween Problem cropped up in 1976 or 1977. Chamberlin in particular remembers Astrahan and Selinger landing in his office on a Friday afternoon with their query update puzzle. But 31 October 1976 was a Sunday, and 31 October 1977 was a Monday. The holiday fell on a Friday only once during the 1970s, on 31 October 1975. Fortunately, this conundrum does not detract from the moral of this story: Always live within your income, even if you have to borrow to do so.

Philip L. Frana
Charles Babbage Institute
frana003@tc.umn.edu

References and notes

1. F.P. Brooks, Jr., *The Mythical Man-Month*, rev. ed., Addison-Wesley, Reading, Mass., 1995, p. 25.
2. Andrew S. Tanenbaum to Linus Benedict Torvalds, as quoted from the post to Usenet group comp.os.minix: "Re: LINUX is obsolete," 30 Jan. 1992, 13:44:34 GMT.
3. E. Raymond, *The New Hacker's Dictionary*, MIT Press, Cambridge, Mass., 1991.
4. Marvin Minsky, for instance, has argued that his particular expertise, artificial intelligence, is conterminant with computer science as a whole. Founders of various other computer science disciplines have argued likewise. See M. Minsky, "Introduction," *Artificial Intelligence Memoranda*, AI Laboratory, Massachusetts Institute of Technology, 1958–1979, Scientific DataLink, New York, 1982, pp. vi-vii. For primary sources devoted to issues of professionalism in computer science, see J.A. Archibald, Jr., and M. Katzper, "On the Preparation of Computer Science Professionals in Academic Institutions," *Proc. 1974 Nat'l Computer Conf. and Exposition*, AFIPS Press, Montvale, N.J., 1974, pp. 313-319; G.F. Palmer, "Programming: The Profession That Isn't," *Datamation*, vol. 21, no. 4, 1975, pp. 171-173; and "The 'Professional' Programmer: Some Industry Opinions," *Data Processing Magazine*, vol. 12, Jan. 1970, pp. 18-21.
5. See the brief biographical sketch "Edgar F. Codd," by J.R. Yost in *Encyclopedia of Computers and Computer History*, vol. 1, R. Rojas, ed., Fitzroy Dearborn Publishers, Chicago, 2001, pp. 161-162.
6. On the navigational model for database design, see C.W. Bachman, "The Programmer

as Navigator," *Comm. ACM*, vol. 16, no. 11, 1973, pp. 635-658.

7. C.J. Date, *An Introduction to Database Systems*, 2nd ed., Addison-Wesley, Reading, Mass., 1977.
8. C.J. Date, "Thirty Years of Relational: Relational Really is Different," *Intelligent Enterprise Magazine*, vol. 2, no. 7, 11 May 1999.
9. On the technical history of System R, see D.D. Chamberlin et al., "A History and Evaluation of System R," *Comm. ACM*, vol. 24, no. 10, 1981, pp. 632-646. On the history of database design, generally see J. Gray, "Evolution of Data Management," *Computer*, vol. 29, no. 10, 1996, pp. 38-46, and A. Silberschatz, M. Stonebraker, and J. Ullman, "Database Systems: Achievements and Opportunities," *Comm. ACM*, vol. 34, no. 10, 1991, pp. 110-120. For a brief popular gloss on System R, see S. Lohr, *Go To*, Basic Books, New York, 2001, pp. 161-68.
10. See D.D. Chamberlin, OH 329. Oral history interview by Philip L. Frana, 3 Oct. 2001, San Jose, California. Charles Babbage Inst., Univ. of Minnesota, Minneapolis, http://www.cbi.umn.edu/oh/.
11. NSF Knowledge and Distributed Intelligence Directorate grant #9979981.
12. Selinger became an IBM Fellow in 1994 for her work on cost-based optimizers. Five years later, she was elected to the National Academy of Engineering. She has directed the IBM Database Technology Institute for 13 years.
13. Selinger's comment is transcribed in P. McJones, *The 1995 SQL Reunion: People, Projects, and Politics*, SRC tech. note 1997-018, Aug. 1997, http://www.mcjones.org/System_R/.
14. Quotation from Donald D. Chamberlin, Charles Babbage Institute, OH 329.
15. Quotation from P. McJones, *The 1995 SQL Reunion: People, Projects, and Politics*, SRC tech. note 1997-018.
16. See Z. Qingqing, "Relational DBMS Implementation: OS vs. DBMS," unpublished essay, Jan. 2002, http://www.cs.toronto.edu/~zhouqq/readings/r_OS.html. Jim Gray and Rick Cattell have described another such update query that has been confused with the Halloween Problem. See J. Gray, *The Benchmark Handbook*, 2nd ed., Morgan Kaufmann, San Francisco, Calif., 1993, chapter 4.

