Early Language and Compiler Developments at IBM Europe: A Personal Retrospection

Albert Endres, Sindelfingen/Germany

Abstract: Until about 1970, programming languages and their compilers were perhaps the most active system software area. Due to its technical position at that time, IBM made significant contributions to this field. This retrospective concentrates on the two languages Algol 60 and PL/I, because with them compiler development reached an historical peak within IBM's European laboratories. As a consequence of IBM's "unbundling" decision in 1969, programming language activity within IBM's European laboratories decreased considerably, and other software activities were initiated. Much knowledge about software development was acquired from this early language activity. Some of the lessons learned are still useful today.

Introduction

For many of us, compiler construction was the first encounter with large software projects. This was certainly the case at IBM during the time period considered here. IBM's contribution to the history of both programming languages and compiler technology are known and well documented. I like to refer to Jean Sammet [Samm81] for a general discussion and to John Backus [Back78] and George Radin [Radi78] for the history of FORTRAN and PL/I, respectively. A survey of compiler technology in IBM is given by Fran Allen [Alle81].

With this paper, I intend to complement the reports cited above by adding some information that is not available in the academic literature. I will concentrate on contributions made by IBM's European development laboratories, an organization I was associated with during the major part of my career. To illustrate the historical setting and the beginnings of our industry, I will start out with some remarks on my personal experiences.

First Experiences as a Language User

My first encounters with computers date back exactly 50 years. It was the summer of 1956 when I attended my first programming class in Columbus, Ohio. It was on the IBM 650. I was an exchange student then at Ohio State University, and the machine was installed at the Battelle Memorial Institute close to the campus. The IBM 650 system can be best characterized by its main storage medium, a rotating magnetic drum, and its punched-card I/O devices. The drum comprised either 1000 or 2000 words, with a word length of ten decimal characters plus sign. With about 2000 installations worldwide, the IBM 650 was the most popular computer of the 1950s.

When I joined the IBM 650 Data Center at Sindelfingen, Germany, in 1957 as an application programmer I earned my livelihood by programming this machine. As a decimal machine, it was not difficult at all to write programs directly in machine code (which we did, of course). Nevertheless, we were relieved when George Trimble and Elmer Kubie [Trim01] of the IBM Endicott lab supplied us with the Symbolic Optimal Assembly Program (SOAP). This program allowed us to write in symbolic assembly language and placed the instructions on the drum in such a way that the latency of the rotating storage was minimized. The throughput gain compared to a sequentially written program approached a factor of six to seven. An easy-to-use approach existed for most engineering applications. The workhorse tool used was the

Bell Interpreter. This system, which had been developed by V. R. Wollontis [Wolo86] from Bell labs, was part compiler, part interpreter.

Fig. 1: IBM 650 system (Photo IBM)

It first converted the numeric source program to an intermediate language, then compiled portions of it, depending on the drum memory available. It provided for floating point calculations in a three-address language with simple indexing and also included excellent debugging facilities. The programs executed considerably slower than those written in SOAP, but they were very short and could be modified easily. In a sense, the Bell Interpreter was a precursor of APL.

FORTRAN appeared on the IBM 650 in early 1959, about a year later than on the IBM 704. The first implementation was called FORTRANSIT. This effort had been initiated and managed by Bob Bemer at the IBM Applied Science department in New York City [Hemm86]. The compiler used a cascading approach, first translating FORTRAN source statements to Internal Translator (IT), then to SOAP and finally to machine code. As a curiosity, one had to stay close to the machine during compilations, because different control panels had to be inserted into the reader/punch for each step of the cascade. The accompanying documentation nevertheless referred to FORTRANSIT as an Automatic Coding System. The language IT as well as its processor had been developed by Alan Perlis, then at Carnegie Mellon University in Pittsburg, PA. A regular one-step FORTRAN compiler was provided two years later by Florence Pessin's group from IBM in New York City.

Starting in 1959, after moving within the company from Sindelfingen to Düsseldorf, I found myself and my group confronted with engineering and business applications at the same time. Had I been struggling with the Runge-Kutta method for differential equations before, now the timely execution of a payroll program for construction workers contributed to my happiness. The computer of interest was first the IBM 650 again, but later the IBM 7070. The latter machine featured 10k characters of magnetic core memory. The languages used were Autocoder, Commercial Translator, and COBOL. Autocoder was a somewhat exaggerated name for the 7070 macro assembler. Commercial Translator (or COMTRAN) was IBM's precursor of COBOL. The IBM COMTRAN compilers (whose developments were led by Roy Goldfinger and Dick Talmadge) were quite efficient, with the effect that they delayed the acceptance of COBOL on IBM hardware. The first COBOL language document appeared in 1960. The COBOL compiler for the IBM 7070 became available in 1962.

