APL in the Nordic Countries

Gitte Christensen

APL in the Nordic Countries

Gitte Christensen

Dyalog Ltd, Hellebæk, Denmark
gc@dyalog.com

Abstract: An introduction to the early years of APL and its use in the Nordic countries. The different applications of APL are described in the context of changing conditions through the last 50 years. The evolution of APL is outlined and the current state established. In summary this is a story of computing driven by end-users.

Keywords: APL, History, Nordic Countries

1 The early History of APL

APL was invented by the Canadian mathematician Kenneth E. Iverson (1920-2004). As the name suggests, Iverson was of Norwegian descent. Born on a small farm in Alberta, he finished the one-room school after 9th grade and worked on his parents' farm, until he was drafted by the army in 1942.

After serving one year in the army, he transferred to the Royal Canadian Air Force and served as a flight engineer specializing in reconnaissance. During his service, he took advantage of the Canadian Legion's offer of correspondence education and almost finished high school. He enjoyed teaching his service mates mathematics, and when he left the service, it was with a promise to his counsellors and mates that he would pursue an academic career in Mathematics.

After his Masters degree in Mathematics from Harvard in 1951, he did doctoral work under Howard Aiken and Wassily Leontief - who later received the Nobel Prize in Economics. He worked on the mathematics for extending Leontief's input/output model, and as part of this work he wrote a matrix package for the Harvard Mark 1.

After completing his doctorate in 1954, Ken Iverson taught at Harvard for 6 years. During this period he became increasingly frustrated with the inadequacies of traditional mathematical notation in expressing algorithms, and he started to develop his own notation, which became known as Iverson Notation.

When he failed to get tenure at Harvard, he moved to IBM at the recommendation of Fred Brooks, with whom he collaborated on the continued development of his notation and in 1962 he published this work in the now-classic "A Programming Language". The notation Iverson had created was consistent and concise and dealt with matrices and higher order arrays. It had none of the precedence rules, which Iverson felt were the cause of many of the inconsistencies of traditional mathematical notation. See, e.g. [6].

```
      2 + 3                 Add two numbers
5
      1 2 3 × 1 10 100      Multiply two 3-element vectors
1 20 300
      mean←{(+/⍵)÷⍴⍵}       mean  is sum (+/⍵) divided by count (⍴⍵)
      mean ⍳4               Compute mean of the first four positive integers
2.5
```

Fig. 1. Simple APL expressions with output

A group at the Yorktown Heights IBM Research Center decided that this concise and consistent notation could be made executable on a computer. The first batch system written in FORTRAN for the IBM 7090 was quickly replaced by an interactive version for the new IBM/360, and this version served the researchers at Yorktown Heights Center for years. The first *cleanspace*, a template interactive session, was saved on November 27th 1966, which is considered the birthdate of APL. Ken Iverson became an IBM Fellow in 1970 and received ACM Turing award in 1979 [6]:

"For his pioneering effort in programming languages and mathematical notation resulting in what the computing field now knows as APL, for his contributions to the implementation of interactive systems, to educational uses of APL, and to programming language theory and practice."

2 Introduction of APL to the Nordic Countries

2.1 Denmark

In July 1967 Hans Jørgen Helms, director of the recently established NEUCC, held a two-week summer course on Programming Languages. It was extremely well attended, and all the major programming language research teams were represented. There were presentations and in some cases demonstrations of Snobol, Lisp, Algol98 – and from Yorktown Heights, Ken Iverson with APL. The interactive APL demonstration was performed using a model 2741 typewriter terminal, with a TV camera to make it visible to everyone.

It was still highly unusual to connect terminals to computers, but the University of Bergen had a system/360, and Dick Lathwell, who had worked on the implementation, was dispatched to Bergen together with a young IBM engineer, Henrik E. Nyegaard, to take care of the technical details. Per Gjerløv, also from IBM, who had been the driving force behind the donation of an IBM 7090 to NEUCC, was responsible for the technical set up – so he was naturally required to sit in on every single presentation. After many presentations of complicated programming languages Per felt that APL was a revelation. The simplicity with which programs were written in

APL made an immediate impression, and Per Gjerløv quickly became one of the most enthusiastic ambassadors for APL in Denmark, and he later wrote an excellent book on APL.