A software lineage

Computer operating systems manage computer resources, control the execution of application programs, and provide system functions for interfacing with the central processor and peripheral devices. Because operating systems provide the interface between the computer hardware architecture and computer programs and human users, they are typically discussed as inseparable from the underlying machine. However, just as hardware architectures have family genealogies, so do operating systems. This article examines the legacy of one lineage that survives to the present day.

The hardware legacy of the Digital Equipment Corporation (DEC) is well documented. From its humble beginnings in an abandoned New England textile mill as a specialized peripheral manufacturer, DEC grew to become one of the dominant computer companies in the 1970s and 80s. DEC engineers produced landmark computers, such as the PDP-8, PDP-11, and VAX. At the company's height, DEC hardware was synonymous with university, engineering, and research computing, but its machines could also be found in hospitals, financial institutions, and offices.

The PDP-11 was DEC's first hardware-compatible 16-bit computer. Prior to the introduction of PDP-11, DEC had produced four disparate hardware families. The PDP-11 was the start of a family of hardware-compatible computers.[1] The PDP-11 was designed to save customers time and money by letting them move applications easily between machines—large or small—as their needs changed.[2] But DEC's vision of application transportability was complicated by the PDP-11's several specialized, incompatible operating systems targeted for various markets. For example, RT-11 was a real-time operating system designed to support laboratory applications, and RSTS-11 (Resource Sharing/Time Sharing) was a time-sharing operating system designed to support commercial applications.

RSX-11 (Resource Sharing Executive) was one of the most robust PDP-11 operating systems. Designed by David Cutler, RSX-11 was intended for industrial and manufacturing control users. It offered users a "sophisticated real-time executive, on-line program development, complete device handling capabilities, and total system protection."[3] Subsequent enhancements to RSX-11 included memory management, robust communications support, and batch programming.

By the late 1970s, DEC needed a 32-bit successor to its 16-bit PDP-11 minicomputer. Although the PDP-11 had been a widely successful computer, it had four different operating systems, which supported "a variety of incompatible languages, data management, and transaction processing software."[4] To solve this software chaos, DEC wanted a machine that could provide a single, homogeneous, distributed computing environment. To succeed the venerable PDP-11, DEC pro-

duced the VAX, probably the most successful minicomputer in history.

Known for its expandability and reliability, the VAX enabled DEC to grow from a small, niche hardware manufacturer into a major worldwide computer company. As a result, DEC engineers defined a set of standard common attributes that enabled all VAX models to run the same software. Although originally introduced as minicomputers, the VAX product line evolved to encompass multiprocessor superminicomputers and single-user workstations. The VAX was the "ultimate complex instruction set computer (CISC)"[4]—the original VAX architecture included 244 instructions, which later grew to more than 350.[5] Adding to its complexity, the VAX included complete PDP-11 hardware compatibility, which enabled it to run PDP-11 user code with the same amount of memory as the PDP-11.[6]

The Virtual Memory System (VMS) was the principal operating system for the VAX. Designed by David Cutler, VMS was a sophisticated, robust operating system that exploited the capabilities of the VAX architecture. Drawing on many RSX features, VMS provided multiuser time-sharing, batch, real-time, and transaction processing capabilities as well as supported multiprocessing, loosely coupled clustered systems, and multivendor wide-area networking. Furthermore, VMS ran both RSX-11 and RSTS operating system code. Throughout the 1980s, VAXs running VMS dominated the minicomputer market.

In the mid-1980s, a new technology emerged: reduced instruction set computers (RISC). RISCs use a small, simple instruction set optimized to perform specific functions. Although a RISC may execute more instructions than a CISC, it does so faster and more efficiently. RISC architectures enabled the development of the new workstation computer, usually running a variant of the Unix operating system. New companies, such as Sun, as well as established companies, such as IBM, raced to introduce RISC workstations.

DEC was slow to recognize the threat of the RISC workstations. VAX sales had catapulted DEC into the Fortune 100, so its engineers and management focused on enhancing and refining its architecture. Only in 1990 did DEC belatedly respond with its own RISC architecture, the Alpha. For the operating system, DEC ported VMS to the new platform and renamed it Open VMS to signify its high degree of support for industry portability standards. But DEC never ported VMS to non-DEC platforms. DEC was primarily a hardware company and viewed software as a supporting technology. Furthermore, VMS was difficult to port. It was written in a variety of languages[7] and required specific hardware features only available in DEC hardware. The Alpha architecture never caught on, and DEC's financial situation continued to deteriorate throughout the 1990s. Finally, in 1998, Compaq acquired DEC and discontinued production of both the VAX and Alpha.

