The Expert Systems Business: How It Grew and Died

Paul Harmon, *Business Process Trends, Las Vegas, NV, 89130, USA*

This article describes the computer environment in the 1980s, the birth of the expert systems software business, the proliferation of companies created to sell expert systems products and services, their various products and business strategies, and finally their collapse and demise in the early 1990s. We describe the marketing strategies adopted by various vendors and the various market niches that evolved during the late 1980s and early 1990s. We offer speculation on the reasons for the growth and collapse of the expert systems market in the 1990s.

EXPERT SYSTEMS AS A NEW TECHNOLOGY

Expert systems applications were first developed at Stanford University in the 1970s. Other work with knowledge systems of various kinds was underway at other universities, but the development of knowledge-based systems and techniques that specifically sought to capture human expertise for focused problem solving were strongly associated with Stanford. The best-known example was MYCIN, an expert system designed to identify the bacteria causing specific meningitis infections and to recommend the correct dosages of antibiotics. The system was developed as a result of work by both physicians, who were experts in meningitis infections, and by Stanford computer scientists.

The computer scientists were artificial intelligence (AI) specialists who were focused on ways to make computer systems capable of more intelligent tasks. In essence, MYCIN was developed by capturing the rules that human experts used to diagnose meningitis infections. The software system allowed the developers to define as many rules as needed and used an "inference engine" (an algorithm) to sort through the rules, at runtime, to decide which rules were relevant to a specific case. The MYCIN application ended up with thousands of rules and tests proved the application was as effective as human experts in diagnosing meningitis problems and prescribing appropriate drug treatments [1].

1058-6180 © 2022 IEEE
Digital Object Identifier 10.1109/MAHC.2022.3149465
Date of publication 8 February 2022; date of current version 18 March 2022.

MYCIN was originally written in LISP, a computer language especially developed to support programming that involved list processing, which greatly facilitates rules processing, and logical reasoning tasks. LISP did not execute very efficiently on the commercial computers that existed in the 1970s or 1980s, but it made logic programming much easier.

The success of MYCIN, and similar early academic expert applications, led to an interest in commercializing efforts to build expert systems. Early initiatives took several forms:

1) Some companies were formed to commercialize versions of the LISP language. Prolog was another logic language that was popular among AI developers, and companies were formed to sell Prolog as well.
2) Some hardware vendors decided to develop tailored computers that would be specially designed to execute LISP or Prolog in an efficient manner. The Japanese Fifth Generation Project was a government-funded effort to develop a Prolog processing computer. The U.S. government contracted with Texas Instruments to develop a LISP machine.
3) Some companies were formed to develop software tools to aid programmers in the rapid development of expert systems. In effect, these products were shells built around inference engines that made it easier to capture specific knowledge and, thus, develop expert systems applications. One tool could support the development of many different expert systems applications. Tool vendors varied from those who

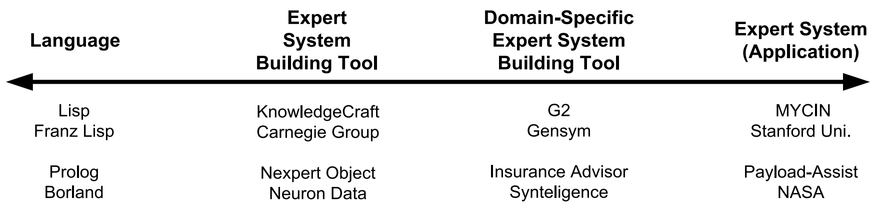

FIGURE 1. Expert systems product continuum.[2]

developed very complex tools that would allow for the development of very large, complex applications, to those that developed simpler tools that would support the development of smaller, less complex applications.

4) Still other vendors packaged an inference engine with some knowledge of a specific domain. This limited the tool to the development of a specific type of applications but made it much easier and faster to develop that specific type of application. A good example of a domain-specific tool would be a tool with general knowledge of banking and loans. Such a tool not only provided an inference engine but lots of common banking business rules that could be tailored. Using this domain-specific expert systems-building tool a bank could quickly develop an expert system for managing loan applications.

Figure 1 pictures a continuum that ranges from languages, with which one could develop any type of application, to tools that limited an application to the constraints imposed by the inference engine and any knowledge included with the tool. The continuum proceeds through domain-specific tools that include specific knowledge within the tool and concludes with an expert system application that includes both the inferencing capabilities and a complete knowledge base.

The expert systems market was complex from the very beginning, comprised as it was by a variety of different companies offering many different products. A company that decided it wanted to develop an expert system could decide to write the application from scratch, using a language like LISP, or it could opt to buy and use a tool. If one used a tool, one necessarily chose to build an application that depended on a specific inference engine. Depending on the size and complexity of the application being attempted, the company could elect to acquire a smaller, simpler expert systems-building tool, or they could opt to acquire a larger, more complex tool.

The entire expert systems market almost immediately became rather political when the Japanese government announced that it was going to launch a major initiative (termed *The Fifth Generation Project*) to build both specialized hardware and tools to support extensive expert systems development by Japanese companies. The Japanese effort was made more confusing when they announced that they would be supporting the Prolog language (an alternative to LISP more popular in Europe than in the U.S.). The Japanese announcement was quickly followed by a book, *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World*, written by Stanford computer scientist and expert systems guru, Dr. Edward Feigenbaum, and Pamela McCorduck, an AI historian. The book described the Japanese effort and positioned it as a major challenge to the U.S. economic position [3]. Recall that in the early 1980s, Japanese car companies had captured a significant portion of the U.S. auto market and U.S. business and government groups were very concerned about U.S. financial competitiveness. Feigenbaum's book attracted considerable attention from U.S. government agencies, and what had started as a shift in computer technology rapidly morphed into a major political issue that included calls from political and military figures for U.S. programs to prevent a Japanese takeover of the U.S. leadership in computing technology.

Suddenly, the development of a LISP-based computer and advances in expert systems became not just a business opportunity but a political imperative, and several U.S. government initiatives were launched to assure that the U.S. would prevail.

Meanwhile, companies suddenly began to debate the merits of LISP versus Prolog as a language for expert systems development, and tool vendors had to decide in which language they should build their tools. To add to the confusion, some new software tool vendors began to explore developing their tools in more conventional languages, like C, which would execute faster on conventionally available computer hardware. One could develop an expert system or tool in any sufficiently complex computer language, but some languages, like LISP and Prolog, made it easier to write an inference engine, but others, like COBOL or C, made it very difficult.