At around the same time ØK Data had announced a time-sharing service based on BASIC. IBM did not have a competing product lined up, and when the sales team heard about APL, things moved fast: a System/360 model 40 was installed at Slotsherrensvej in Copenhagen, and IBM was able to offer up to 30 simultaneous users access via regular phone lines – with reasonable response times.

Soon the municipal computing centre (Kommunernes EDB Central) also decided to offer a timesharing service. The director Johs Nielsen was a creative man: in addition to administration tools for the municipalities, he felt that APL should be made available to schools, so that pupils could experiment with mathematics – in the same way that they could do experimental physics or chemistry. ØK data also made a bid with the BASIC service. However, despite a large number of demonstrations and courses for teachers, APL was not adopted by the schools. Whether this was due to costs, or simply because the current thinking was that children should be taught to "program" rather than to solve problems on a computer, was never revealed [1].

At the Danish University of Technology (DTH, now DTU), professor O. I. Franksen and Peter Falster were very early adopters, and they taught APL to electrical engineering students. Many of these engineers subsequently embarked on careers as software developers and several are programming in APL to this day.

2.2 Sweden

In Sweden it was Sten Kallin, also of IBM, who became the big APL ambassador. Sten was a mathematician and had been impressed by one of the earliest uses of APL: a complete description of the architecture of the IBM system/360 using only a few pages of APL. He installed one of the very first versions of APL and was one of the select few who have installed APL from a deck of punch cards. He became extremely enthusiastic and taught many APL courses [4]. Volvo became interested and quickly installed an APL system, which engineers used to design and plan the production of entire cars, transmission units, and other parts. Volvo's Hans Dalqvist was a proponent and taught APL to many engineers.

Staffan Persson at Stockholms Handelshøgskola was an APL enthusiast with an interest in decision tables, and Professor Carl-Erik Fröberg at Lund also became a fan and wrote a textbook on APL.

2.3 Finland

In 1969 IBM was also responsible for the introduction of APL to Finland: Erkki Juvonen of IBM became the APL ambassador to Finland in the early 70's – together with his son Arto. However, the most influential person was probably Tauno Ylinen from the ministry of Finance, who was on the board of the Finnish Computing Centre, which provided services to governmental institutions. He was a great advocate of APL, and the computing centre ran a large number of APL courses under his leadership. Several Finnish universities taught APL, including the Helsinki University of Technology in Otaniemi, where associate professor Lasse Hyvärinen lectured on

"APL and its mathematical applications". Although Finland was a slow starter compared to Denmark and Sweden, Finland eventually became the Nordic country with the highest number of APL users per capita [3].

2.4 Norway

Ironically, despite Iverson's Norwegian roots, APL never really acquired a following in Norway. Possibly, the almost simultaneous creation of Simula by Ole-Johan Dahl and Kristen Nygaard at the Norwegian Computing Center had an impact.

3 How APL was Used

3.1 The 1970s - Timesharing

As mentioned above, most of the early uses of APL involved timesharing systems. IBM was naturally the first mover, but others quickly appeared on the scene. In Canada Ken Iverson and Roger Moore, who had worked on the first APL system for IBM, convinced Ian Sharp that APL timesharing was an exciting opportunity. In 1969, Scientific Time Sharing Corporation (STSC) was formed in the USA by other members of the APL implementation team, and an APL timesharing system was launched, hosted on I. P. Sharp Associates (IPSA)'s data centre in Toronto. Under the name APL*PLUS, the system added an "object store" for APL arrays, which allowed users to develop custom databases and other features needed by commercial applications. IPSA marketed the service in Canada and STSC in the USA until 1972, when STSC built a data centre of its own in the USA [5].

In 1973 IPSA started offering timesharing in London, UK. The connection to Toronto used a multiplexed 4800bps channel, which was shared by clients who dialled in with 150 baud modems. Initially, the service was so poor that successful printing of an entire report was a rare occasion! By 1975, mainframes had evolved "frond end processors", and IPSA laid the foundations for IPSANET, one of the worlds' first packet-switched networks. Customers made local calls to one of the IPSA offices, connected to each other via leased lines. IPSANET quickly grew to 20 "nodes", and later peaked at around 300 in the early 90's. In some countries where IPSA was not represented, large clients hosted the nodes [2].