Up to this point, the lineage of VMS and its predecessor RSX are tied to DEC hardware. With the demise of the Alpha architecture, they became orphaned operating systems. Surprisingly, they live on in an operating system partly responsible for its demise—Microsoft's Windows NT.

Yet another David Cutler operating system, Windows NT was designed to be "Microsoft's operating system for the 1990s."[8] However, like VMS, Windows NT could not ignore Microsoft's previous success. It needed to run on the same hardware (primarily Intel platforms) and be compatible with existing Microsoft operating systems, especially MS-DOS. Cutler gave Windows NT many new and innovative technologies, such as system resource objects, but for its core, he modified the proven technologies within VMS. Modified VMS components such as the file structure, the input/output system, and most importantly, the kernel, provide the crucial low-level foundation for Windows NT.[9] Today, Windows NT and its successors, Windows 2000 and Windows XP, are the dominant operating systems on Intel hardware.

Although much has been written about the genealogies of computer hardware and programming languages, similar lineages exist throughout computer software. From the development of industry-wide technologies such as transaction processors to the evolution of company-specific computer portfolios, there are numerous family trees yet to be documented. This article has introduced just one of software's many genealogies.

Laurie Robertson
Titan Systems
lroberts@ieee.org

References and notes

1. C.G. Bell, J.C. Mudge, and J.E. McNamara, *Computer Engineering: A DEC View of Hardware System Design*, Digital Equipment Corp., Bedford, Mass., 1978.
2. J. Pearson, *Digital at Work*, Digital Press, Burlington, Mass., 1992, p. 60.
3. *Digital Equipment Corporation: Nineteen Fifty-*

Seven to the Present, Digital Equipment Corp., Bedford, Mass., 1978, p. 31.

4. C.G. Bell, *A Retrospective on What We Have Learned From the PDP-11: What Else Did We Need To Know That Could Have Been Useful in the Design of the VAX-11 to Make Alpha Easier*? 1998, http://research.microsoft.com/users/GBell/Digital/Bell_Retrospective_PDP11_paper_c1998.htm.
5. The VAX instruction set provided special instructions for procedure call and return, saving and restoring process context, array index computation, polynomial evaluation, character string manipulation, and the transformation of packed decimal strings to character strings.
6. A. Ralston and E.D. Reilly, *Encyclopedia of Computer Science*, 3rd ed., Van Nostrand Reinhold, New York, 1993, p. 464.
7. The Open VMS FAQ (http://www.openvms.compaq.com/wizard/openvms_faq.html) identifies 14 different component languages used in VMS—Bliss, Macro, Ada, PLI, C, Fortan, UIL, SDL, Pascal, MDL, C++, DCL, Message, and Document.
8. H. Custer, *Inside Windows NT*, Microsoft Press, Redmond, Wash., 1993, p. 4.
9. Ibid., pp. 20, 203.

The MESM and the monastery

The first electronic digital computer in continental Europe was constructed not in Germany or France, but in the Ukraine. In the late 1940s, Sergei Alexeevich Lebedev began constructing the MESM (Malaya Elektronaya Schetnaya Mashina, or Small Electronic Calculating Machine).

Although computer historians Gregory Crowe and Seymour Goodman in their 1994 article trace the early career and work of Sergei Alexeevich Lebedev and the creation of the MESM in fine detail, many of the events and culture that surrounded this project deserve a closer look.[1] We will describe some of them here.

At the time of the Communist revolution—when Lebedev was only 15 years old—the majority of the Russian population was illiterate. To his advantage, Lebedev's father was a school teacher from the Russian intelligentsia and the family valued education. Also, the Soviet Union's leadership valued education, believing that a functioning and effective Communist society required a highly literate population. By the height of the Cold War, the Soviet Union had more scientists and specialized engineers than any other nation in the world.[2]

Lebedev joined the waves of young people studying science and engineering in the wake of 1917. In 1923—the same year that the Ukraine joined the Soviet Union—Lebedev entered the Baumann Technical Institute in Moscow and majored in the field of high voltage technology. He completed a diploma thesis on *The Stability of Parallel Work of Electric Power Stations* in 1928. He then enrolled at the Lenin State Electrical Engineering Institute and received his doctorate in 1939 for his work on the theory of artificial stability of power systems.[3] Upon graduating, Lebedev began to supervise the Automation Department of the Electrical Engineering Institute, which oversaw national power engineering projects because Russia and the Soviet Union were gradually electrified and industrialized.[4]

Almost every project in the field of power engineering developed by the Lenin State Electrical Engineering Institute's scientists and engineers required what were considered at that time elaborate computing facilities. For example, for the calculations on the giant 9,600-MW, 1,000-Km-long electric power line—the Kuibyshev-Moscow hydroelectric project—Lebedev and his colleagues had to develop a highly automated set of powerful inductors and capacitors that simulated the mathematical model of the line.

