An Interview with
WILLIAM MITCHELL
OH 203

Conducted by David S. Cargo
on
26 July 1990
Flagstaff, AZ

Charles Babbage Institute
Center for the History of Information Processing
University of Minnesota, Minneapolis
Copyright, Charles Babbage Institute
Abstract

Mitchell discusses his introduction to Icon through his use of SPITBOL. He describes the process of making Icon compatible to various environments and notes the types of documentation and tools that are produced. He talks about several research assistants that he worked with on the Icon project. Mitchell discusses the usability of Icon in a software development environment and describes the use of C in conjunction with Icon.
CARGO: I am trying to get some background from the people about how they got involved with the Icon Project and what experience they had with programming in general before they got involved with the Icon Project. If you could go over your academic career a bit prior to your involvement, that would be the first thing I would like to cover.

MITCHELL: My first real exposure in computer science was at North Carolina State University. I was a freshman there in the computer science program. One of the first trips I made to the library there I happened to notice a book on SNOBOL4. It seemed like an interesting language, so I checked out that and a stack of other books on some odd things. I don't know if I ever got around to reading the book; I guess I looked at it some, you know, but it stuck it in my mind. My sophomore year I took a class in SPITBOL. That's one of the faster implementations of SNOBOL, and I enjoyed the course. I started using SNOBOL in all the classes I could find some use for it in. One day, in I suppose it must have been 1978 or 1979, I noticed on the board there was going to be a seminar given by Ralph Griswold on the Icon programming language. Since I knew that Ralph had been involved with SNOBOL4, I said, "Well, SNOBOL4 is a good thing, so maybe this is a worthwhile thing to go to." So I mentioned it to some of my friends. We started speculating about this name, Icon, so we looked it up in the dictionary and it didn't make any sense. I said, "Maybe it has something to do with images of strings, or something." Anyway, I went to the talk. Ralph gave a presentation on the Icon language and said it was being developed at the University of Arizona. They were working on a portable implementation written in Ratfor and they were going to make the implementation available.

After a span of a couple of months I wrote to Ralph saying that I had attended his talk at North Carolina State, and I was wondering if I could get a copy of Icon to try to port it to an IBM 370. He sent me a tape and he said, "A couple of 370 ports are in progress, but you are welcome to try your hand at it." On the 370 there were a number of problems posed. Icon as distributed was a tremendous number of source files and it was a problem in itself. I started working on it though. I don't know how long it took. I worked on it during nights and weekends just at odd times for maybe
three or four months, but I finally got the thing built. That was quite a feat, I thought. We had problems with the FORTRAN compiler - just everything imaginable. But I finally got a system built.

As I recall they supplied a guest account at Arizona for people who were porting to send mail and stuff. Using that account I wrote to Ralph that I got my 370 implementation working and he was pretty excited about that, and so we of course wanted some more follow-on data. I believe it was Version 1.2, or maybe even 1.1; it was a very early version of the language. I ended up porting the Version 2 FORTRAN system to VAX/VMS as well, and as I recall the final 370 port took about maybe a couple of weeks off and on. Everything was difficult on the IBM system, but on VMS it took a couple of days maybe. During this time I was continuing to work on my degree. Around the start of my fourth year, the autumn of 1979, I learned that they had a UNIX implementation of Icon. We had a UNIX system on a PDP-11/40, but they had implemented Icon for a split I/D PDP-11, which had the instruction and data spaces separated. Since our PDP-11 running UNIX didn't have the split I/D spaces, I couldn't use that implementation.

Shortly after that, though, I became the system manager of a VAX UNIX system, and so I got Icon again. This time it was Version 3 in C, and I was trying to think how that went. Actually, it was developed on a PDP 11. We got a VAX and a neighboring site had a PDP-11 running UNIX. I installed it there and messed around with it some. I really didn't do much serious with it. I guess a subsequent version said, "I'm going to see about porting this to the VAX," but nothing happened there. Later on, I think another guy ported Version 3 to the VAX; that was Paul Eggert, I believe, at the University of California, Santa Barbara. So anyway, I was around Icon, but I really hadn't spent much time programming in it. I had gotten some independent study credit for the ports of Version 2, and I did a couple of other things with it. I forget exactly what; nothing significant as I recall. At this time I was graduating and I was looking around for a graduate school. I got it down to three possibilities: the University of Arizona, the University of Central Florida, and Virginia Tech. I was from North Carolina originally, so Arizona was a long ways away. At Virginia they only had an IBM system, but in Florida they had a fast new VAX. That seemed the best to me. At Arizona all they had was what seemed to be a fairly overloaded PDP-11/70, so that didn't sound very exciting.