It immediately became apparent that the network provided multi-national companies like Kodak and Xerox with an opportunity to input data to central data repositories, but IPSA also provided a MAILBOX system (written in APL, of course!), which provided clients and customers with global communications.

Clients typically developed their own software, representing their own business logic. There were virtually no software packages available, so not until 1973 did IBM start selling "Program Products". IPSA provided consultants to help clients develop software often heavily subsidised consulting, since application development was a bottleneck preventing the flow of timesharing revenue from new clients. As a result IPSA also accumulated a lot of knowledge about customer requirements, and by 1980 a number of applications were available "off the shelf": MABRA was an early database based on Codd's relational model from 1970. MAGIC was a domain specific

language for time-series analysis, with complementary products like MAGICSTORE, a multidimensional time-series database and SUPERPLOT for business graphics.

Timesharing customers were typically large companies involved in manufacturing or other forms of international trade. Only banks and insurers could justify having computers of their own, since computing quickly became the means of production for these companies. APL was particularly useful to insurance companies, since almost all the calculations that they need to perform are based on matrices. Until packages for insurance math became readily available, APL was on the curriculum for many actuaries.

Timesharing was also used by public organisations, using shared data centres. In Denmark, ATP, which was formed in 1964, had a unique challenge: Registering pension contributions from all employees in Denmark was no easy task, using the hardware and operating systems of that age: They solved the problem by implementing their own database system in Assembler and using APL for most other tasks: reporting, analyses, statistics and investment strategy.

3.2 The 1980s – APL ”In House”

In terms of market share, APL peaked in the eighties, with many companies using APL to manage their business. A variety of “midframes” became available, and many of them supported APL. Smaller and cheaper machines put pressure on IBM, who responded with the 4300 series, which was both cheap and reliable. Many clients took the opportunity to get off timesharing and run APL on their own computers.

IBM never made APL a Program Product, so it was not actively sold, but you could buy it. IPSA made SHARP APL available on the 4300 series, including all applications and connection to IPSANET so that worldwide subsidiaries could connect to headquarters as before. SHARP APL had extremely high performance and reliability, and the ability to perform backup/restore without taking the system down. STSC sold add-on file systems for IBM’s APL, but not the APL system as such. However, STSC was the first company to address the other big development in the market – the first IBM PC. In 1982, APL*PLUS was released and was an immediate success on the new PC’s.

In the eighties “shrink-wrapped” products appeared on the market: Accounting systems, word processors, database systems, and so on. Systems that had previously cost a small fortune to develop were now available at low cost for the PC or Unix workstations. However, large corporations were still running large, complex systems that could not immediately be replaced by these budding products, and there was a need to report on data collected from diverse sources. IBM provided Information Center products, to a large extent written in APL, which allowed large organisations to develop so-called User Service Centres.

In the early eighties IBM developed DB2. Oracle had released their RDBMS in 1979 and IBM needed a similar offering. Programming languages needed to evolve to match the new databases, and APL was no exception: Early APL could only support arrays containing either numbers or characters. In 1984, IBM introduced APL2, which supported nested arrays. Nested arrays allowed numbers and characters to be combined in the same array, which made it a good companion language for DB2. STSC, IPSA and many other vendors immediately followed suit.

3.3 Major Nordic APL Users in the 1980s

Most of the major banks in the Nordic countries used IBM mainframes for their production, capturing and storing all the various transactions which are the bread and butter for a bank. The databases where the data were kept were most often developed in-house and offered only the most rudimentary reporting facilities. With APL the users could access these data and analyse and report on them. IBM did offer a development tool which allowed the APL developers to make data from different sources available to the end-users who could then produce their own reports and calculations. Many of these applications had a striking similarity to the spreadsheets we all are used to today. The smaller banks organised themselves with joint data centres and developed their own reporting based on the repositories kept on the computers at the data centres.