In my first five years in the industry, being an application programmer first and then a data center manager, I came into contact, mainly as a user, with all of the seven languages mentioned above. In addition, I became somewhat familiar with FORTRAN on the IBM 704 and with the 1401 Assembler. Starting in 1962, I switched from a language user to a compiler developer and language designer. Later, I became a developer of other types of software, such as operating systems, data management and teleprocessing packages, and development tools.

The Programming Environment during the Early 1960s

"Compiler writing has long been a glamour field within programming and has a well-developed folklore." With this statement Jerry Feldman and David Gries [Feld68] open a

survey paper in 1968. With the hindsight of some 40 more years in the field, this statement provokes a few comments. What Feldman and Gries considered as a "long" period, amounted to maybe 10-15 years then. Compiler writing started in about 1954, when John Backus[1] initiated the FORTRAN project. When talking of folklore, they were most likely referencing the then state of the art. It mainly relied on customs and traditions and very little on knowledge. Thanks to the academic interest that compiler writing has attracted, there is probably no area in software development that subsequently has benefited more from theoretical work.

With the advent of high-level procedural programming languages, the utilization of computers achieved a major breakthrough. Within a short period hundreds of languages arose. IBM offered some six to eight different language processors for each of its large machines. The tower of Babel was used by the "Communications of ACM" (January 1960) to characterize the situation. At the same time, the number of different machine architectures grew. Soon a need was seen to find ways to handle this multiplicity problem. It was considered as a, if not the, key technological challenge of the time.

One solution being pursued at IBM was a project named SLANG [Sibl61], led by Bob Sibley. The main ideas of this approach are depicted in Fig.2. Each program in a problem oriented language (POL) is first translated into a common intermediate language (CIL). With the aid of a machine description (MD), i.e. a data set specifying the target machine's data format and instruction set, the appropriate machine language (ML) code is generated.

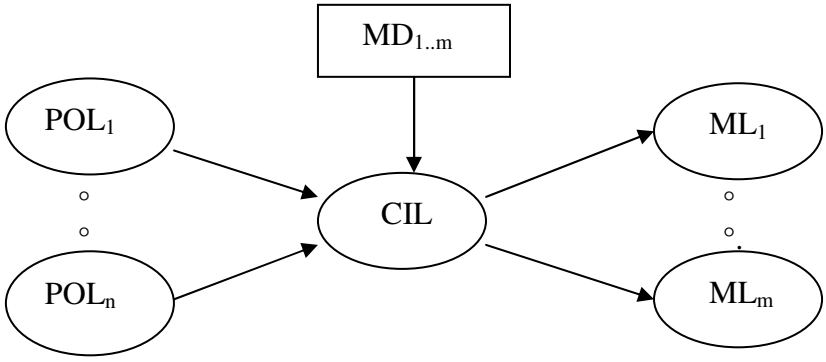

Fig 2: Multi-language multi-machine compiler

Although this concept was never implemented, Sibley's group , located at IBM's development laboratory in Poughkeepsie, NY., made contributions to the definition of PL/I and subsequently implemented PL/S, the IBM internal system implementation language.

Algol 60 Interludes

My change from language user to compiler developer occurred when I joined a project in La Gaude, France, funded by IBM Europe, attempting to provide an Algol 60 compiler for the IBM 7090. This laboratory had just occupied its architecturally impressive building. The project had been initiated by the afore-mentioned Bob Bemer, then IBM's liaison to standards authorities together with the scientific marketing organization in Europe. The effort was lead by Ken Hanford, an Englishman, and staffed with developers from Austria, France, Germany, Netherlands and Sweden.

[1] All papers cited in this survey that precede Backus (1957) are mathematical in nature (Church, Post, Tarski)

With the exception of Peter Lucas from Vienna, none of us had prior experience in compiler writing. The design of the compiler was hardly completed when the project terminated in the fall of 1963 with the approach of the System/360 announcement. Later I learned that an effort similar to this one had been carried out at the University of Illinois under the name ALCOR Illinois. It was there where Manfred Paul and Rüdiger Wiehle from the Technical University of Munich won their professional spurs.

Fig.3: IBM La Gaude laboratory (Photo IBM)

Starting in 1964, an Algol 60 compiler for OS/360 was designed at the Boeblingen laboratory by Hans-Jürgen Hoffmann and his group. When the PL/I D effort, described below, had to be staffed, the Algol project was transferred to the Swedish lab at Lindingo. The transfer was complete by the end of 1966, and the product was implemented and shipped from Lindingo. As Hoffmann[2] wrote in a retrospective in 1998 [Hoff98], the key problem was to decide on the language for input/output. Among the choices were FORTRAN-like facilities, the ECMA standard and a local proposal. The implementers gave in to what some potential users insisted on and decided for the IFIP proposal. The main storage allowed for the compiler, i.e. its design point, was 44 kB (F level) and it was intended to run under the operating system OS/360. It utilized the mathematical functions from the FORTRAN F library. It could link with routines written in assembly language or FORTRAN. Besides Hoffmann, the following members of the project team heavily contributed to the design: Klaus Alber, Elisabeth Müller and Hans Rachel.