So I went to Florida, but shortly after going there I found that the new program really wasn't too strong. I still had my
guest account at Arizona. Ralph and I had corresponded just off and on, and so I sent him a note about something and I mentioned that things weren't working out very well in Florida. He said, "That's unfortunate, because we had hoped you would come to the University of Arizona." I said, "Well, maybe I can still come." [laugh] So we made some negotiations. I wanted to be an RA, not a TA to start with. We worked it all out. I transferred to the University of Arizona after a year in Florida, so I started in Arizona the fall of 1982. I remember distinctly my first meeting with Ralph. For some reason someone was using his office; I guess one of his guests or something was using his office so he was in another person's office. We sat down and talked about a variety of possible things I could work on. What he decided was to start me on some simple work. He had a couple of bugs - I guess in the left, right, and center functions - a couple of odd cases, so he assigned me to work on those. I got those fixed up and I guess he liked my work, because he then put me on porting the Icon interpreter to the VAX.

At that point Icon was at Version 5. There was a compiler and an interpreter both. The compiler had been ported to the VAX, but the interpreter hadn't, and so what he wanted me to do was to get the interpreter working on the VAX. A lot of time was put into working on assembly language code. I remember the first version. One of the routines in one section of code must have had 50 assembly language instructions that was later reduced to ten. The instructions had been added by people who weren't sure, and they did this. I eventually got that going. That was really the first thing of any import I did on the project.

As I recall, after getting the VAX interpreter working, I spent a lot of time trying to get the two sources identical on both the PDP-11 and VAX systems using #ifdefs as appropriate, and also working on various sorts of compiler problems. The PDP-11 compiler and the VAX compiler were of different heritages. The PDP-11 compiler was the so-called Richie compiler; the VAX compiler was based on "pcc" -- the Portable C Compiler. There were a few things that were done in Icon source on the more permissive PDP-11 compiler that weren't permitted on the VAX. There was a period of a couple of months just basically trying to beat down some problems in the source.

At the same time, I didn't really know Icon to any extent when I started working on the system. I had been involved in porting, but that was about it. So at the same time I was porting the system to the VAX, I was taking Ralph's Icon
programming course. I was sort of a step ahead but not by much. There was a fairly extensive test suite -- I guess it was like 80 some programs and essentially all of them had all sorts of odd expressions and stuff. I didn't know enough about the language to often know what the correct output should be in a lot of situations, so I was always having to work off the PDP-11 version, which was known to be correct. I would compare that against the behavior on the VAX.

I was trying to think what the next thing I did on Icon was. I got my nose in a couple of other things. 1) I got the Icon-group mailing list organized at some point there. I got the systems in computer science hooked up to some other networks so we could have some wider reaching electronic communication about Icon. At that point I was handling a lot of the Icon mail that was coming in. In the process of porting Icon to the VAX, there was a lot of assembly language code involved for the routines that implemented the control structures, like suspension and failure indication; anything of that sort was done in assembly language. In the process of porting the code I had learned a lot about how the assembly language portions of the system worked, so I wrote a document. I guess it was called "Porting the Icon Interpreter," or something like "Porting the UNIX implementation of Icon." That was a 60 page document that basically just described in great depth what each of the assembly language routines did. I spent the better part of a summer working on that.

This whole time I'm thinking that there had to be some sort of a better way, but the assembly language seemed to be inescapable. It seemed like we had to have assembly language to do all these manipulations that were needed.

After that we got a Sun-2 in and I ported Icon over there, once again writing all this assembly code. By that time it seemed like I had a pretty good understanding of how all that stuff works. It was pretty much a straightforward thing.

Another thing done along the way, Ralph wanted to teach an internals class in Icon, so we had all this code but it hadn't been documented very well. So I got the job of documenting the source code for the Icon system. What I would do was I would get a big bottle of Dr. Pepper and a box of Wheat Thins, and just go type in comments,
analyzing sections of the code, and stuff like that. I don't know how much all that came to, although it didn't seem like it took more than a couple of weeks to get all that source code documented to some extent. In the process of doing that we learned a few more things about the system.

Things are a little blurry here, because after I graduated from the University they hired me to a position on the staff there, so I am having a hard time separating the things I did as a student and the things that I did on the staff. Let's see, one thing I did on Icon was an independent research project. I had been a user of the Emacs text editor, written by James Gosling. And I don't know if you know anything about Emacs, but it had an embedded language called MLISP - MockLISP - that had the syntactic structure of LISP and some similarly named routines but it really didn't have a lot past that. But what it gave you was the capability to program the editor. The problem was it didn't have any concept of any data structures. If you wanted to have a list of some sort, you had a buffer and the lines in the buffer would be the elements of the list. It was very clumsy.