Also insurance companies used APL to a large extent. Most actuarial calculations are table based and APL was a mandatory course at the actuarial studies and rumours has it that the language was named Actuarial Programming Language in those circles. Outside the financial sector Novo Nordisk used APL for pretty much everything. They had Sharp APL in-house on their 4300 and later on the larger 3090 machines with MVS and VM. They had a staff of a dozen APL programmers and used consultants extensively. They used MABRA (the SQL system from IPSA) for administrative purposes and, when the 3270 screens became available, its successor ViewPoint, which was an end user relational database and reporting tool which is, to this day, the most user friendly implementation of a SQL database the author has ever come across. They developed a clinical trial system, a batch control system, and their statistical libraries were renowned all over the world. They had implemented a sophisticated source code management system, which allowed them to share utility functions and basic functionality across developers and consultants and to version their software.

Under the management of the former airline captain Stig Eriksen, InWear managed planning, production and financial reporting using home-made software written in MIPS APL on PRIME minicomputers – which allowed them to become early adopters of just-in-time production.

B&O developed a management information system with interactive access from the boardroom and modelled the Beologic system in APL. In the transport sector, A. P. Møller used APL for management reporting and created Mærsk Data who provided APL consulting. DSB used APL for planning and engineering. During this period a number of consulting companies thrived on APL. In Denmark, SimCorp and IPSA were the leaders, with Datema and Mærsk Data also playing a role.

In Sweden Volvo continued their use of APL, and Sandvik built a design and production control system. Main consultancies were Cybex and Novator.

In Finland most governmental organisations based calculations on APL services from the Finnish Datacenter. Large corporations like Wärtsila had projects going on in APL. A Financial Statement Analysis application was created for Postipankki, which was downsized to PC in 1984 and is still in use by Finland's "Financial Times", Kauppalehti. Under Timo Seppäla, another Finnish proponent of APL, TMT-Team was a major consulting force, and also represented IPSA in Finland [3].

3.4 1990s – The Turn of the Tide

Microsoft Windows 3.1: The IBM PC was introduced in 1981, but it was the release of Microsoft Windows 3.1 in 1992 that made personal computing outside the mainframe "take off". Windows put word processing, spreadsheets and (as the internet started to spread) e-mail on every desk (and a few homes), and challenged APL on several fronts:

- Spreadsheets immediately provided a cheap, accessible alternative to APL (for simple applications), with built-in presentation capabilities.
- The new market encouraged the growth of new businesses which "commoditized" many of the solutions which APL had been used to prototype.
- The APL users, who had happily been doing personal computing for a decade or two on the mainframe and had been able to produce state-of-the art applications with unique business content, were suddenly struggling to manage their own hardware and deal with an explosion of technical complexity.

IBM's support for APL faltered: During the 1980s, APL's ability to deliver large amounts of CPU cycles and memory to analytical users had led to bad system performance on operating systems, which were core components of IBM's long-term plans. Most importantly on the systems designed for high volume transaction processing (CICS) and on early versions of TSO. Large parts of the IT community had decided that standardization on a single programming language for everything from systems programming to application development was a good idea, and C++ was the leading candidate. IT professionals firmly believed that end-users should stay away from application development, and often did what they could to sabotage any such activities. As a result, many APL applications were downsized to PCs or re-implemented. In many cases this resulted in a significant loss of functionality and flexibility, but the users typically had little influence on the decisions.

I. P. Sharp Associates was acquired by Reuters: Reuters was a major client, and purchased IPSA (vulnerable in the face of crumbling timesharing revenue) for its collection of historical time-series databases with global currency, commodity, and stock market prices, that IPSA had accumulated over the previous decade. During the same period Reuters had broadcast this data but neglected to collect it, and now PCs had created a market for technical analysis based on the data. However, Reuters had no interest in continuing to sell APL language products.

The removal of the two biggest corporate supporters of APL on the mainframe spelled the end of that era. Although quite a few systems are still running on mainframes today, most APL systems did move to PCs and workstations: costs were cut significantly, and APL consulting companies found themselves with a declining business. Having recognized the growing trend towards buying rather than developing software, many of these companies turned to software development themselves – and were able to use the experience gathered during the years of consulting to produce a number of excellent products. In the Nordic region, the best known examples of this are SimCorp A/S who developed a Treasury Management system known as SimCorp

Dimension, and Adaytum Ltd., who developed Adaytum Planning, a budgeting and planning application, in Denmark in the late 1990s.