World War II also sparked Lebedev's interest in computers. Toward the war effort, Lebedev developed an analog computer to assist with differentiation and integration calculations for a tank gun stabilization system.

In 1946, Lebedev accepted a post as Director of the Ukrainian Academy of Sciences Institute of Energy. In a 1947 reorganization, it was split into the Institute of Thermal energy and the Institute of Electrical Engineering.[5] Lebedev headed the latter institute when he arrived in Kiev in 1948.

By this time, Lebedev was already well known, and students migrated from Russia to the Ukraine to study and work with him. Lebedev also brought with him to Kiev his new idea to construct a large electronic digital computer, which was a complete novelty to most of his students.

After presenting his plans to the Ukraine Academy of Sciences and securing funding and workers, the Academy grudgingly gave him a space for the project—a good distance out of town in an area called Feofania. Kiev had been nearly leveled by World War II, so housing and industrial space were both at a premium.

Feofania lies about 15 Km southwest from the Kreschatik, Kiev's main street. Up until the time that the Ukraine joined the Soviet Union, Feofania was famous in the Eastern Orthodox Church as home to the St. Panteleimon the Healer Monastery, which was active until it was

closed under the Soviet dictum of atheism.

The Soviet government converted the monks' two-story 500-square-meter red brick dormitory into a branch of the Kiev Psychiatric Hospital. When World War II broke out, the building's occupants were abandoned there. As soon as the Wehrmacht reached Feofania in 1941, they killed all the psychiatric patients and set up a hospital for themselves there. (They also murdered four-fifths of Kiev's population.) By the time the Soviet Army liberated Kiev in 1943, the hospital building was badly damaged. The Ukraine Academy of Sciences subsequently acquired it for use as a laboratory. When Lebedev and his team began to work there, they had to do much of the building repair and maintenance themselves.

Getting to Feofania could be harrowing. There was no public transportation that reached that far, and from autumn to spring, the roads were practically impassable—the Ukraine's notorious mud in the colder months made for stuck vehicles. However, in summer, Feofania became extremely pleasant. The area was surrounded by oak trees, and all kinds of fauna and plants flourished there. The MESM workers frequently engaged in the favorite Slavic cultural pastimes of gathering mushrooms and berries.

Given the difficulties in getting there during the majority of the year, for convenience, many of the scientists and engineers moved out to Feofania, including Lebedev, his wife Larissa Grigorievna, and their children. Most of the MESM team lived there like one large extended family, working and playing together. Oktobrysa Malinovskaya (Malinovsky's wife) lived with her husband and young son at Feofania. She described the environment there as friendly and that participants truly enjoyed their jobs, creating something new and experimental. Malinovskaya described Lebedev as a strong, silent man who worked all the time. He expected his employees to clock in to work at the laboratory at 9:00 a.m. each working day, and as they would sign in, he rarely said a word.[6]

Among the workers, Lebedev's two chief deputy engineers were Lev N. Dashevsky, who, along with the tough but brilliant Ekaterina A. Shkabara, led the hardware design effort. Another woman, Ekaterina L. Yushenko, wrote MESM's first programs. Her subsequent publications on programming became famous in the Russian-speaking world.[7]

Lebedev and his team estimated that the MESM would occupy about 50 square meters, and thus the team had to remove some of the building's inner walls and part of the first floor ceiling. Yet they had not considered another consequence: when the entire unit was assembled and the power turned on, the MESM's 6,000 vacuum tubes turned the dormitory into a sauna. Air conditioners did not exist in postwar Soviet Ukraine, so the team cut a hole in the building's roof to draw off some of the heat.[8]

The machine was a success in terms of its architecture and operating capabilities. By 1950, the experimental MESM was functioning and operated at 50 instructions per second. It contained 63 three-address instructions and a memory of 31 16-bit words.[9] It used fixed-point binary numerical representation. At that time, the only other similar working machines were Frederick Williams and Tom Kilburn's Baby and Maurice Wilkes' Electronic Delay Storage Automatic Calculator in England. However, each British computer employed a sequential operational arithmetic unit, while MESM worked on parallel arithmetic units.