I said, "Gee, what if we could replace MockLISP with a real language in there like Icon?" and I spent a while considering that. I was familiar with Icon internals, but I didn't know much about Emacs, although I had poked around in the code some. I studied the Emacs code, and I got enough figured out to see how to do it. I said, "Well, before I start to do this I had better get an up-to-date copy of the source." I did, and I found out that all the argument handling mechanism had changed and that was really the crux of the problem - a way to pass arguments from an Icon routine to a MockLISP routine and vice versa.

Anyway, so finally I got the system working, and at that point we had a version of Emacs that had a complete Icon interpreter embedded in it. The system was called "ice". It provided a facility to load an Icon file just like you could load a MockLISP file. It was possible to have an arbitrary interweaving of Icon and MockLISP routines. It was a very general system, but in the end it turned out to be a maintenance headache to try to keep up with updates of the Emacs source. There also were a lot of problems with the basic architecture of interface between the MockLISP subsystem and the Icon subsystem. I tried to have the interface be as small as possible, but in the end it looked like this fairly narrow interface that made it easy to implement was the cause of the difficulties, too. I had hopes of
reviving ice for a while, but finally I just said, "Aw, I haven't got time to experiment..." But ice was a pretty interesting thing.

One of the things I did while working for the department, which is probably the most important thing I did for Icon, was to nearly eliminate the need for per-machine assembly language routines. I would answer mail from people porting Icon to various machines, and much of the correspondence was about porting the assembly language routines. I don't know how much mail I sent, but it seemed like a lot. I don't know what the next port was. It might have been the AT&T 3B series. On the VAX and the PDP-11 the stacks grows down, but on the 3B it grows up. That caused a few problems, and it seemed like there had to be a better way. Phil Kaslo (he was an employee of computer science - still there) was doing some groundwork for a port to one of the HP 9000 series machines. He was saying, "Why can't we just write this whole thing in C?" People said, "No, that's impossible; it can't be done." And I don't know if Phil understood the problem then or not, but I got to thinking about it and it seemed like it would be possible by using a recursive model for the interpreter -- by using recursion to model the suspension operation. That was a real problem. I started working on that and I was doing one piece of the interpreter at a time. I was always worried that I was going to find some absolute stopper, something that would keep this thing from working, but I never did. Eventually we had a version of Icon that had no assembly language, except for the code used to implement co-expressions and overflow checking. To this day, that assembly language code still remains. Although today I was thinking that if we could just get rid of that assembly language code for co-expressions we would be in pretty good shape. I don't know what can be done there.

Another small Icon tool I wrote was a system called Ixis. The idea there was to observe an Icon program as it executed. What I had planned to do initially was to have a program to be studied loaded along with a number of Icon routines that would monitor this running program. I had this really elaborate scheme sketched out, but I was lazy and I just didn't want to start out working on it. But then one day it occurred to me that a better way to do it would be to use UNIX's ptrace system call. It lets you do things to a process like get a word out of a process, put a word in a process, and get control when break point is reached. What I did was I wrote some specialized routines for the interpreter that made use of the ptrace call. I inserted break points into the interpreter, and so by doing that we were
able to have a way to monitor a running Icon program without having any interference, based on the observation that the only interference that we possibly could have had would have been if there were some sort of a bug in the interpreter that would have caused us to fetch the memory where the instruction of the break point was stored - do something like that. But aside from that, it was a very clean implementation.

IXIS ran on a ASCII terminal and had a two-part display. Up at the top it had a representation of what was on the Icon stack. I don't know how much this has changed these days, but at that point in time an Icon stack would be composed of procedure frames, generator frames, and expression frames. IXIS would set a break point at the top of the main loop in the interpreter. Every time execution stopped there it would look at the stack, print its tokenized list of what a stack looks like. It had a source code window under it that would have a pointer that would move from line to line, so as each pass from the interpreter was made this arrow would move appropriately. The stack would grow and shrink. We would have an idea of what this stack looked like while the program was running. I did various other minor work on Icon. I did some work on the garbage collector - things of that sort. I don't know if I can think of anything else.

CARGO: I have got a list of questions to go back over. When did you graduate?

MITCHELL: I graduated from the University of Arizona with a Master's in 1984 - spring of 1984. I worked in the department from May of 1984 to March of 1988. Well, I started as an undergraduate in the fall of 1976, graduated with a B.S. in May of 1981. I was in Florida from August of 1981 to August of 1982. I graduated in 1984. And while I was in graduate school I was a research assistant on the Icon Project. Once I was hired onto the staff that position was slated as half research, half support. And Ralph was just one of the persons that I worked for at the time. A person on the faculty would write up a research support request, outlining working at some level, detailing assignments where they passed out to people in a position such as mine. But I did the work. Because I had had experience with Icon I tended to get a lot of Icon assignments.