Function group	***Nr. of phases***	***Detailed function***
Tokenizing; Identifier table manipulation	2	Convert external representation in unified internal code; numbers into internal format; storage allocation for variables and constants
Loop & subscript inspection, Optimization	1	Check validity of subscript expressions; separate into initialization and increment part. Allocate registers for loop indices
Syntactic analysis; Code generation	2	Left-to-right analysis based on operator-token stack; code generation in linkage editor format

Tab. 1: Structure of OS/360 Algol 60 compiler [Hoff98]

For IBM, Algol 60 played at that time a role similar to that of UNIX later. As long as IBM did not have an offer, it served other vendors as a vehicle to emphasize their unique strength. In certain markets we were told, we could not compete without it. As soon as the product was there, hardly anybody installed and used it. Either the hardware was criticized as being too expensive or other problems prevailed. This was prior to the unbundling decision, when selling hardware was the only way to justify a software investment.

After the 7090 Algol project, i.e. during the fall of 1963, I joined the NPL definition and evaluation team under Paul Comba and Nat Rochester in New York City. A colleague of mine, Willy Plöchl, became a member of Bob Sibley's group in Poughkeepsie, NY. New Programming Language (NPL) was then the name of the effort that was later renamed PL/I.

[2]After the project, Hoffmann joined the IBM Research lab in Zurich for while, before he became a professor of computer science in Darmstadt, Germany.

A Language Adventure Called PL/I

As is well known, IBM tried to solve the multi-language multi-machine problem described above, not in the way Bob Sibley had suggested, but in a radically different way. First, it decided to unify its machine architectures, leading to what became System/360. While all of us admired this tremendous intellectual effort, the software field lagged far behind. Only a few months before the announcement of System/360, the IBM software community was finally challenged to bring some order to the programming language side. This led to the definition of PL/I, i.e. a language to unify all languages. Much has been said about its high goals - and the results. Although closely associated with the effort, I have no problem to concede that it failed. There were many reasons for this. To use Fred Brooks' terminology, some of the reasons were accidental, i.e. of temporal or chance nature, others were essential or inherent to the problem. Let's look at the accidental ones first.

The language was designed by a committee, consisting of three people from IBM and three from its main user groups, SHARE and Guide [Radi78]. We were, of course, well aware of the joke floating around concerning this approach. The technical influences on this group of people came from many directions, most of them conflicting. The colleagues from IBM US labs and marketing centers wanted to build on the success of FORTRAN and gain from its high reputation. Therefore its expression format and variable naming conventions should be preserved. IBM Europe insisted most strongly that the language ought to have good business processing characteristics. Since COBOL was less popular in Europe than in the US, it was felt that the spread of COBOL could be curtailed this way. People close to academia and Algol argued in favor of an Algol-like block structures. Finally, the demands of internal users were for a system implementation language, with features such as storage control and exception handling. The proposal accepted for pointers and list processing originated from the afore-mentioned SLANG project [Laws67]. The biggest pressure was schedule. Originally, three months were allocated for this task (from October to December 1963). It took until February 1964 for the first draft to become available, and there were a number of subsequent iterations.

From the beginning, I had some (although minor) quarrels with the designers. Primarily, I did not like the luring compromise of including a little bit of something for everyone. I did not like if there was a FORTRAN way to do a certain function, it had to be there, but also the COBOL way of doing it, or that many features were included in the language that overlapped with basic functions of the then emerging operating systems (this applied to tasking and storage management in particular). Like many others, my pet proposal did not make it into the language either. I wanted to introduce scaled variables and units (Größen in German) in order to eliminate a total class of errors in many calculations. I still remember the answer I got: "Numbers as invented by mathematicians are abstract and beautiful. This should not be spoiled by engineers". This certainly saved the implementers a considerable amount of work. It also helps explain my prejudice against mathematicians and their contributions to computing.

IBM's European Development Laboratories

During the period 1950 to 1960, IBM established five development laboratories in Europe. Some of the historical background and the thinking guiding this process is described by Karl Ganzhorn [Ganz04], the founder and first director of the Boeblingen laboratory. Historically, the French and the German lab existed first, Hursley in England was next. IBM's Hursley laboratory is located just outside the town of Winchester, about 70 miles west of London. The central building of the lab is called Hursley House and has historical significance for the British.