The academic organization that promoted AI in the U.S., the American Association for Artificial Intelligence (AAAI), had been holding modest conferences for years. Suddenly, in the early 1980s, as expert systems tool vendors were established, the AAAI conferences became large affairs with significant commercial exhibits. In addition, IEEE and other computer conference organizations became involved and there were a dozen conferences each year where people could go to learn about this new computing technology. In a similar way, computer and business magazines were filled with stories about how AI and expert systems were going to revolutionize the way businesses operated. By the mid-1980s the expert systems market was very hot!

KNOWLEDGE OF COMPUTING IN THE 1980s

To understand why expert systems seemed so exciting to computer people in the early 1980s, one needs to have a general understanding of commercial computing at that time. One of the things that characterized the software market in the 1980s was a lack of knowledge about the real nature of computing on the part of most commercial users. Digital computing is so pervasive today that it may be hard to realize, but knowledge of, and interest in, computing grew slowly over the course of its first 50 years. Digital computing arose toward the end of World War II, but only began to be widely used in business in the 1960s. (IBM and Univac began to sell computers to businesses in the mid-to-late 1950s.) As the interest in computers grew, companies hired bright people with backgrounds in various fields—there were, after all, no computer science courses offered in universities in the 1960s and only a few in the 1970s—and trained them to program in early computer languages, like COBOL and Fortran. At the same time, the principal computers these individuals worked on were mainframes. At first, data input was handled with punch cards, and then keyboards and terminals linked to the mainframes were used. Early programmers evolved to become the heads of computing departments in major companies, and many of them thought of computing and software development in rather narrow terms. Thus, in the 1980s, when computer scientists began to talk about commercializing AI and Expert Systems, they introduced ideas that were completely new to most of the people working in computer or data processing departments in large organizations.

Anyone who taught a course in expert systems technology in the early 1980s encountered questions that demonstrated a lack of computer knowledge on the part of even high-ranking computing executives. Many did not believe that computers could be programmed to reason and to reach uncertain conclusions. They were so familiar with lock-step algorithms that always reached the correct answer that anything less than certainty simply did not seem like real computing.

In a similar way, AI languages like LISP and Prolog were nearly incomprehensible to those who had always imagined that Fortran and COBOL pretty much-defined software development. The excitement about AI and expert systems was almost as if experienced computer professionals were being asked to go back to college and learn computer science all over again.

Herbert A Simon, one of the founders of AI, took his Ph.D. in political science and was awarded his Nobel Prize in Economics in 1978 for his work in Decision Making. Edward Feigenbaum, who studied under Simon, was initially hired by UC Berkeley in 1960 to teach in the Department of Business Administration. In 1965, Feigenbaum was hired at Stanford to be a part of its newly formed Department of Computer Science. We note this simply to remind everyone of how the early pioneers of computing began in other fields and only subsequently created computing science as an academic field.

EXPERT SYSTEMS AND KNOWLEDGE

An expert system was a software application that performed a task that would normally be performed by a human expert. To accomplish this, the software application incorporated a large amount of knowledge that was examined, in the course of problem solving, to reach a conclusion or prescription.

Knowledge is made up of statements (rules) that combine information to generate conclusions or suggest actions. Thus:

If there are more than 50 Oranges in a carton,

Then the carton should be shipped using FedEx. (Certainty = 100)

Statements of knowledge could be accompanied by different certainties. In the case above, we indicate 100 certainty, and thus, whenever this rule applies, its conclusion is certain.

Some knowledge was less certain. The MYCIN expert system was used to determine what type of meningitis a patient had. In most cases, the system suggested 3–4 different types of meningitis agents, each with a different degree of certainty. This was typically what human physicians did, and MYCIN then prescribed drugs to assure that the major possibilities would be covered by the proposed treatment. One

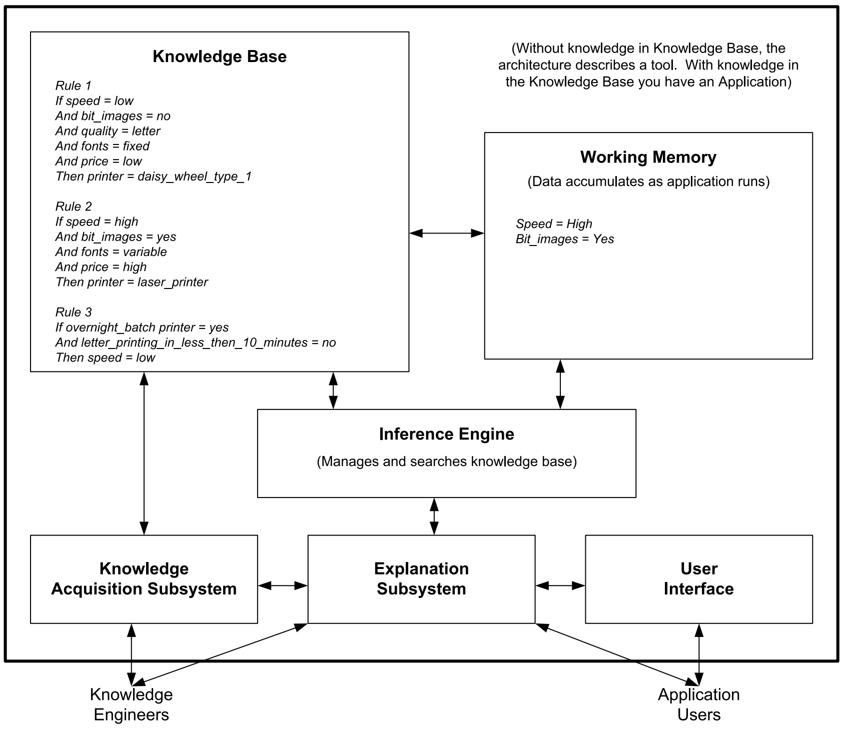
Knowledge Base
Rule 1
If speed = low
And bit_images = no
And quality = letter
And fonts = fixed
And price = low
Then printer = daisy_wheel_type_1
Rule 2
If speed = high
And bit_images = yes
And fonts = variable
And price = high
Then printer = laser_printer
Rule 3
If overnight_batch printer = yes
And letter_printing_in_less_then_10_minutes = no
Then speed = low
(Without knowledge in Knowledge Base, the architecture describes a tool. With knowledge in the Knowledge Base you have an Application)
Working Memory
(Data accumulates as application runs)
Speed = High
Bit_images = Yes
Inference Engine
(Manages and searches knowledge base)
Knowledge Acquisition Subsystem
Explanation Subsystem
User Interface
Knowledge Engineers
Application Users

FIGURE 2. Expert systems tool architecture.

could develop cultures and determine exactly what kind of meningitis one faced, but that took weeks and patients needed to be treated immediately, so physicians routinely "guessed" what they were dealing with, based on various rules-of-thumb (heuristics), and treated the worst possibilities while they waited for the cultures to be grown and for the specific meningitis agent to be definitively identified.