CARGO: When you went to the U of A were there other RAs at that time working with Icon?
MITCHELL: Well, as you probably know, the two people originally involved with the UNIX . . .

TAPE 1/SIDE 2

MITCHELL: . . . implementation of Icon were Steve Wampler and Cary Coutant. By the time I got here Steve was still working on the assignment he had. I mentioned earlier Steve had done some of the work on porting the interpreter to the VAX, but he didn't have much time to put into it. At that point he had just moved to Flagstaff. The other two people on the project were Mike Novak and Allan Anderson. Mike was working on programmer defined control operations and Allan was working on some higher level list operations. And so I guess I was the fourth person to get my hands on the UNIX source behind Cary, Steve, and Ralph.

CARGO: Were there additional RAs who came on when you were working as an RA?

MITCHELL: Well, I was trying to think what the next step there was. As I recall, Mike was graduating that fall, or maybe spring. I am not sure; I don't recall. But he was around. I don't know if Allan graduated or not. You know, once again, I don't have a lot of memories of them past that first massive output. Ralph would know that better than I would. I was trying to think who the next person to work on Icon was. That's a good question. One of the other persons working on Icon was Janalee O'Bagy. She ended up going on and getting a Ph.D. with Ralph. Originally she was working on the SR Project. She approached me one day and said that she had a chance to work on Icon and wanted to know what I thought about working for Ralph. We walked around the block and we talked about that. She ended up switching over to Icon.

CARGO: Was this when you were still working on your Master's?

MITCHELL: Yes, I was still on my Master's at that point. That might have been actually just after I graduated. I had a hard time figuring who else was working on Icon. I believe that there might have been some problems in that period.
with either a grant coming through slowly or something, so it wasn't clear what the support situation was going to be.

But I don't know. Ralph would probably say, "Oh, yes, there were so and so, and so and so." I am having a hard
time thinking of who else was involved with Icon. I was pretty much the only person that had my hands in the code
for quite a while.

CARGO: When you were working as staff, how often do you think you wound up working on things for the Icon
Project?

MITCHELL: That's hard to say. That position was probably less that optimum because it put me in a position doing
system management, while also trying to do research support. I personally had a hard time leaving the system
management stuff alone if I felt that there was a problem that had to be taken care of. I was constantly dividing it as
where do these problems go, and go ahead and do the research support stuff. In the four years I worked with the
department I might guess that the equivalent of one year of that might have been on Icon stuff. I was one of the
ones who wanted a debugger for Icon, and one of the research support projects was to work up some ideas on that. I
did that, but no further action was taken at that point. Another guy who also got involved was David Gudeman. I'm
having a hard time remembering what he was working on initially. I know he and I did a couple of small things
together on a string functions. So about a year of the four years or so I spent working on the staff I was probably in
some sort of Icon-related work, although I was still on the Icon project mailing list. I was, you know, very much
involved from an interested observer standpoint, but I wasn't dedicated to that project.

Something else. We would take another crack at a version of Icon that had support for some of the things I had done
with the Icon and Emacs system. When I had done that there were a few corners that were cut and quite a few
problems remained, so I worked on that for a little while. I forget what finally happened. It seemed like we just sort of
lost interest or lost the time to work on it, so that sort of just petered out.

Ralph was in charge of the software side of the lab. We had a couple of Icon things going on and some more work
on interactive Icon. It happened, but nothing really came of that, but a few ideas; I guess some of the work that Bill
Griswold did built on that work.

CARGO: What kind of working relationship did you have with Gregg Townsend?

MITCHELL: Well, Gregg was hired into a staff position the same as mine shortly after I was hired. Gregg and I shared an office for a while. He did various Icon things too. In terms of a working relationship, I am not sure what you mean. I don't recall any of the things that I did on Icon really were a matter of collaboration with anyone else. It was basically like Ralph would direct a number of individual projects, and the whole thing would fit together. It was only towards the end of the time that I was involved with Icon that there was starting to be more than person with their hands in any one part of the source. So, you know, Gregg and I both worked on the Icon Project, but I didn't work with anyone else in particular.

CARGO: What sort of decision-making process was used to determine what piece of Icon you would be working?