Laboratory *(Year Founded)*	***Early HW Mission***	***Early SW Mission***	***Today's SW Mission***
Boeblingen (1953)	Processors, Printers, Memory Chips, Banking Equipment	Compilers, Operating Systems	Operating Systems, Data Mining
Hursley (1954)	Processors, DASD	Compilers	Transaction Systems
La Gaude (~1953)	Teleprocessing Controllers. Modems	Language Converters	(Networking)
Lidingo (1956)	Process Control, Data Collection Equipment	Real Time Monitor, Compilers	None
Uithoorn (1958)	Document Handling Systems	OCR, Operating Systems	None

Tab. 2: IBM European development laboratories

It is here where Oliver Cromwell's son lived in the 17th century and where the famous Spitfire, England's best known fighter plane, had been designed during World War II.

Fig. 4: IBM Hursley laboratory (Photo IBM)

Unlike the two older labs, Hursley worked from the beginning on both hardware and software. After some work on a FORTRAN interpreter, the lab in Hursley became IBM's premier compiler development center. Key developers were Tony Burbridge, Nobby Clarke, Jim Cox, Ray Larner, Brian Marks, John Nash, John Nicholls and Tom Peters. From 1965 to 1974, Hursley was also the home of what was called the corporate PL/I mission.

The IBM development laboratory in Boeblingen, Germany, is located about 20 km south of Stuttgart. It was founded as a hardware laboratory. Its software activities started about 1962 [Endr04]. From the beginning, the software development work was split between compiler development and operating system work. Over the years, both groups competed for resources, and it was the compiler group that eventually lost. While the operating systems found their justification largely as an adjunct to the Boeblingen hardware mission, the compiler group did not have that advantage. It was more or less a junior partner to Hursley's mission. The compiler work was initially led by Hans Gerstmann, with Hans-Jürgen Hoffmann and Werner Thiele as project managers.

Besides Hursley and Boeblingen, two other IBM development labs have to be mentioned in the context of language related products. The Swedish lab in Lidingo near Stockholm was only shortly involved. It completed, shipped and later maintained the OS/360 Algol compiler, described above. The French lab in La Gaude, near Nice, was involved first as the home of the 7090 Algol project and later for its responsibility for language conversion programs (LCP). This was a set of products intended to (partially) automate the conversion from existing languages, such as COBOL and FORTRAN, to PL/I. The Uithoorn lab in the Netherlands did not assume software responsibilities until much later. Although Vienna's role is well known, it was not considered as a development laboratory at this time as it did not have a product responsibility. The Zurich lab was always part of the IBM Research division. The

software development centres at Sindelfingen (Germany) and Croydon (England) with responsibilities for cross-industry applications are not considered here either.

The overall missions of the five labs mentioned in Tab. 2 were established in 1958 as reported by Ganzhorn [Ganz04]. Individual projects were negotiated during the annual operating plan cycle. As normally products were to be marketed worldwide, the respective development projects were funded by the corporation. Exceptions, like the Algol projects mentioned, could be funded by a region or area. Business cases had to be done on a regional basis. This means that any product had to be projected as profitable in each area in which it was to be announced.

Fig. 5: IBM Boeblingen laboratory (Photo IBM)

The most significant area was, of course, North America, but Europe, Asia and South America counted as well, although they were evaluated separately. A key function was played by the so-called product or systems managers. They were responsible for each product in their product or system family, starting from the initial conception to the end of its life. They were located at the laboratory site, and their responsibility may have comprised a pure hardware line of business, as in the case of printers, a pure software business, as for PL/I, or a union of both as in the case of the S/360 Model 20. All employees of a lab were hired by the local company and paid according to local regulations and standards. The five European development labs were originally managed by a directorate at the IBM World Trade Headquarter in New York City. It moved to downtown Nice in 1964. The software activities were first coordinated by Michael de V. Roberts, an Englishman, and later by Horst Remus, who had founded the software group in Boeblingen in 1962.

The majority of the people hired by IBM (or other computer manufacturers) in Europe during this period for work in systems programming possessed degrees in mathematics, engineering or physics. Others had backgrounds in economics, or in the humanities. All software developers had to be trained by the company in basic software skills and development methods. In IBM the basic training occurred locally, intermediate and advanced training was done centrally for Europe.

The First Generation of PL/I Compilers

IBM's investment in PL/I was quite significant, both planned and actual. As documented by Pugh et al. [Pugh91], initially four compilers were planned in 1964. According to the machine main storage size assumed, they were referred to as the C (8 kB), D (16 kB), F (64 kB) and H (256 kB) compilers. While the language responsibility and the implementation responsibility for the F and H compilers were assigned to the Hursley laboratory in England, the low end compilers were assigned to Boeblingen. After planning had proceeded, both the C and the H compilers were dropped, both for resource reasons and a lack of market demand.