Other than a simple demonstration to representatives from the Academy of Sciences, the first actual problem MESM solved was on 25 December 1951, when it computed a probability distribution function:

$$p = \frac{1}{C_{2n}^{n}} \sum_{k=-\left[\frac{n}{\alpha}\right]}^{\left[\frac{n}{\alpha}\right]} (-1)^k C_{2n}^{n-k\alpha}$$

For two and a half hours, 585 values of p were calculated to five places, requiring nearly 250,000 operations. The result provided values for defining and increasing the accuracy of artillery weapons.[10]

When word got out that there was an operating computer in the Ukraine, a steady parade of Kiev and Moscow scientists with scientific and defense-related problems that could not be solved without the aid of a computer began to head to Feofania.[11] By 1952, MESM remained the only fully operational computer in the Soviet Union.

MESM was completely occupied with assisting in the solution of the highest priority defense-related problems and what were then some of the hardest problems in physics: a fragment of Yakov Zeldovich and Andrei Sakharov's thermonuclear weapon problem, and parts of Sergei Korolev's space flight trajectory calculations.

Although nuclear weapons scientists and

others used MESM initially for defense-related projects, computing as a field was not considered a military priority in the postwar Soviet Union and thus did not benefit from the extensive espionage network that was part of the Russian atomic bomb project, for example. MESM was an entirely independent creation. Although well aware of the American and British electronic digital computer projects from the secondary literature available, Lebedev and his team completed MESM without primary knowledge of foreign computer architecture.

Although Lebedev returned to Moscow in 1953 to head the Institute for Precision Mechanics and Computer Technology to begin work on the BESM (Bolshaya Elektronaya Schetnaya Mashina) series of computers, he left a thriving computer science legacy in Kiev, and his students continued to develop remarkable hardware and software under the leadership of Victor M. Glushkov. The MESM remained in operation in Feofania until 1957, when it was disassembled and the components were given to the Kiev Polytechnical Institute for student use. Many Ukrainian engineering students' first hands-on experience with computing equipment were with the MESM's various parts.

The building at Feofania was taken over by the Academy of Sciences Institute of Mechanics in 1957 and used as a laboratory for many years. When the Soviet Union was dissolved in 1991, the Ukraine quickly declared its independence, and the Academy of Sciences returned the Feofania dormitory building back to the Orthodox Church. It is now an active convent.

Anne Fitzpatrick
Los Alamos National Laboratory
Afitzpatrick@lanl.gov

Boris N. Malinovsky
V.M. Glushkov Institute of Cybernetics
icfcst@icfcst.kiev.ua

References and notes

1. G.D. Crowe and S. Goodman, "S.A. Lebedev and the Birth of Soviet Computing," *IEEE Annals of the History of Computing*, vol. 16, no. 1, Spring 1994, pp. 4-24.
2. L.R. Graham, *What Have We Learned about Science and Technology from the Russian Experience?* Stanford Univ. Press, Stanford, Calif., 1998.
3. B.N. Malinovsky, *Istoriya Vuichislitel'noi Tekkhniki v Litsakh* [A History of Computing Technology in Personalities], KIT, Kiev, 1995, pp. 23-27.
4. J. Coopersmith, *The Electrification of Russia, 1880–1926*, Cornell Univ. Press, Ithaca, N.Y., 1992.
5. G.D. Crowe and S. Goodman, p. 5.
6. Fitzpatrick personal communication with B.N. Malinovsky and O.A. Malinovskaya, 1 May 2001, Kiev.
7. See, for example, V.M. Glushkov, G.E. Tseitlin, and E.L. Yushenko, *Algebra, Iazyki, Programmirovanie* [Algebra, Languages, Programming], Akademiia nauk Ukrainskoi SSR, Kiev, 1974.
8. B.N. Malinovsky, p. 37; Fitzpatrick personal discussion with Zinovuiy L. Rabinovich, July 2000, Kiev.
9. S.V. Klimenko, "Computer Science in Russia: A Personal View," *IEEE Annals of the History of Computing*, vol. 21, no. 3., July–Sept. 1999, pp. 16-30; Fitzpatrick personal discussion with Malinovsky, July 2000, Kiev.
10. B.N. Malinovsky, p. 33.
11. L.N. Dashevsky and E.A. Shkabara, *Kak eto nachinalos'* [How it Began], Znaniye, Moscow, 1981, p. 53.