In the USA, STSC had managed to develop a successful PC product during the 1980s and continued as a successful, if reduced, APL vendor following the collapse of the timesharing and mainframe markets. In the UK Dyadic Systems Ltd, a consulting company, also saw the writing on the wall and decided to build their own APL for Unix in 1983. Dyadic immediately went bankrupt following the project, but the re-constructed company is now a major player (under the name Dyalog Ltd.). Also in the UK, MicroAPL Ltd produced a version for the Apple Macintosh.

3.5 The 2000s

In the "naughties", the evolution of professional IT infrastructure and professional IT staff continued. However, it was eventually recognized that excluding subject matter experts from the development process leads to many failed, expensive projects. The bulk of the efforts of the IT department had been spent on storing and retrieving data, but the retrieval was only really possible if it had been predicted by the so-called architects. As a result, much of the information about the business - which existed somewhere within the vast amounts of data collected -- was left untouched, giving birth to the concept of the "Data Jail House".

It became clear that the software development methodologies adhered to by the so-called professionals were seriously flawed: Treating software development as a branch of engineering led to strictly componentized requirement gathering, analysis, design and implementation – with multiple communications gaps between the knowledgeable user and the programmer, who had to manifest the program logic. Big software projects always failed, were delayed or did not have the required functionality. The Standish Report documented that software projects with a labour cost of $1M had a success rate of 50%, and at $10M the success rate was zero.

In [7] *Extreme Programming Explained* (1999) Kent Beck describes *Extreme Programming* as a set of rules for agile implementation of software, which puts the user at the centre of the development process in order to ensure, that the programmer is constantly aware of the requirements, and receives continuous feedback. With some exceptions, the use of APL closely follows the rules outlined in the above-mentioned work; APL users would claim to have invented them, although many of them have not demonstrated the discipline that Mr. Beck would demand.

During the 2000s, APL continued to be used in many areas where the ability to combine domain expertise with rapid development provides competitive advantage. Areas like Financial Analysis, Budgeting, Asset Management, Risk Analysis, Business Intelligence, Big Data analysis, Medical applications and big industry business logic. In the middle of the decade Adaytum Planning was sold to Cognos for $160M.

4 The Evolution of APL

4.1 Language

Most modern APL interpreters will run APL written in 1966. However, a number of evolutionary steps have added very significant new language features. The biggest step was the development of APL2 by IBM (1983), which also created a major split in the community: Ken Iverson disagreed so strongly with certain design decisions made by the team headed by Dr. James A. Brown that he left IBM for I. P. Sharp Associates, where he was responsible for an alternative design for nested arrays. This work was the precursor for what was later to become a new language J, which Iverson designed and implemented with Roger Hui after he retired from IPSA in 1987. Other milestones in APL language evolution include:

1995: Control structures (if/then/else) adopted by several vendors.
1996: Optional lexical scope and lambda expressions in APL ("dfns" – Dyalog APL).
2006: Object orientation (Dyalog, MicroAPL, VisualAPL).
2014: Point-free or "tacit" syntax (from J) adopted by first APL vendor (Dyalog).
2014: Futures and isolates for parallel programming (Dyalog).

4.2 Environment

APL started on mainframes in an era where every aspect of computing was in its infancy. At that time the system needed to drive a 24x80 monochrome screen, a printer, plus random-access, sequential-access and possibly a keyed index file system. Vendors like STSC and SHARP added commercial report formatting, batch sorting and an "array object store" which allowed any APL array to be written to permanent storage. The environment remained essentially the same on the minicomputers that became important APL platforms in the late 1970s and early 1980s, there were APL systems for DEC, PRIME, Data General, and many more. Colour terminals and plotters with graphics capability and relational databases arrived on the scene, but the picture did not change much until the PC arrived.