The lead product was unquestionably the F compiler. It was the first implementation of the full PL/I language and ran in the IBM OS/360 environment. The storage space required by the compiler and its output was 44 KB, the same as for the Algol compiler described above. The compiler consisted of 15 phases, and was shipped in 1966. It met its target, namely to produce

code that ran as fast as similar program code produced by the best FORTRAN compilers of the time. Its optimizing approach benefited from work that had been done for the H compiler.

The D compiler was to compile and execute in the IBM TOS/360 (Tape Operating System/360) and DOS/360 (Disk Operating System/360) environments. The D compiler implemented a subset of the full PL/I language. The subset could not give up any of the key features that made the language attractive to both scientific and commercial users. In fact, one should be able to convert easily, if not mechanically translate, any existing FORTRAN or COBOL program into PL/I. Therefore, the many different data types, with their conversions and the file handling, had to be supported. Only features intended to ease the construction of large programs could be abandoned. Into this category fell dynamic arrays, data directed I/O, recursion, and tasking (the latter because of operating system limitations).

Product	***F Compiler***	***D Compiler***
Laboratory	Hursley	Boeblingen
Operating Systems	OS	TOS, DOS
Language Level	Full	Subset
Design Point (kB)	44	10
Size (kLOC)	100	120
Number of Phases	15	64
Shipment Year	1966	1967

Tab.3: First generation of PL/I compilers

The implementation of this compiler was constrained by its severe main storage space limitations, a problem that is of historic nature now. As mentioned before, the machine size to be supported was 16 kB. As 6 kB were taken up by the operating system, this resulted in a design point of 10 kB for the compiler. Since the same structure was to be used for the tape and the disk version, disk space was used in a limited, i.e. sequential, fashion. The secondary storage was specified either as four tape drives or four disk spaces (there could be multiple disk spaces on the same drive). One of the tapes or spaces was used as compiler residence, the other three as work files on which intermediate results were stored. Most of the library routines could be shared with the F compiler. The methods used for syntax analysis and code generation were state-of-the-art at the time. The entire code of the compiler was written in assembler language, with the aid of macros for certain repeating tasks [Thie69]. Besides Thiele, key designers were Karl-Heinz Dutke, Moniem Ismail, Dieter Jung, Helmut Kraft, Willi Plöchl, and Hermann Schmutz.

Two overwhelming problems confronted all PL/I developers at that time: language instability and performance. As a new language, many definitional problems existed, be it missing definitions, ambiguities, or desired changes. The implementation projects going on simultaneously in three laboratories (Boeblingen, Hursley, Poughkeepsie) produced hundreds of language issues to be resolved. Within one particular month, about 80 new language items arose. To achieve satisfactory object performance, more and more effort had to be spent at compile time. As a result, the first deliveries required lengthy compiles. This problem only went away when faster machines became available. The quality of the generated code was generally very good, however.

The Second Generation of PL/I Compilers

Following the F compiler, the Hursley lab developed a pair of second-generation compilers, called Optimizer and Checkout. They were shipped in 1971 and supported both the DOS/VS and the OS/360 environments. The Checkout compiler was a unique product and has been

externally documented by its designer [Mark72]. The PL/I source code was thoroughly analyzed and then converted to a 3-address internal format which was interpreted. It provided detailed error explanations at the statement level, fast recompiles, and extensive runtime debugging support.

Product	***Optimizer***	***Checkout***	***Mod20***
Laboratory	Hursley	Hursley	Boeblingen
Operating Systems	OS, DOS/VS	OS, DOS/VS	DPS
Language Level	Full	Full	Subset
Design Point (kB)	44	44	8
Size (kLOC)	60+	60+	100
Number of Phases	27	3	70
Shipment Year	1971	1971	1970

Tab. 4: Second generation of PL/I compilers

The Optimizer was expected to be run only when the program in question was considered to be debugged. The machine code produced could either be optimized with respect to storage consumption or run-time. Both compilers supported exactly the same language, allowing both to be used in combination. Also compiled routines could be mixed with new routines being debugged. Although written in assembly language, both compilers made extensive use of macros. This gave the compilers somewhat of a language-independent structure. As a result, parts of these compilers could later be used to build FORTRAN and COBOL compilers. The size number (kLOC) given in Tab. 4 for both Hursley products does not include the runtime library. It was largely re-used from the F compiler.