MITCHELL: Well, I am trying to think of how that whole process went. In the Icon Project, one of the unique things about the project, I suppose, is that there was very extensive use of electronic mail. I don't know how many messages we might have sent out on an average day, but it might be like on the order of 50. Ralph and I did the bulk of our communication through electronic mail. We would meet maybe either once a week or once every couple of weeks. Something that Ralph told me was that people do their best work when they are working on something that they are interested in. As a rule we always had plenty of things to work on in Icon, so he and I would just discuss what sort of things seemed interesting. I was always willing to get involved with implementation, so I got jobs that dealt with having my hands in the code. Other people were more interested in kind of a "sit back and think" situation, perhaps with a small, clever idea, and they got jobs with desk work. Ralph seemed to basically try to take a set of assignments, whether it was a set of possible jobs, look at the interest of it as RAs, and then make assignments based on that, and in the end it seemed to fit together pretty well.

One of the things that always stood out in my mind was Ralph's ability to direct the work. He'd direct and then we
would meet. In those meetings we would usually hash out a pretty good plan for the next couple of weeks. With electronic mail we would handle details. One thing about Ralph and electronic mail is that he would always respond to messages very quickly. I don't know how he managed to keep up with it, because I was having trouble keeping up with just correspondence with Ralph. He had the same level of correspondence with other persons as well. So that always surprised me. As far as the general structure of the way the work was directed (I am sure Ralph has the correct view of this), it seemed to me that basically it was just a matter of looking at the number of tasks that had to be done, looking at the interest that was raised, and making assignments appropriately. You know, at that point a lot of interest was being placed on getting Icon to work on a number of platforms, before Version 6 when we changed from assembly language to C for the control structure manipulations. You know, it wasn't a lot of work to get Icon to work on a machine after that. It was a matter of compiling, if you were lucky.

CARGO: It's certainly a good start. That leads me to the second part of that, and that is when you were no longer an RA, when you were working on the lab staff. Was that still approximately the way that he parcelled out the work?

MITCHELL: It was essentially the same. It seemed like a lot of the stuff that I worked on seemed to be small ideas that I had for stuff. One of the things was string invocation. I am getting off the subject, but I guess that's okay. I remember when in the Icon course we were talking about a mutual goal-directed evaluation where you would have a number outside and a list of arguments in parentheses. Any numeric value selects a value to return. In a class Ralph remarked, "If we could think of something to do with strings we would do that too." And after class I said, "Well, gee, if it's a string then, how about we could just see if it's a procedure or a built-in and convert it, or an operator, for that matter?" So that was typical of how things would get started. We would just kick ideas around, and it always seemed like a meeting with Ralph was always a very fruitful process in terms of just the ideas that were cooked up. This is off the subject again a little bit, but apparently at one point in the early days of the computer science department they had these research meetings where the faculty would go to a meeting and talk about projects to work on, and what Ralph said was that they'd meet once a week and come up with a month's work. [laugh].

But getting back to your question, when I was on the research staff I was still communicating with Ralph via
electronic mail. You know, all of the work came up by someone maybe some other site, comments on something. They would say, "Gee, maybe this would be a good feature to add." And I might write, "Yes, I think we could do it this way." And maybe a couple weeks later I would get a slip in my mailbox, you know, and it's a research support request. And so I would mail to Ralph and I would say, "Hey, I have got this request. We need to talk about it." So we would meet and we would do that. So on a research basis it was very much the same sort of process once the work landed in my queue as it was when I was working for him, but we would have these periods when there was no work assigned to me on Icon.

CARGO: What was a typical range of tasks that you were invited to do or assigned to do for the Icon Project that way?

MITCHELL: Well, there were a number. Well, I would say the largest thing was certainly the Version 6 interpreter. Where the assembler routines were replaced with C routines. Next was Ixis - that was the program that observed another program running. I added instrumentation to the interpreter at one point to see how often the stack was swept, how many times things failed, stuff of that sort. That might have been done for some graduate students. In terms of large things done as a staff person, it was Version 6, Ixis, - I am kind of blank on this. I can recall there were small things, but I don't recall any other large projects.

CARGO: I was asking how long they took.

MITCHELL: How long they took? I'm sorry. It could vary in length. Some were an hour. I guess the Version 6 maybe took a couple months; but that was real time - maybe a month of "system" time. Hard to say. Thinking back on it, it doesn't seem like a long time now. You ask what the typical range of work was - a typical project might take a week to do - some a little longer.