In other words, expert systems took computing into a new realm. It applied heuristic knowledge and reached probabilistic conclusions. The whole idea that computers could deliver uncertain judgments was unsettling to many when they first encountered expert systems. Of course, uncertain judgments were exactly what human experts delivered, and the expert systems were designed to perform similar tasks.

Figure 2 pictures an architecture for a simple expert system. Such a system had a database (usually called a knowledge base) in which it stored rules. These rules were developed by programmers (called knowledge engineers) who worked with human experts to determine what rules of thumb the human experts used when they examined actual cases. Complex problems often required hundreds or thousands of rules and the process of examining cases and formalizing rules could take many months. Each expert system also included an inference engine, an AI algorithm that defined how the tool would search the knowledge base looking for answers.

In addition to a knowledge base and an inference engine, each expert systems-building tool included a working memory space in which it maintained facts it had ascertained. In some cases, data was input. In other cases, users were asked questions and their answers placed in working memory. For example, a clerk might have been asked how much a package weighs. Let us say the customer answered, "59 pounds." This fact was placed in working memory. Then, when an inference engine ran and examined the rule base, if it found a rule that asserted that "If a package weighs 50 or more pounds, then the package should be sent via FedEx," then it would enter a new fact into working memory: The package should be sent via FedEx.

An inference engine examining a knowledge base was an exercise in applied logic. Different types of inference engines were used for different types of problems. One type of inferencing worked from facts to conclusions (forward chaining). This was an ideal approach if you were trying to configure a complex computer system. In effect, one asked a customer to tell you want elements he or she wanted in their computer system, and then the inference engine used rules to determine if a certain part were desired, what other items would be required. Conversely, a backward chaining inference engine was best for medical diagnosis. In effect, one started with a patient describing symptoms. The system then hypothesized a given type of meningitis and searched for rules to support the hypothesis. In effect, a backward chaining system runs through possible conclusions, eliminating those that could not apply to the specific case.

Most expert systems-building tools also included an explanation subsystem. Using this utility, a user or a knowledge engineer could ask a system how it had reached a specific conclusion. In response, the tool would provide the logic chain, showing which rules had fired and which facts had been placed in working memory. This utility provided users with increased confidence in the conclusions being obtained from the system.

One could create an expert systems tool by taking an existing application, like MYCIN, and removing the knowledge. Thus, the user got a working inference engine and all the utilities to help input data and run consultations. To generate a new expert system application, one simply needed to insert new knowledge (e.g., new rules).

Stepping back a bit, one could develop an expert system by building every element of it, from scratch, in a language, like LISP or C. One could work with a package that included a lot of predevelopment utilities, one could work with a tool that had everything one needed, save specific knowledge. Or, if you knew you wanted to develop a specific type of application, say an expert system to monitor and manage an oil refinery, you could acquire a tool that came with an interface and knowledge rules about oil refineries. This latter approach was often referred to as a domain-specific approach, since, in effect, you bought a nearly complete oil refinery application and then simply tailored it to work with your specific oil refinery.

Over a decade, companies tried all of these solutions and offered products configured in a wide variety of ways—each seeking a product that would best satisfy their customers.

The original companies were founded by academic experts and focused on offering expert systems tools derived from academic experiments. These tools were designed to develop applications that functioned like world-class experts. This type of expert application would typically save a company millions of dollars over the course of a few years, and hence the vendors could charge a premium price for their products.

As companies began to learn more about expert systems technologies, however, they identified all kinds of knowledge-based tasks that inference-based techniques could be used with, some quite unexpected. Some organizations had small problems that only required a little expertise, but the problems were still important and worth an effort if the solution to a problem could be automated. As this became obvious, new vendors entered the market offering tools that could be used for smaller "knowledge" tasks that were not really major expert systems tasks. Thus, a market was created for knowledge-based "job aids" [3]. These smaller PC-based tools were still often referred to as expert systems, and this further confused the market.

EXPERT SYSTEMS COMPANIES

Expert systems companies began to appear in the early 1980s, and, as the computer press became excited about the possibilities of expert systems, they proliferated rapidly. There was a lot of confusion about the various companies and what they offered and anyone seeking to understand the market needed to keep some important distinctions in mind. In trying to describe the marketplace for expert systems software products, the newsletter, *Expert Systems Strategies*, gradually evolved into a six-part classification.

1) High-end LISP Tools. (e.g., Carnegie Group, Intellicorp, Teknowledge)
2) Domain-specific High-end Tools. (e.g., Syntelligence, Gensym Corp., Carnegie Group)
3) High-end Mainframe Tools. (e.g., Aion, AICorp, IBM)
4) More Powerful Workstation and PC Tools. (LISP, C) (e.g., DEC, TI, Gold Hill, Neuron Data)
5) Simpler PC-based Expert Systems Tools. (Pascal, C) (e.g., Level Five, Exsys, Paperback Software)
6) AI Language Vendors: (e.g., LISP: Gold Hill, Franz; Prolog: Borland)

One of the problems in generating a retrospective survey of the expert systems tools market is that it changed so rapidly over the course of ten years. The earliest vendors were the High-end LISP Tool vendors that were founded by teams of university researchers. Within a couple of years, however, other vendors were established offering different tools—often tools for what were then newly evolving personal computers (PCs), tools written in languages other than LISP, tools for mainframes, etc. As each new class of tool was offered, existing vendors would adjust their own offerings, change prices, and emphasize live different features. Within two years of offering S.1, a LISP-based tool designed to run on DEC workstations, Teknowledge (a company founded by Stanford researchers) developed a PC offering written in Prolog and called M.1. They initially positioned M.1 as a learning tool, but then repositioned it as a tool for the development of small expert systems, which were increasingly called knowledge systems.