Function group	*Nr. of Phases*	*Detailed function*
Syntax Analysis	1	Parse input string; isolate names, keywords and operators; separately output DECLARE statements and executable source statements
Type Checking	1	Expand factored attributes in DECLARE statements; add undeclared items (e.g. intermediary results); relate names with dictionary entries
Code Generation	1	Enter all symbols into dictionary; convert executable statements into internal 3-address form used by interpreter

Tab. 5: Structure of OS/360 Checkout compiler [Mark72]

In 1970, the Boeblingen laboratory shipped a PL/I compiler for the System/360 Model 20 Disk Programming System. It had even less main storage (about 8 kB) and used the same multi-phase approach as the D compiler. The Model 20 (see Fig. 6) has been the first major success story of the Boeblingen lab. Its software support came in three stages. The initial configuration to be supported was the card programming system (CPS), a configuration with punched card I/O devices, but without a permanent residence device. For the configurations with magnetic tape and disk, these devices were used as program residence. The respective software packages were named Tape (TPS) and Disk Programming System (DPS), respectively. More details are given in [Endr04].

After the D compiler, work was done in Boeblingen on a conversational PL/I compiler, i.e. an incremental compiler for a timesharing environment. This product did not survive the design phase. A so-called Intermediate PL/I compiler, however, that compiled the full language

within a 32 kB design point, completed system test but was not shipped. Its market suddenly disappeared when advances in semiconductor memories prompted the company not to offer any System/370 configurations with less than 64 kB of main storage.

Fig. 6: System/360 Model 20 (Card system)

The second generation of PL/I compilers described above were the result of well-managed projects, which met their cost and schedule targets. They were priced separately from the underlying operating system as program products. The products were stable and achieved adequate and balanced performance. This was undoubtedly true for the two Hursley products. The Mod 20 PL/I compiler never became a real production tool. It essentially remained a teaching and demonstration toy, somewhat comparable to FORTRANSIT on the IBM 650.

The Impact and Fate of PL/I

Thanks to the above mentioned compilers, PL/I found its place across the entire range of System/360 users. There were several thousand customer installations throughout the world that used PL/I. The various compilers existed in the market for at least 10 to 15 years.

As a language, PL/I found more users in countries outside North America than inside, perhaps because of a larger investment in North America in FORTRAN and COBOL. Its acceptance was quite high both in Europe and Japan. Though PL/I was unable to meet its original goal, namely to supersede both FORTRAN and COBOL, it eventually became the most popular secondary language of both scientific and commercial users. Explained in simple terms, this meant: If a FORTRAN user needed better I/O handling, or a COBOL user wanted to do some arithmetic, they most likely switched to PL/I. This leads to some of the essential reasons for the failure of PL/I.

It was an illusion to believe that the two communities, business and scientific computing, can be united. They obviously feel more comfortable as separated camps that ignore (or look with disdain) at each other. With C, Pascal, Ada and BASIC (to name the most popular ones only) the divergence of languages continued, each addressing a different market segment. Only Ada's goal was comparable to IBM's intention with PL/I. Its requirements gathering and design process were much more formalized, however. Also the pressure exercised by all NATO purchasing departments on its suppliers helped to spread the language, but it is still a question whether it will survive much longer than PL/I. In the case of PL/I, the original idea was to define a comprehensive language first and then to create a subset for every unique group of users, all forming a true lattice. This great concept was never fully instantiated. What came closest to a teaching subset was implemented by Cornell University [Conw73]. Even a publication language that was more readable to humans did not come about.

As is well known, PL/I was never accepted by the academic community, at least in Europe. Here the argument "small is beautiful" has many followers. Of the many outspoken critics, I like to quote just one. At the NATO Software Engineering conference at Rome in 1970 [Buxt70], Friedrich L. Bauer made the following statement: "Some current languages are doing considerable harm in the educational process; one can't help mentioning PL/I here" That PL/I was not alone in this respect, can be concluded from the quip addition which Brian Randell made to Bauer's statement: " … and Algol 68". It would have been interesting to ask which features of PL/I had been meant: tasking/parallelism, different data types or exception handling? These may have been difficult concepts to teach then, but they were useful for students to know.

The Turning-Points in 1968 and 1969

The decade between 1960 and 1970 saw a first major struggle to understand the nature of software and to find ways to structure the needed supply. The decision to "unbundle" software from hardware, announced by IBM in summer of 1969, can be considered as a major turning-point in the history of our industry. An account of some of the deliberations leading to this decision is given by Watts Humphrey [Hump02]. The Garmisch NATO conference of October 1968, another turning point, was certainly influential for the academic side of computer science, but it did not have much relevance for the industry until such time as students with better training in software engineering methods appeared on the scene. That took at least 8 to 10 years. A systematic development process had been in place at various locations prior to 1968, particularly at the two labs mentioned here. The emphasis on software quality grew naturally from the fact that more systems were used in applications where failures and interruptions of service were critical for the business. It may have been an effect of the Garmisch conference that new methods were introduced faster, even if a thorough evaluation lagged behind.