CARGO: When you were writing your programs what sort of coding characteristics were you striving for, like portability, readability, efficiency, ease of modification?
MITCHELL: Well, I don't know if there was any special attention there. Certainly portability was a must - no doubt about that. I always tried to change as little as possible of the existing code, so there was typically a lot of study involved making the change, then actually doing it. In terms of readability, I don't recall a lot of emphasis on that, but I have always thought of the interpreter as being a fairly simple program. Basically it just has a core that provides interpretation of virtual machine instructions and garbage collection, and a set of routines that map nearly one-to-one to operators and built-in functions. I always thought of the interpreter as being a fairly simple thing from a structural standpoint. All of the code in the individual routines is not complicated. It's not a program that has a lot of structure to it. It's fairly straightforward, I have always thought.

CARGO: Did you work much on the translator?

MITCHELL: I did a few things on the translator. One thing I did was I got it speeded up quite a bit. It seemed like in the linking step in the interpreter it would take about as much system time as CPU time. We isolated that to an unbuffered write, which generated a kernel call for every byte being written. But most of the work I did really focused on the interpreter and making it more portable. The translator and linker were fairly portable. There were a few things there, but there was nothing major. I did some odds and ends on the translator too. I don't know if you're familiar with the Yacc. One of the lost Icon programs was this thing that would take Yacc off and generate an error table. So I upgraded that once, but the rest of my work focused on the interpreter.

CARGO: Did you wind up advocating any particular changes to the language, and did any of them get adopted?

MITCHELL: When I started on the project I had this initial slew of ideas for things to add. Ralph said, "For every feature you want to add be ready to throw one out." [laugh] In terms of language features, I guess I was the first person to suggest string invocation. Well, actually, for string invocation they had thought of that before but they didn't know what to do with it, or they hadn't thought much about what to do about operators that are both unary and binary like minus. I knew that could just be solved with looking at how many operands are involved. There were
two other little things, and one was a sequence function. Seq generates an infinite sequence and we added that, but in retrospect I don’t think it was good idea [laugh]. That was just added so if you were writing an infinite loop you didn’t have to decide how infinite you wanted to be. There were some other minor things, but in terms of the language features I never really had much input at that point. During the stage I had the heaviest involvement with the language, the general sentiment was the language was pretty much stabilized. So at the time I was involved with Icon there really wasn’t a lot more happening with the language. I felt the language itself was fairly solid - I just wanted to port it to try to make it available to a large number of people, so it was what the bulk of my work as a research assistant focused on.

CARGO: When did you wind up stopping work on the Icon Project? I am not sure what happened between the time you might have left the University of Arizona and now, for example.

MITCHELL: Right. Well, at the moment I work for a company in Tucson. I design medical information systems software. The last year I was employed by computer science I don’t recall doing a lot with Icon. At the time of the first Icon workshop, you know, Ralph invited me but at that time I felt that I was sufficiently out of touch with the language that I didn’t really have a place here. So I said, “I had better pass it up.” So working back from that I guess you could probably say that my involvement with the project pretty much came to an end sometime in 1987, in terms of any kind of serious work. I guess the language was really winding up. That was the time that Ken Walker started. I’m not sure exactly when he arrived. I was pretty much out of it by the time he had gotten into it. As I recall, there was just a little overlap there, and that’s why as I say 1987 is probably good time. It was roughly eight years.

CARGO: Do you still have an involvement with the Icon Project, or are you now an Icon user?

MITCHELL: I am an Icon user now. As I mentioned this morning, (I don’t know if this appropriate to discuss here), but the real problem I see in Icon is that a lot of the programs that I have had occasion to write usually involve a system call of some sort. It was always awkward to try to do that type of stuff in Icon, so I ended up doing more stuff in C and less in Icon. I thought that C++ was going to be the answer there, and that it has a way to provide
similar facilities to Icon. But it's just not as nice. It's clumsy. For a big program, I suppose C++ works out well. One possibly interesting thing to think about is when programs start feeling shaky. I always get nervous when I get an Icon program that's over one or two thousand lines. In C maybe it's 10,000; C++ maybe it's, I don't know, higher.

And so anyway, I felt that Icon had some problems for the type of work I was doing so I sort of got away from it for a while there. But I guess recently I have been coming back to it more. One of the influences that Icon has had on me is it seems like Icon has a fairly (I know Ralph would disagree) small set of features in some ways, but it lets you get a lot of things done. For example, there's really only a handful of things that can be done with a list. You can get the size, put things on the ends, get an element out, change a value. But it's really just about all you'll need. What we have done in C++ is to implement C++ classes that work along the same lines as an Icon counterpart, and it has worked out fairly well. So you are going to have a lot of economy in Icon. I have used these concepts that I learned as being good in Icon and taking them and applying them elsewhere.

TAPE 2/SIDE 1

CARGO: Later versions of the Icon system added a facility for adding your own C routines to the interpreter. Do you think that, given that opportunity, you might put those system calls you always wanted to make into the interpreter so you could go back to using C for the interfacing and Icon for the main part of the program?