In describing the software vendors, we will limit the list to a few of the major players in each of the categories and will describe them as they originally appeared in the marketplace. We will briefly consider the evolution of various offerings later in this article but will not attempt to describe all the mergers, versions, price changes, or repositioning efforts that occurred over the course of a decade.

HIGH-END LISP TOOLS

The earliest expert systems software products were developed by companies that were established by researchers and offered products that were spinoffs of university expert systems research projects. For example, two of the first expert systems were developed at Stanford University (e.g., Dendral and MYCIN). A little later, a group of Stanford University professors with entrepreneurs from Silicon Valley formed Intelligenetics and Teknowledge. Intelligenetics was initially expected to develop medical applications, like Dendral, while Teknowledge planned on selling a generic version of the software package that had been used to develop MYCIN. In essence, by stripping out the knowledge of meningitis, the Teknowledge team arrived at a software package, called S.1, into which others could insert new knowledge to create a new expert system. In effect, a company buying Teknowledge's S.1 was acquiring much of the software code required to build an expert system and the company only had to focus on acquiring knowledge from a human expert in a new field.

A good example of an application developed in S.1 was an Oil Prospecting expert system developed by Schlumberger. In essence, the oil company worked with employees who were already studying various forms of geological data and predicting where new wells should be drilled. The expert system encoded the rules these human experts used to analyze the seismic data and make predictions. The resulting expert system, running on MYCIN's old inference engine, analyzed seismic data input directly into the expert system from seismic recording devices and made predictions as to where test oil wells should be drilled.

It is worth noting that Teknowledge used an approach that was very similar to the one used at Stanford when the original MYCIN was developed. They initially built the S.1 tool in LISP and designed it to run on the same DEC computers that Stanford's computer scientists used. Offering this package to commercial companies meant that their customers would have to proceed up a steep learning curve, training programmers in the LISP language and in all the techniques used to develop new software on DEC workstations. It also meant that the companies would have to learn to transfer their data from mainframes, where it resided, to the DEC workstations for analysis and then move results back to mainframes for subsequent distribution. Recall that this was taking place in the 1980s, well before common standards for data transfer were widely available, and hence all these transfers needed to be hand-coded. The promises of powerful expert systems were sufficient to tempt some large corporations to invest in developing teams of expert systems programmers (usually starting by hiring newly minted AI graduates from major universities who were already versed in LISP development), but it made the effort very expensive, and out of the reach for most business IT departments.

The first group of expert systems companies in the U.S. all depended on LISP-based software tools derived from earlier university research in AI development. The vendors all asked a premium price for their products, required a very significant investment in learning, and significant development efforts. Of necessity, this investment was only justified by projects that would solve significant problems and generate major income for those involved.

The following companies are examples of vendors who offered High-end LISP Tools.

Intelligenetics/Intelicorp. Tool: KEE. Initial purchase price: $9,000–$98,000. Platforms: Symbolics, DEC, Sun, and HP workstations running Unix and LISP. Inteligenetics was founded in 1980 to sell a domain-specific tool or applications, but quickly decided that it would be better to sell a generic tool, and renamed itself Intellicorp. Where Teknowledge initially only focused on representing knowledge as rules, exactly as they had in academic applications like MYCIN, Intellicorp put more effort into representing some of the knowledge as a network of objects. Initially, most AI people spoke of object-oriented programming environments as frame-based systems but gradually switched when "objects" proved the more popular term. Much of the research used in developing object knowledge representation was derived from work at Xerox PARC, which, at this time, also proved a source of graphic environments that migrated into the Apple Lisa, and then into the Macintosh. In essence, a "hybrid" expert systems tool represented some of its knowledge as rules and other bits of knowledge as object network (or cognitive network) that could handle more complex knowledge and hierarchical relationships more efficiently and graphically than simpler rule-based systems. As time passed, most of the leading LISP-oriented vendors switched to hybrid environments, which in turn introduced a sharp difference between the simpler, rule-based development environments and the hybrid tools with both rules and object-oriented ways of representing knowledge.

Teknowledge. Tool: S.1. Initial purchase price: Tens of thousands of dollars. Platforms: DEC, Sun, and HP workstations running Unix and LISP. Teknowledge was founded in 1981. Like Intelligenetics/Intellicorp, it was started by researchers from Stanford University (including Edward Feigenbaum) who had developed the first academic expert systems (Dendral, MYCIN). Like many startups from academia, Teknowledge's founders could be a little over-confident, feeling that they had, after all, invented the field, and knew what was important and what was not. Teknowledge spent about a year working hard to educate the press, government and business about AI and expert systems. They were surprised to find that others felt free to jump into the market and offer tools on mainframes and PCs, or use expert systems techniques to develop different kinds of applications.

Carnegie Group. Tool: KnowledgeCraft. Initial purchase price: $10,000 to $70,000. Platforms: DEC, Sun, and HP workstations running Unix and LISP. Carnegie Group was established by a group of researchers who originated at Carnegie-Mellon University. While the West Coast expert systems companies derived their initial stimuli from Stanford and tended to focus on backward chaining approaches, the East Coast companies were more beholden on OPS5 (Official Production System, version 5) and other academic experiments that favored forward chaining, which introduced still more complexities into the market.

Inference. Tool: ART. Initial price: Tens of thousands of dollars. Platforms: DEC, Sun, and HP workstations running Unix and LISP. Formed in 1979 by Chuck Williams, a researcher from Cal Tech, Inference also mixed rules and an object network to support complex knowledge representation.

DOMAIN-SPECIFIC HIGH-END TOOLS

Almost as quickly as the first generic expert-systems building tools appeared on the market, other companies were founded to offer domain-specific tools. In essence, consultants offered to build expert systems applications for companies, and then, having built one or two, went on to generalize their work and create tools that had all the generic capabilities of the high-end tools, and retained some of the specific knowledge and specialized adaptations they had made to make them particularly good at solving a specific type of application. These tools were only marketed to companies within specific industries, or for specific types of applications. Obviously, the market for domain-specific tools was more limited than generic expert systems tools, but domain-specific tools could be developed and installed much faster and justified higher prices.