Initially, most customers were reluctant to accept priced versions of software. They stayed with the old free versions longer than expected. New functions had to be significant enough to justify both the traditional migration cost and the new prices. In this new world, compilers had to be justified based on their software revenue. Initially, this brought about a reduction of the number of different compilers within IBM, but later on it entailed the disbanding of the compiler missions at the Boeblingen and Hursley labs. The related product responsibility was transferred to labs in the US. The corresponding skill deteriorated fast and was never re-established. This is highlighted by an observation communicated by a colleague from the IBM Research Division (Marty Hopkins). When he visited one of the labs to talk about the new compiler optimization techniques developed at Yorktown, only subcontractors were there to listen. No local IBM employees seemed interested. Since suddenly start-ups or small companies could compete in the language area, software groups at IBM concentrated in areas with more revenue and profit potential such as database systems and networking.

Other Influences and Events

After both Boeblingen and Hursley had abandoned their compiler missions, Boeblingen's software resources became totally absorbed by the work on the DOS/VS operating system, while Hursley took over the responsibility for the transaction processing system CICS. Both labs eventually got back into priced software components, however. New releases of both DOS/VS and CICS were gradually determined by the functions and features that were priced. To be legally on clear ground, all new code that was added to an existing module was separately identified. For both systems, this resulted in a collection of three types of modules: old unchanged ones, totally new modules, and modules containing both old and new code. As a side effect of the antitrust case, the US Federal Judge ordered all IBM locations to make

available to the courts all material generated as part of S/360 related development. This caused all European labs to transfer all documents and print-outs that normally would end up in the waste basket to a permanent storage. This certainly influenced our cost picture.

Looking back, the PL/I (mis)adventure was not the only one in the history of the company. At least three other projects lead to similar expenses and showed even less positive results. The best known one of the three was called Future Systems (FS). It tied up considerable hardware and software resources for several years but did not have any major effect on the mainline of the business. It resulted in a high-level architecture, however, for at least one family of systems, the System/38 and its successor the AS/400. This was not the case for two other major software undertakings, namely AD/Cycle and Office Vision. Although engaging some of the best skills for years, both in Europe and in the US, they have left hardly any traces in the company's business. As a consequence of these and other events, IBM's position in the market has changed dramatically.

Some Lessons Learned

A number of lessons can be drawn from the efforts undertaken before 1969. Although most of the items were controversial at the beginning, they appear as common sense today.

- *To strive for a single, all-encompassing programming language is an illusion.* It is the programmer's utopia. There is a strong indication that the different communities want to stay apart. They are in fact better served by different languages. This is a burden only for people crossing boundaries. Although systems programming has converged towards Java and C++, application programming is still split today between COBOL and FORTRAN adherents.
- *Serious users highly value their software investments.* Software that is being used represents an intellectual asset and enlarges the skills of the employees. This applies to applications as well as to development tools. Any migration is costly.
- *Software development is manageable, provided the goals are understood and the technology to be used is under control.* It is difficult to hit a moving or ill-defined target. New technologies require a learning curve. A top-down approach to new application areas is usually risky. Bottom-up approaches may lead to overlaps and inconsistencies, but can be managed with fewer problems.
- *Advances in hardware help and challenge software.* It is no question that the removal of storage limitations considerably eased the software task. On the other hand, new hardware facilities and capacities created new challenges. I constantly underestimated the progress of hardware technologies, although I had close contacts to some of the driving individuals and groups. I simply did not believe them.

There is a second group of lessons for which evidence became obvious later than 1969. I will mention them, nevertheless, because they are relevant to today's discussion.

- *Whoever constrains software to support just one vendor's hardware, misunderstands the nature of software.* Metaphorically speaking, it amounts to self-mutilation. All hardware vendors had to learn this. One of the reasons IBM lost the OS/2 versus Windows war was IBM's unwillingness to offer this system for other PC hardware.
- *Software products can be very profitable.* Several successful software companies are giving evidence for this. It is a unique business which has to be learned. High front end costs may become insignificant because low reproduction and distribution costs allow for the world wide installation of huge quantities. After initial development costs are recovered, selling more copies is almost pure profit.

- *The economic life of software exceeds that of hardware by a factor of 10.* Several products are more than 30 years in use now, supporting or tolerating the eighth or tenth hardware generation. The technical life is theoretically unlimited.
- *Product properties and timing are more important than productivity.* Software is considered a cost item unless it earns its own revenue. Over the years, one industry after the other moved away from individual solutions in the direction of standard products. Whenever development costs and productivity are of concern, this is merely a sign that this switch has not yet occurred. As in the book or film market, nobody will be concerned about the developers' productivity if the product is successful; nor does high productivity help if the product is a flop.