MITCHELL: Right. It has always been possible to add those routines that required specialized system facilities. I haven't looked at Version 8, so I don't know exactly what work they have done and how that differs from personalized interpreters. Could you tell me some more about what you know about that and then maybe I could decide.

CARGO: Sure. What they did is they added an explicit outcall routine that passes everything through a gateway to one C interface routine that is responsible for decoding the arguments and then dispatching to any of a number of C routines that would presumably be linked in when the interpreter was built.
MITCHELL: Well, that sounds about like the right way to go. There might be some ways to go further with that. I guess the kind of thing I have always wanted to do was to be able to have a program that I could dump a bunch of ANSI C compatible header files through and have this generate a set of Icon interfaces, Icon stubs, for all these routines. Maybe, though, this facility is just the type of thing I have been wanting. Yes, it will be interesting to see how that goes. I think one of the problems you are always going to have is that C is C and Icon is Icon. They are just different things. Look at something like a system function like Read, for example. It takes a file descriptor, a number of bytes, and here's a part where it takes a buffer. It's just hard to really get that map back. You have to allocate a string in advance, and then put the result in there. I don't know if it will ever be possible to use Icon for systems programming as conveniently as you can C, but I still hold a hope for that. I think some of the VAX work can tremendously increase the potential of Icon for expansion or for a wider acceptance.

Another problem I have always with Icon is a lack of a debugger. [laugh] Maybe I am just a sloppy programmer, but I came up using a debugger, and SNOBOL had a debugger that would be just dumped in with my SNOBOL programs, so I got used to being able to look around inside of running programs. In Icon it's just not as convenient, so I figured that was another thing that has kept me from using Icon, or that's made me pick C in some cases where I might have picked Icon otherwise. Another thing I think about in terms of Icon is what is the source of the power of Icon, really? It seems like really what it comes down to is simply the ability to not have to work up a large critical mass to get a job done. With Icon it's just barely a subcritical mass; by adding even just a few lines of code you have got the ability to accomplish a significant number of tasks. But for larger programs, it seems like once you have built up this critical mass, it's not a lot harder. You can imagine you have a curve that has development time on a horizontal axis and functionality on the vertical axis. With Icon you get a curve that's very steep and it rises very steeply in the beginning; then it begins to flatten out as the time increases. With a conventional language like C or C++, you see more of an S-shaped curve where it takes a while before you actually start to see things happen; when the curve starts to climb once this thing I call critical mass starts to be achieved. I don't know how much success Icon will eventually have in the large program area. Maybe that's not really been the intention. I would like to see some way to keep this steep climb of functionality all the way through a program development cycle that does not have it flatten out. With Icon you have this steep climb initially, and it just seems to taper out. With conventional
languages, as you know, it has a slow start, but it eventually gets up to that same peak. So I am not sure of a real key
in this whole puzzle of programming languages. You know, what's the source of a language's ability? I think Icon
starts out with a good base; it gives you the things you need.

In Icon it's not hard for me to remember what Icon function I want to use. The function setup seems to be very well-
designed. It seems small in comparison to something like LISP where you have got hundreds or thousands of
functions of some implementations, all that do slightly different things - it's very hard to write a common LISP
program without spending some time with the manual. So that's an advantage too. But anyway, I have gotten way
off [laugh] of your track.

CARGO: Have you tried using Icon as a prototyping language for things that eventually get written in C?

MITCHELL: Well, I don't know if I have done a lot of prototyping. I have certainly used Icon for some things that
simply weren't feasible in C. One thing I mentioned is that disk allocation system. That was just a program that
couldn't have been written in C. There simply wasn't enough time to do it. If I had said, "It's going to take five times
as long," they would have said, "Let's find another solution." But as far as prototyping in particular goes, I guess
there have been a few things I have done - maybe an algorithm that I wrote in Icon first and then translated to
something else. Although if it's a complex algorithm I might be inclined to not use Icon. If it's something that's
implementable in C or in Icon I would probably do that in C because I have got better debugging tools in C than I do
in Icon. My lack of use of Icon for prototyping partially rises from just the type of things I have had to do. I really
haven't been faced with a lot of situations where I have had to use a prototype for something. It has usually been a
matter of if I can do it in C or I can do it in Icon and if I thought it would work in the long term in Icon, if I didn't I
would do it in C, so prototyping is really something I haven't done a lot of.