Syntelligence, Inc. Tools: Underwriting Advisor, Lending Advisor. Initial purchase price: Tens of thousands of dollars. Development platforms: DEC, Sun, and HP workstations. Syntelligence developed a LISP environment and focused on developing products that could be tailored by financial institutions. Thus, its Lending Advisor, and its Underwriting Advisor were designed to be quickly tailored by banks and insurance companies, respectively. Each included considerable knowledge of the task and only required the purchaser to add rules specific to the institution. Syntelligence worked hard to make it possible for their applications to work in mainframe environments, which dominated both financial and insurance companies in the 1980s.

GensymCorp. Tool: G2 Real Time Expert System. Initial purchase price $18,000. Platforms: DEC, Sun, and HP workstations running Unix and LISP. This tool was designed to build expert systems for real-time industrial process-control applications. A popular use was to model oil refineries or steel mills. The tool would allow the user to build a graphic model of the refinery or process, indicate real-time monitoring points, and write rules to determine what to do if monitored values went outside set parameters. All the graphics required to model an oil refinery were included in the tool, so creating a model was as simple as assembling the model of the plant. In a similar way, one could click on a graphic element—say a valve—and indicate that a monitor would be attached there. Rules were written to specify what actions would be taken when monitored data reached or exceeded certain set-points.

Carnegie Group offered a series of application-specific tools. Testbench was a machine diagnosis tool offering a combination of rules and failure-mode analysis. It also offered LanguageCraft, a natural language understanding tool that customers could use to build their own natural language interface. LanguageCraft was one of the first natural language tools in the market.

Most domain-specific tools were high-end tools, but as time passed, smaller vendors marketed smaller domain-specific tools which they derived from applications built with their products. Level 5 (Later Information Builders), for example, offered a domain-specific tool designed to develop COCOMO (Constructive Cost Model) cost-modeling applications.

HIGH-END MAINFRAME TOOLS

The initial expert systems building tools were all workstation-based tools written in LISP and running on Unix operating systems. But, at that time, the computer systems actually used at most businesses and in government were mainframes. A few companies had workstations, mostly focused on industrial applications. The first IBM PC was introduced in 1981, and most companies did not begin to use PCs until the late 1980s. Remember that the Internet and standard communication protocols were not in widespread use until the 1990s, and that in the mid-1980s it was hard to create applications that passed application data from one operating system environment or software language to another. To make expert systems development easier for corporate IT groups, some startups were founded by entrepreneurs with mainframe backgrounds focused on developing expert systems tools in languages like C that were designed to run on mainframe operating systems. This required a significant vendor development effort, since mainframe environments were not originally designed to support the online memory requirements that LISP environments and workstations did.

Aion. Tool: Aion Development Environment. Price: $7,000–$97,500. Platform: S/370 (MVS/TSO). Written in Pascal, COBOL, and C. Aion, in California, and AI Corp, in Massachusetts, were rivals for the most powerful mainframe tool and eventually merged. Since both products significantly reduced the effort required to get started and to field an expert system, both vendors received quite a bit of attention. Both vendors struggled to work around the limitations that mainframe environments placed on AI functionality, and both eventually succeeded.

AI Corp. Tool: KBMS Price: Tens of thousands of dollars. Platform: S/370 (MVS/TSO) Pascal, COBOL, and C.

IBM. Tool: TIRS. Price: $11,000–$60,000. Platform: MVS/CICS, MVS/TSO, MV/CMS. C. In time IBM offered its own product, but never managed to become a major player in the AI mainframe tools market.

MORE POWERFUL WORKSTATION AND PC TOOLS

As the IBM PC and the Apple Mac came into widespread use in the latter half of the 1980s, lots of startups formed to offer expert systems-building tools for the PC. Recall that the original IBM PC was based on an Intel 286 chip and thus had very limited memory or speed. The early Macintosh was a little better, but not by much. Offering anything substantial on a PC was a major challenge. Most leading PC vendors eventually also developed a version of their tools that would run on workstations from DEC or Sun.

Gold Hill Computers. Tool: GoldWorks. Price: $7,500. Platform: 286 PC, Mac, Sun (LISP).

Gold Hill was closer to the original expert systems vendors than any other early company offering a tool on a PC. The company was founded by MIT researchers and originally offered a LISP development environment for PCs. It then proceeded to develop and sell an expert systems-building tool written for its own version of LISP. Most people who tried to use Gold Hill opted for an add-on board that could upgrade a 286 PC to a 386 machine and add quite a bit of memory, but that only resulted in a limited improvement. Gold Hill products sold well in university environments, but always met quite a bit of resistance in business environments.

Neuron Data. Tool: Nexpert Object. Price: $5,000–$8,000. Platform: Apple Mac and C.

Neuron Data was developed by two French graduate students and was a real breakthrough when it was first offered. Until then all the PC expert systems tools were written for workstations, or used a Microsoft OS. Neuron Data showed that one could have a rich graphic environment on a Mac, and, using C, could have a fast execution environment. Nexpert Object became the expert systems building tool "for the rest of us" and always got ambiguously positioned between high-end tools (in LISP) and PC tools.

Texas Instruments. Tool: Personal Consultant. Price: Thousands of dollars. Platform: TI PC and IBM PC and C. The U.S. military became very interested in expert systems and contracted with Texas Instruments (TI) to develop a computer designed to execute LISP. In essence, it would be an analog computer that would provide features that would make LISP execute efficiently. TI also received several U.S. government contracts to develop major expert systems applications for the military. In the process, they developed Personal Consultant, a rule-based expert systems-building tool designed to run on a Texas Instrument's PC, and later an IBM PC.

Teknowledge. Tool: M.1 Price: Thousands of dollars. Platform: IBM PC and Prolog.

M.1 was developed by Teknowledge as a way of countering all the companies that began to eat into Teknowledge's market with PC offerings. M.1 was written in Prolog and could do most of the things that S.1 could do. Thus, in fact, M.1 cut into Teknowledge's own high-end tool market.

SIMPLER PC-BASED EXPERT SYSTEMS TOOLS

Level 5 (Information Builders, Inc.). Tool: Level 5. Price $685. Platform: IBM PC, C.

