

Babbage - The language of the future

Author's notes:

This article originally appeared in Datamation, September 1981.

Long before Java, long before Perl, and long before C++, Ada was the favorite of the hypemeisters and headline writers, who predicted it would revolutionize the process of software development, eventually replacing outmoded languages like COBOL, FORTRAN, and even Pascal.

Guess what happened?

Not much. Things went on pretty much as before. Banks continued to use COBOL, NASA continued to use FORTRAN, and the common folk continued to use nasty languages like BASIC and C.

Ada, like many a behemoth, collapsed under its own weight and, although it still has its proponents, it didn't revolutionize much of anything.

It's just something to think about while you're reading today's headlines.

"I've seen the future, and you're not going to like it."
> Vincent van Gui

Babbage - the Language of the Future

There are few things in this business that are more fun than designing a new computer language, and the very latest is Ada - the Department of Defense's new supertoy. Ada as you know, has been designed to replace outmoded and obsolete languages such as COBOL and FORTRAN.

The problem is that this cycle takes 20 to 30 years and doesn't start until we're really convinced present languages are no good. We can short-circuit this process by starting on Ada's replacement right now. Then, by the time we decide Ada is obsolete, its replacement will be ready.

The new generation of language designers has taken to naming its brainchildren after real people rather than resorting to the usual acronyms. Pascal is named after the first person to build a calculating machine and Ada is named after the first computer programmer.

As our namesake, we chose Charles Babbage, who died in poverty while trying to finish building the first computer. The new language is thus named after the first systems designer to go over budget and behind schedule.

It's a good sign if your language is sponsored by the government. COBOL had government backing, and Ada is being funded by the Department of Defense. After much negotiation, the Department of Sanitation has agreed to sponsor Babbage.

Babbage is based on language elements that were discovered after the design of Ada was completed. C. A. R. Hoare, in his 1980 ACM Turing Award lecture, told of two ways of constructing a software design: "One way is to make it so simple that there are obviously no deficiencies and the other way is to make it so complicated that there are no obvious deficiencies."

The designers of Babbage have chosen a third alternative - a language that has only obvious deficiencies. Babbage programs are so unreliable that

maintenance can begin before system integration is completed. This guarantees a steady increase in the programming job marketplace.

No subsets of Ada are allowed. Babbage is just the opposite. None of Babbage is defined except its extensibility - each user must define his own version. To end the debate of large languages versus small, Babbage allows each user to make the language any size he wants. Babbage is the ideal language for the "me" generation.

Like Pascal, Ada uses "strong typing" to avoid errors caused by mixing data types. The designers of Babbage advocate "good typing" to avoid errors caused by misspelling the words in your program. Later versions of Babbage will also support "touch typing," which will fill a long-felt need.

A hotly contested issue among language designers is the method for passing parameters to subfunctions. Some advocate "call by name," others prefer "call by value." Babbage uses a new method - "call by telephone." This is especially effective for long-distance parameter passing.

Ada stresses the concept of software portability. Babbage encourages hardware portability. After all, what good is a computer if you can't take it with you?

The examples that follow will give you some idea of what Babbage looks like.

Structured languages banned GOTOs and multi-way conditional branches by replacing them with the simpler IF-THEN-ELSE structure. Babbage has a number of new conditional statements that act like termites in the structure of your program:

WHAT IF - Used in simulation languages. Branches before evaluating test conditions.

OR ELSE - Conditional threat, as in: "Add these two numbers OR ELSE!"

WHY NOT? Executes the code that follows in a devil-may-care fashion.

WHO ELSE? - Used for polling during I/O operations.

ELSEWHERE - This is where your program really is when you think it's here.

GOING GOING GONE - For writing unstructured programs. Takes a random branch to another part of your program. Does the work of 10 GOTOs.

For years, programming languages have used "FOR," "DO UNTIL," "DO WHILE," etc. to mean "LOOP." Continuing with this trend, Babbage offers the following loop statements:

DON'T DO WHILE NOT - This loop is not executed if the test condition is not false (or if it's Friday afternoon).

DIDN'T DO - The loop executes once and hides all traces.

CAN'T DO - The loop is pooped.

WON'T DO - The CPU halts because it doesn't like the code inside the loop. Execution can be resumed by typing "May I" at the console.

MIGHT DO - Depends on how the CPU is feeling. Executed if the CPU is "up," not executed if the CPU is "down" or if its feelings have been hurt.

DO UNTO OTHERS - Used to write the main loop for timesharing systems so that they will antagonize the users in a uniform manner.

DO-WAH - Used to write timing loops for computer-generated music

(Rag Timing).

Every self-respecting language has a CASE statement to implement multi-way branching. ALGOL offers an indexed case statement and Pascal has a labeled case statement. Not much of a choice. Babbage offers a variety of interesting case statements:

The JUST-IN-CASE Statement - For handling afterthoughts and fudge factors. Allows you to multiply by zero to correct for accidentally dividing by zero.

The WORST CASE Statement - Takes the path that will do the most damage.

The BRIEF CASE Statement - To encourage portable software.

The OPEN-AND-SHUT CASE Statement - No proof of correctness is necessary with this one.

The IN-ANY-CASE Statement - This one always works.

The HOPELESS CASE Statement - This one never works.

The BASKET CASE Statement - A really hopeless case.

The Babbage Language Design Group is continuously evaluating new features that will keep its users from reaching any level of effectiveness. For instance, Babbage's designers are now considering the ALMOST EQUALS sign, used for comparing two floating-point numbers. This new feature "takes the worry out of being close."

No language, no matter how bad, can stand on its own. We need a really state-of-the-art operating system to support Babbage. After trying several commercial systems, we decided to write a "virtual" operating system. Everyone else has a virtual memory operating system so we decided to try something a little different.

Our new operating system is called the Virtual Time Operating System (VTOS). While virtual memory systems make the computer's memory the virtual resource, VTOS does the same thing with CPU processing time.

The result is that the computer can run an unlimited number of jobs, all at the same time. Like the virtual memory system, which actually keeps part of memory on disk, VTOS has to play tricks to achieve its goals. Although all of your jobs seem to be running right now, some of them are actually running next week.

As you can see, Babbage is still in its infancy. The Babbage Language Design Group is seeking suggestions for this powerful new language and as the sole member of this group (all applications for membership will be accepted), I call on the computing community for help in making this dream a reality.

Tony Karp - Jamaica, New York - 1981

Tony Karp, TLC Systems Corp tkarp@tlc-systems.com

Our web sites:

Techno-Impressionist Journal: <http://www.ti-journal.com>

Techno-Impressionist Museum: <http://www.techno-impressionist.com>

TLC Systems: <http://www.tlc-systems.com>

Last modified July 1, 2004