There are a number of people who prefer to consider software not as a product but as a service. This may be a valid business model for certain types of application. In a mass market, users and vendors are usually better off if their relationship is as loose as possible, i.e. the products are as self-contained as possible.

Final Remarks

Some of these lessons we had to learn the hard way. Today's decision makers have the advantage of being able to draw on the experience of others - provided they are willing to look at them. The concept of a systematic development process, denoted as "software engineering" later on, was part of Boeblingen's and Hursley's approach from the beginning. For us, it was never a question that high technical demands were placed on software developers.

I like to conclude with an episode on which I previously reported in [Endr04]. During the week of the 1968 Garmisch conference, IBM's director of software planning (the late Ted Climis) was visiting. I left Garmisch on Tuesday evening to see him in Boeblingen on Wednesday. When I told him what I was doing that week, he commented: "You are wasting your time. Academics cannot tell us practitioners anything about processes. This is true for chip design and manufacturing as well as for software development." Nevertheless, I returned to Garmisch on Wednesday evening. In retrospect, the academic interest in software processes has certainly helped to provide more focus on many issues related to it. In my view, it is best for both sides to talk to each other, to work together, and thus to reduce the gap between practice and theory. Computing is too important for positional wars.

Acknowledgement: I like to thank Tony Burbridge, Lucian Endicott, Karl Ganzhorn, Hans Gerstmann, Hans-Jürgen Hoffmann, Brian Marks, Horst Remus, Heinz Sagl, and Toru Takeshita for confirming, complementing or correcting my recollections in a number of instances.

Literature

Alle81 Allen, F.E.: The History of Language Processor Technology in IBM. IBM J. Res. Develop 23,5 (September 1981), 535-548

Back78 Backus, J.W.: The History of FORTRAN I, II, and III: ACM SIGPLAN Notices, 13,8 (1978); 165-180

Buxt70 Buxton, J.N., Randel, B. (eds.): Software Engineering Techniques. NATO Science Committee Rome conference 1970

Conw73 Conway, R.W., Morgan, H.L., Wagner, R.A., Wilcox, T.R. : Design and implementation of a diagnostic compiler for PL/I. Comm. ACM 16,3 (1973), 169-179

Endr04 Endres, A.: IBM Boeblingen's Early Software Contributions. IEEE Annals of the History of Computing 26,3 (July-September 2004), 31-41

Feld68 Feldman, J., Gries, D.: Translator Writing Systems. Comm ACM 11,2 (1968), 77-113

Ganz04 Ganzhorn, K.E.: The Buildup of the IBM Boeblingen Laboratory. IEEE Annals of the History of Computing 26,3 (July-September 2004), 4-19

Hemm86 Hemmes, D.A.: FORTRANSIT Recollections. IEEE Annals of the History of Computing 8,1 (January 1986), 70-73

Hoff98 Hoffmann, H-J.: The IBM OS/360 Algol 60 Compiler. http://www.informatik.tu-darmstadt.de/PU/docs/HJH-199804xx-Algol60.rtf

Hump02 Humphrey, W.S.: Software Unbundling: A Personal Perspective. IEEE Annals of the History of Computing 24,2 (March-June 2002), 59-63

Laws67 Lawson, H.W., Jr.: PL/I list processing. Comm. ACM 10,6 (1967), 358-367

Mark72 Marks, B.L.: Design of a Checkout Compiler. IBM Sys. J. 12,3 (1972), 315-327

Pugh91 Pugh, E.W., Johnson, L.R., Palmer, J.H.: IBM's 360 and Early 370 Systems. Cambridge, MA: MIT Press 1991

Radi78 Radin, G.: The Early History and Characteristics of PL/I. ACM SIGPLAN Notices 13,8 (August 1978)

Samm81 Sammet, J.E.: History of IBM's Technical Contributions to High Level Programming Languages. IBM J. Res. Develop 23,5 (September 1981), 520-534

Sibl61 Sibley, R.A.: The SLANG system, Comm. ACM 4,1 (1961), 75-84

Thie69 Thiele, W.: Die Entwicklung des PL/I-Übersetzers für das Platten-/Bandbetriebssystem. Elektronische Rechenanlagen 11 (1969), 25-35 (in German)

Trim01 Trimble, G.R.: A Brief History of Computing. Memoirs of Living on the Edge. IEEE Annals of the History of Computing 23,3 (July-September 2001), 44-59

Wolo86 Wolontis-Bell Interpreter. IEEE Annals of the History of Computing 8,1 (January 1986), 74-76