One of the first companies to develop an expert systems building tool for the PC was Level 5 which was created by two brothers from Florida who came from a NASA background. Their tool was strictly limited in what it could do but did it very efficiently on a PC and proved very popular for people who just wanted to get a quick introduction to rules-based programming. Level 5 was quickly acquired by Information Builders, another PC vendor who proceeded to port the tool to workstations, and then tried to port it to mainframes.

WHEN EXPERT SYSTEMS BECAME A HOT TOPIC, AND PEOPLE BEGAN TO RESEARCH THE ORIGINS OF THESE APPLICATIONS, IT SEEMED AS IF YOU NEEDED TO LEARN LISP OR PROLOG TO DEVELOP THIS TYPE OF APPLICATION.

Level 5 proved very popular for those who wanted to build small knowledge-based applications. Each application might only have a few hundred rules and solve a very specific problem, but, by allowing lots of people who could otherwise not program to quickly assemble an application that would solve a specific problem, they let companies generate quite a bit of automation really quickly. This approach was, of course, nothing like what those who created the first expert systems had in mind. It empowered technicians to automate specific activities, usually in support of employees who would consult the system on an as-needed basis. Nevertheless some companies like DuPont invested heavily in this strategy and developed hundreds of small knowledge-systems in the late 1980s.

Exsys. Tool: Exsys Professional. Price $795–$12,000. Platform: IBM PC, Sun Workstations. Pascal. Another of the many popular PC tools, the tool was written in Pascal, ran primarily on PCs, and offered a basic rule-based development environment.

Paperback Software. Tool: VP-Expert. Price: $95. Platform: IBM PC. C. VP-Expert came in the 1990s, was developed by a computer science student at the University of California at Berkeley, and offered as much power as any of the other simpler rule-based PC tools, at a bargain price. This tool illustrates the problem that any software vendor faces. Once ideas are out in the public domain, someone will decide to create a very cheap and reasonably effective version that will significantly reduce the perceived value of the product. Borland did the same thing for AI languages at about the same time. Few companies used VP-Expert to develop serious knowledge systems, but lots of students used it as a cheap way to learn about expert systems development.

AI LANGUAGE VENDORS

AI applications rely on large amounts of working memory, on logical operations, and on the ability to process rules in a rapid manner. These are not features that are easily supported by conventional computer languages like Fortran, COBOL, Pascal, or C, since they were initially developed for the business and scientific communities. AI researchers had developed two languages that were designed for AI work, LISP (List Processing) and Prolog (Programming in Logic). LISP and Prolog, in turn, were not especially efficient for high-speed arithmetic calculations or conventional data processing. All these languages were, of course, Turing languages and could all support any basic computer operations. Some, however, were more efficient and easier to configure for one type of application, while others were better for writing different types of applications.

LISP was the most popular AI language in the U.S., while Prolog was more popular in Europe and was adopted by the Japanese government for its Fifth Generation Project. Both Japan and the U.S. decided to develop hardware that would be more effective in processing Prolog and LISP, respectively.

When expert systems became a hot topic, and people began to research the origins of these applications, it seemed as if you needed to learn LISP or Prolog to develop this type of application. Companies that wanted to develop a comprehensive capacity in expert systems development often bought LISP or Prolog and trained programmers to create applications in these languages. Similarly, companies that hired graduate students with degrees in AI invariably found that these individuals preferred to work in AI languages.

Many versions of AI languages had been developed in universities and were available without charge. Vendors also offered AI languages but were limited as to how much they could charge. Nevertheless, some language vendors began to sell AI languages to business IT groups.

Gold Hill, DEC, and Franz are among the many companies that sold LISP to businesses. Borland, a company founded by a Frenchman, offered a very inexpensive Prolog development environment. As time passed, several other language vendors entered the market, but none of them ever had much impact on the overall expert systems tool market. Most companies preferred to buy tools that already had utilities they needed to build expert systems. Most companies that used AI languages acquired them when they bought expert systems-building tools written in one or another AI language. More important, as time passed, most companies began to be more interested in tools written in more conventional computing language.

TI and the Japanese Fifth Generation project both, eventually, produced LISP and Prolog chips for computers designed to execute applications written in those languages. By the time they did, however, expert systems were already in decline, and, in the meantime, Moore's Law had resulted in conventional, digital PC chips (e.g., Intel 586) that were powerful enough to execute AI applications without too much difficulty and for a much more modest investment.

The companies named so far were all U.S.-based companies, who were most active in the commercial development of expert applications. This is not to suggest that companies in Europe and Japan did not explore expert systems, but they did it in a more cautious manner. Just as there were tool vendors in the U. S., there were tool vendors in Japan and in Europe, but they were mostly smaller. In the mid-1980s, it was often easier for French expert systems tool vendors to sell tools in the U.S. than it was to sell them in Germany. As the European market developed barriers have come down, but, at least in the mid-1980s, very few of the European expert systems vendors became very large. Most, in fact, were really consulting companies that found a niche, say banking, developed a solution for one client, then created a domain-tool from the core of their solution and tried to market it to other banks. This approach and the limited size of the market in the individual European countries meant that European companies were usually working on the first or second generation of their product while U.S. firms were introducing the 3rd or 4th generation of their product.

CONSOLIDATION OF THE MARKET

The market for expert systems software products began in the early 1980s. Initially, there were a number of startups. As the startups received press attention and seemed to be making money, other companies entered the market. Some tried to do what existing companies were already doing, only better. Most tried to find a new niche and offered specialized products. By the late 1980s, some of the larger software vendors, like IBM and DEC had become heavily involved in the market. Some, like Texas Instruments, preceded to develop their own offerings. Others, like IBM, sought to acquire existing companies and then improve the products acquired. There was some market consolidation, but it was not too obvious because the companies being acquired were largely balanced by new startups entering the market, thus the number of vendors in the market appeared to remain about the same. Other large vendors, like Microsoft, never offered a proprietary expert systems product, feeling that the market, as a whole, was too small for them.

MOORE'S CURVE AND NEW TECHNOLOGIES

As we studied the expert systems software market in the 1980s, we tried to conceptualize the development in various ways. In 1991, Geoffrey Moore, a high-tech marketing guru who had been involved in numerous different technology launches, published a book—*Crossing the Chasm*, (HarperBusiness, 1991)—that seemed to provide the best explanation of overall development of the expert systems market that we encountered [5]. Moore's model focuses on the types of companies that try to implement new technologies. In terms of the expert systems market, these are the companies who bought expert systems software tools and built the initial expert systems applications.