CARGO: So you tend to skip a prototyping step if you can foresee where the program is ultimately going to be
implemented.
MITCHELL: Right. Well, there have been a couple of programs that started out in Icon, ended up in C. Well, there was one thing - a news archiving system. I wrote the first version in Icon, I think, while I was waiting for a phone call or something; it was like a page of code. And we wrote another version in C. I don’t know, it took a long time. It was probably just as good as the Icon version. It was a lot quicker. There were a couple of unusual programs that were done first in Icon; then once we felt we had the concepts worked out we said, “Well, we are going to speed it up; we’re going to do it in C.” And it was, “You never should have; you probably should have left it in Icon.”

CARGO: What would you say the high points were with your involvement with the Icon Project?

MITCHELL: Well, I guess working for Ralph is a high point in itself as far as my career goes. Ralph is a tough guy to work for. He works a lot, you know. It seems like he works all the time almost. I am sure he doesn’t, but he is always on at the crack of dawn, it seems like, and stays on all day. And working for him is challenging, but it has also been great to work for someone I think really stands out in the field. A lot of languages have been designed, and various ones have gotten to various stages. A lot of Icon I am sure is still being written, but SNOBOL achieved a substantial following. A lot of people with a computer science background have always heard of it, or maybe they had in a class. I think Icon is in that same category, and maybe we have got in more Icon per hours, more to come in the years to follow.

And I think the fact that Ralph has been involved with two languages that have had some level of success is a pretty good thing. Just having the privilege to work for Ralph has been quite an education. It seems like I worked for Ralph for a couple of years full time or half time basically, and then some time after that I knew a lot more than I knew when I started. I feel like I got a lot of that just from being around him, seeing the way he handled problems, the way he approached things, and just the way that he worked on stuff. It was always an inspiration. On the other hand, Ralph can be a frustrating person to work for. He works a lot, and I guess he expects a lot from people that work for him. So he is demanding but he is very fair. I guess I am answering a different question here really, but at least it would be a little more background of this.
I always thought that a person in Ralph's position really shouldn't get his hands dirty with stuff like going in and taking care of little problems in the source, and being in on the distribution and stuff like that, but Ralph always would pull his fair share - surely more than his fair share of that, really. That was always a frustration to me that I felt he had better things to do than do this work that probably should be done by someone at a lower level such as myself. But he really works on a system from top to bottom.

In terms of the highlights, the one thing that just stands out for me is just the thing I did with getting rid of some of the assembly language code and making this available to all the people. In the long run that's the accomplishment that I will look back on and say, "That is what I did for Icon," although I did some of the initial work in porting the stuff. If that step hadn't been made I don't think we would see Icon like we do now. We would see it on a few platforms, I think. We would have some huge maintenance headaches. I say "we." The Icon Project would have a lot of maintenance headaches. Instead of seeing just a couple of files for each implementation, you would have something like 15 or 20 assembly language files, or even problems with different version of assembly. For example, there's the 68000 version - it might work on the Sun but not on the Apollo, for instance. I think of that as a highlight, freeing Icon from the shackle of assembly language. Although I also think that if I had been less willing to write that porting document then maybe I would have seen that it could be done in C sooner. So instead of writing this huge document, if I had just been watching, I would not even have been thinking about that problem. So I was too willing to say, "I don't think it can be done." And I always thought we had to have it, but once I started thinking about it we didn't have to have it. So maybe co-expressions can be done without assembly language. I guess it's fulfilling, too, to think about just being involved with a successful thing. I mean, this thing I am working on in my current job is interesting, but I think 50 years from now it's not going to be in a history book or anything. I think Icon might, so that's important.

CARGO: Now, as I recall, the book, I think it is "Implementation of the Icon Language," coincided with Version 6, which is the one that had your recursive calls to the interpreter, is that correct?

MITCHELL: Ralph would be the one that could settle this, but as I recall, I came up with that just as Ralph was
maybe not in the final stages of that, but definitely he was heading towards the end of that, so he had a hard time deciding whether to just make that like an appendix, or have the assembly code in the main portion, or what. And so that was about the time that he was doing maybe the final draft, but it seemed like he was heading towards the end of that development. And he had this idea. [laugh]

CARGO: Were there any other aspects of your involvement with Icon or the Icon Project that you wanted to mention?

MITCHELL: Well, I've pretty much covered everything that I jotted down. I certainly have a lot of both fond and vivid memories of a lot of that. But in terms of what else, if I could do things differently, I would have to take an organic view. I certainly wish that I hadn't invested all the time in that one document about porting assembly language version instead and put some more thought into that up front. I wish that I had on my own time developed an Icon debugger. I think that that would have been a good volunteer effort. In terms of other things to mention, gee, I don't know, I can't think of a thing - very enjoyable experience. I'm sure I'll see it as one of the highlights of my career when it's all over.

END OF INTERVIEW