As mentioned in section 3.4, the 1990s and early 2000s were a struggle for many APL users, as they were bombarded with a stream of new GUI APIs, security and other demands that made life as a combined domain-expert-and-programmer a challenge, following the serenity of the 70s and 80s. Vendors quickly focused on Microsoft Windows and created fairly usable Win32 GUI toolkits – but the API explosion in this dark period made it difficult to keep up, and APL lost significant market share.

The pace of hardware innovation has increased, with applications migrating to the web and to phones or tablets, and one might suspect that life for the APL user would therefore deteriorate. However, some of the technologies that have become widespread in the 2010s are helpful: through interfaces to Microsoft.NET, Java and HTML5/Javascript, APL vendors can once again provide APL users with usable interfaces for application development. The same interfaces make it straightforward to provide computational components written in APL as web servers or -services. Mod-

ern platforms support functional, dynamic languages and array-oriented interfaces based on reflection – so the environment is generally more APL-friendly today.

5 Where is APL Today?

Current APL systems are modern, dynamic, multi-paradigm programming languages with a focus on functional array-oriented programming. They support 64-bit addressing and unlimited array sizes, and thanks to Unicode the special symbols required in APL source files are no longer an issue. Modern APL remains a tool of thought for expressing ideas to a computer – or to other humans. It is most useful when the domain expertise of the user is important for the outcome of the development efforts, and where constant change is a condition. Given the complexity of modern computing environments, successful APL teams usually embed a few software engineering experts to provide assistance with interfacing tasks or other technical issues.

Examples of current uses of APL include:

The world's largest Patient Journal system in use is the TakeCare system used in Stockholms Län in Sweden. More than 2.5M patients and 45,000 users - 250,000 look-ups every day with sub-second response time.

APL-based research is being done in the emerging Bio Informatics area. In one example, APL's ability to handle large arrays helps researchers analyze genome data with an ultimate goal of creating personalized medicine. Another new APL-based application provides a visual data-mining tool for repositories of clinical data, to help identify correlations previously hidden in the data.

In Finland many governmental institutions still use APL for their analysis and reporting. Statistics Finland has created an end user application which helps organize statistical data for presentation. This application is part of an exchange of software between statistical bureaux worldwide and is used in more than 60 countries.

In South Africa an APL-based smartphone app allows small businesses and individuals to quickly forecast the viability of a business idea and its funding options.

Another Finnish application was the winner of the "Impress" category in the 2013 "apps4finland" competition: The Stormwind simulator is a stunningly realistic boat simulator intended to teach navigation and sailing – written in APL.

Asset Management and Risk Analysis will continue to be a sweet spot for APL, because financial markets are always in a state of flux. The value of being able to react to market changes with software, or model the impact of new legislation before it takes effect, is enormous – so companies like SimCorp will continue to use APL.

6 Summary

The story of APL is very much a story of end-user computing. The fact that APL was created with the purpose of teaching and communicating ideas, and that this quality was preserved during its implementation as an executable notation without detailing the operations of the computer, means that new ideas can easily be applied on a computer by the inventor.

The recent realisation that successful software implementation depends on close user involvement has been successfully proven by APL users through the last 50 years. Not only have the users been closely involved – they have been implementing their own solutions, and they will continue to spearhead new developments for the next 50 years.

References

1. Gjerløv, P.: Personal Communication (May 2014)
2. Wikipedia: I._P._Sharp_Associates. Copied from http://en.wikipedia.org/wiki/I._P._Sharp_Associates (May 2014)
3. Paavola, Olli V. M.: Personal Communication (May 2014)
4. Orrghen, A.: Sten Kallin. (2007). Copied from http://www.tekniskamuseet.se/download/18.6aa228912529fe96108000127/1339755638556/7_Sten_Kallin.pdf (2014)
5. Wikipedia: Scientific_ Time_Sharing_Corporation. Copied from http://en.wikipedia.org/wiki/Scientific_Time_Sharing_Corporation (May 2014)
6. Smillie, K.: amturing. Copied from http://amturing.acm.org/award_winners/iverson_9147499.cfm (July 2014)
7. Beck, K.: Extreme Programming Explained. Addison-Wesley, USA (1999).