Moore's model postulated a series of phases that any new technology must go through.

Innovators

New technologies are initially adopted by Innovators, companies that are focused on the new approaches and are willing to work hard to make a new technology work in order to gain an early advantage. Innovators have their own teams of sophisticated technologists and are willing to work with academics and vendors to create highly tailored solutions.

Early Adopters

Once the Innovators prove that a new technology can be made to work, Early Adopters follow. Early Adopters are not focused on new technologies, as such, but on new business approaches that can give them a competitive advantage. They are less technologically sophisticated than Innovators, but still willing to work hard to make a new technology perform, if they see a clear business advantage. (See Figure 3.)

Early Majority

The market for a new technology does not really get hot until the Early Majority are convinced to adopt the technology. The Early Majority represents about 35% of the market. They would not adopt new technology until they consider it well-proven. In fact, they are not interested in technology at all, and do not have a lot of

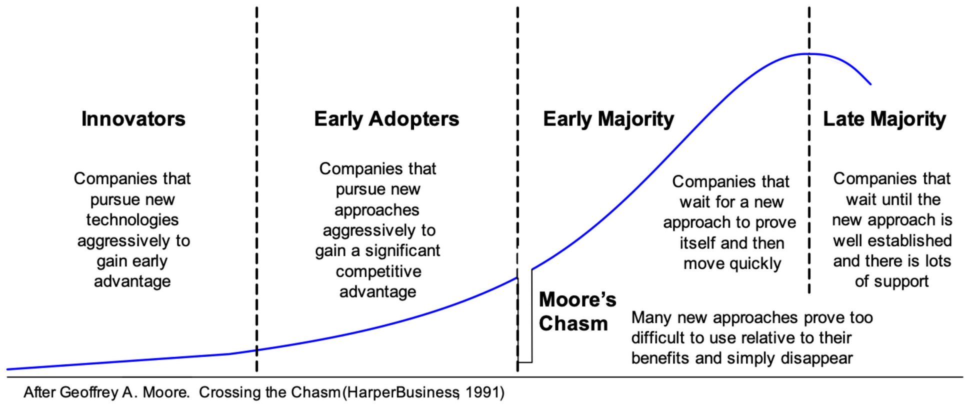
Innovators
Companies that pursue new technologies aggressively to gain early advantage
Early Adopters
Companies that pursue new approaches aggressively to gain a significant competitive advantage
Early Majority
Companies that wait for a new approach to prove itself and then move quickly
Late Majority
Companies that wait until the new approach is well established and there is lots of support
Moore's Chasm
Many new approaches prove too difficult to use relative to their benefits and simply disappear
After Geoffrey A. Moore. Crossing the Chasm (HarperBusiness, 1991)

FIGURE 3. Moore's technology adoption life cycle curve.

sophisticated technologists who are willing to struggle with the technology. They wait for case studies to show that the technology really gets the benefits that are claimed. And they insist on products that make it easy for less sophisticated developers to deploy the technology quickly, without significant difficulties.

Moore's Chasm

Moore's Chasm falls between Early Adopters and the Early Majority. Lots of technological innovations that are tried by Early Adopters fail to gain sufficient acceptance to pass the criteria of the Early Majority companies. The new technology gets lots of publicity, for a while. Conferences are launched to provide information about the technology and it is described in glowing articles in all the high-tech magazines and business publications that are always touting the next new thing. Ultimately, however, the technology may fail to produce enough concrete proof of usability and benefits to convince the Early Majority to make an investment, and the technology drops out of sight.

Late Majority and the Laggards

Companies that make up the Late Majority, like the Laggards who lie even further to the right, are reluctant to spend money or take chances on new approaches. They wait until their competitors among the Early Majority have started gaining benefits from the technology, and then reluctantly follow suit. These companies do not want to adopt new technologies, but finally do, because they decide they need to do so to survive.

When you go to conferences and hear vendors talking about the technological features of their product and why it is better technology than whatever came before, you are in an Innovator's Market. When the market begins to transition to Early Adopters, you begin to hear more business cases and get information on specific benefits. This is also the time when vendors begin to worry about wider acceptance, and become concerned with standards, user interfaces, and assuring their products can work with legacy applications.

If the technology is successful and crosses the chasm, the technology conferences tend to drop away, and the vendors begin to show up at traditional business shows and promote their products as a cost-effective way to solve a class of business problems within a specific industry. The majority do not care about technology. They just want to solve business problems quickly and effectively to get ahead of or at least stay even with their competitors.

When a new technology is first introduced, lots of relatively small vendors rush to offer products. If the market is small, ironically, the number of vendors is large. No one vendor makes very much money, but they are all full of hope, each spending venture capital and believing that their technological approach is superior. As the market grows and customers become a little more sophisticated, they begin to demand more comprehensive products and features and support for evolving standards. It is not uncommon for products to go through 3–4 generations over 2–3 years. The cost of constantly developing new versions of one's product, coupled with the need for more aggressive advertising, forces the smaller vendors to search for more capital to continue to remain competitive. Mergers are common and weaker vendors drop out.

Sometime during the Early Adopter phase of the market, the major vendors begin to incorporate the technology into their more comprehensive offerings and begin to promote the technology. In effect, the large vendors guarantee that the new technology is safe. As the competition heats up, most of the smaller vendors disappear. Larger vendors acquire some; many decide to specialize in industry or niche-specific markets. Others simply fail to earn enough money to survive. The key thing, however, is that Majority companies only buy from established vendors that they are reasonably confident will provide the rather extensive support they will require and who they are sure will still be in business 5 or 10 years from now. Thus, if a new technology succeeds in crossing Moore's Chasm, the leading vendors will be companies like IBM, Microsoft, and SAP. One or two of the new startups may have been successful enough to have grown into a 100-million-dollar company and still be viable in the Majority market, but most did not make it.

Moore's Model and the Expert Systems Market

As the 1980s passed, *Expert Systems Strategies* wrote about the market monthly, and frequently discussed the overall success of the market. Many CEOs from the larger expert systems tool vendors were convinced that the market had become an Early Majority market. In fact, it never really evolved beyond the Early Adopter phase and ended up falling into Moore's Chasm by the middle of the 1990s. In essence, it never provided what the Early Majority companies wanted: Solid proof that the technology could reliably deliver applications that would provide bottom-line returns on investment at a reasonable cost. While some applications delivered outstanding results—indeed, a few are still used today—most never really solved the compatibility problems, were too costly to develop and were very expensive to maintain.

Not With a Bang But a Whimper

The market for expert systems software tools, and the interest in developing expert systems applications died in the early 1990s. In essence, companies bought expert systems-building software, developed applications, experimented, found the effort was not worthwhile, and decided not to invest any more in expert systems.

OVERALL, HOWEVER, MOST OF COMPANIES THAT EXPLORED EXPERT SYSTEMS TECHNOLOGY CONCLUDED THAT THE COST OF DEVELOPING USEFUL APPLICATIONS WAS TOO HIGH, THE APPLICATIONS WERE TOO INFLEXIBLE, AND THE COST OF MAINTENANCE WAS UNACCEPTABLE.

Some companies developed valuable expert systems applications. A few are still in use today. Overall, however, most of companies that explored expert systems technology concluded that the cost of developing useful applications was too high, the applications were too inflexible, and the cost of maintenance was unacceptable.

There were significant hardware difficulties, but they were gradually being overcome as new generations of computers provided more power and speed. The real problem was the cost and time involved in the development of the applications—knowledge engineering—and the fact that cutting-edge applications involved constantly changing knowledge and thus needed to be constantly updated.

It is significant that today's renewed interest in AI has focused on a different set of technologies—often referred to as neural networks—that guarantee that the applications can learn for themselves and update themselves with use. It is too early to see if the neural network-driven approach will overcome the challenges that killed the expert systems market, but at this point, it seems likely it will.

By the late 1990s, most expert systems vendors had disappeared. Some expert systems tool companies survived until later, and large companies like IBM simply shifted their emphasis to other product lines as demand changed. In essence, the expert systems market never really achieved take-off. Some would argue that the market had advanced into the Early Majority portion of Moore's curve, but, in fact, the market simply disappeared into the Chasm. It happened over the course of a few years. Sales slowed, venture capital moved on, and companies laid people off, and then closed.

Several of the companies survived, however, by reinventing themselves and offering new services. The expert systems companies had introduced the idea of reasoning by means of rules that could be manipulated by an inference engine (algorithm). In the 1980s, while expert systems were enjoying attention, other companies had introduced relational databases, and those databases, in some cases, emphasized developing applications that captured business rules. In the sense described here, *business rules* were much simpler than those used in expert systems. Instead of working with human experts to capture their knowledge, one simply examined written business policies to evolve top-down procedural guidelines. Thus, a niche market for software vendors who supported storing and managing business rules evolved. Banks and governments were especially interested in capturing business knowledge as rules and some vendors switched from focusing on expert systems to offering what are now most often called Business Rule Management Tools.

In a similar way, the more sophisticated expert systems vendors had developed tools, like Intellicorp's KEE and Carnegie Group's KnowledgeCraft, that stored knowledge in object networks. The 1980s were also the time when Apple introduced the Mac (based on object-oriented concepts) and software developers became interested in new object-oriented languages (e.g., Smalltalk, C++) and object-oriented databases. Since the expert systems vendors already understood these concepts, several former expert systems tool vendors evolved into vendors of object-oriented programming tools or databases.

Intellicorp, by the late 1990s, was focused on offering a version of its KEE product that configured SAP applications. Texas Instruments used its expert systems tool as the basis of a Computer-Aided Software Engineering tool that it offered in the early years of the new millennium. Gensym evolved its real-time process-control tool into a tool for creating powerful business process management applications.

We would not bother to trace the way expert systems developers evolved to develop software games or graphics tools. Suffice it to say that expert systems represented the first major effort to bring commercial computing into line with lots of the new ideas that had developed in computer science departments. Companies may not have ultimately embraced expert systems, but they learned a lot about what was possible, and they hired lots of new computer science graduates who ultimately revolutionized what companies learned to expect from their computer service departments. Expert systems and expert systems-building tools faded in the late 1990s, but the concepts and the people involved in the effort proceeded to change commercial computing, making it much more sophisticated and flexible and they proceeded to prepare business groups for the eventual rise of neural network approaches in the following decades.

ACKNOWLEDGMENT

I would like to thank Avron Barr, Mark Fox, Curt Hall, and Celia Wolf for reading this paper and providing me with important corrections.

REFERENCES/ENDNOTES

[1] B. G. Buchanan and E. H. Shortliffe, *Rule-Based Expert Systems: The MYCIN Experiments of the Stanford Heuristic Programming Project*. Reading, MA, USA: Addison-Wesley, 1984.

[2] All figures in this chapter were originally developed for use in the Expert Systems Strategies newsletter, written by Paul Harmon and Curt Hall, and published by Cutter Information Corp. from 1985 to 1995. All are copyrighted by Paul Harmon and are used with permission.

[3] E. Feigenbaum and P. McCorduck. *The Fifth Generation: Artificial Intelligence and Japan's Computer Challenge to the World*. Kolkata, India: Signet Books, 1984.

[4] For a good explanation of the use of knowledge techniques for simpler tasks, see Chapter 11, "Building a small knowledge system," in *Expert Systems: Artificial Intelligence in Business*, P. Harmon and D. King, Eds., New York, NY, USA: Wiley, 1985.

[5] G. Moore, *Crossing the Chasm*. New York, NY, USA: HarperBusiness, 1991.

PAUL HARMON was an Educational Technologist in 1982 when he was hired by a new expert systems startup, Teknowledge, to help them improve courses they were offering to business executives to teach them about AI and expert systems technology. After working with Teknowledge for two years, he coauthored a book that explained the technology: *Expert Systems: AI in Business* (1985). It was the first book aimed at a business audience and proved very popular. Subsequently, he started a monthly newsletter, *Expert Systems Strategies* (ESS) that reported on developments in the expert systems market. ESS became the newsletter of note for the market and he authored or coauthored the newsletter from 1985 to 1995. During that time, he authored three more books on expert systems, offered popular training courses, and gave talks at leading Expert Systems Conferences throughout the world.