The evolution of Rexx

Rexx Symposium - Raleigh, 29 April 2002

Mike Cowlishaw
IBM Fellow

mfc @ uk.ibm.com

Overview

- Early days
- Language concepts and philosophy
- Development principles
- Questions?

Whence Rexx?

- Two core concepts:
  - A *single* macro language for *many* applications
  - A language designed for the benefit of the *user*, not the *language implementer*

Traditional macro languages

- Macro languages assumed that most of the content of a program would be literal data:

```
&IF &NODE&J ¬= &LOCAL &USER = &STRING OF &USER&J AT &NODE&J
```

- By 1979, programs existed where more than 50% of the tokens began with &. The solution:

```
if node.j ¬= local then user = user.j 'AT' node.j
```

(March 20, 1979)

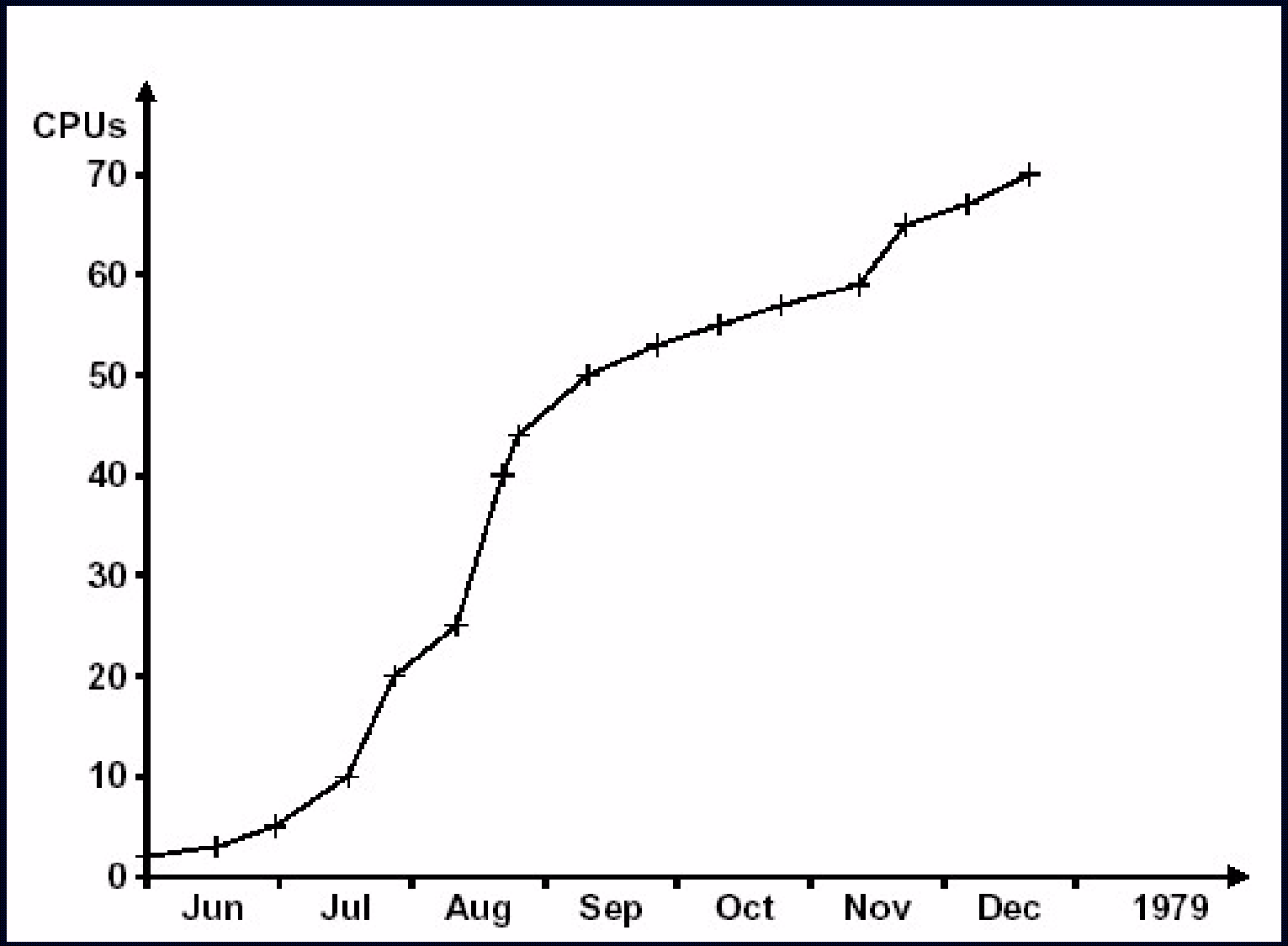
e-mail, 1979.03.22

"... I'm thinking of implementing an experimental EXEC processor to handle a more ... PL/I-like language. ... This is of course the dual of the EXEC/EXEC 2 languages, in that literals are identified, rather than variables/control words, but ... EXECs nowadays often seem as complex as programs ... and that therefore literals are often a very small percent of the tokens in an EXEC"

Timeline, 1979

- March: Initial Specification (10 pages + examples)
- May/June: First implementation (30-page manual)
- August: 'VM News' mailing list
- December: FSX and an animated Xmas card...
  - "It is spectacular ... it has swept through our installation this morning. I put it on a subsystem disk and everybody is telling everybody else to type TRYTHIS"

Growth chart



Ingredients

- Lots of feedback and ideas from users
  - At peak, 350 e-mail a day
- 10,000 lines of code and 5,000 of documentation
- 1,000 hours in first year, 4,000 total
- Only evenings and some weekends
  - few interruptions
  - good response time (machines were slow!)

How slow?

- Test loop:

```
i=0
do 2000
  i=i+2
  end
```

- IBM S/370 model 155 1979: 3.31 seconds
- 800MHz laptop 2002: 0.0013 seconds

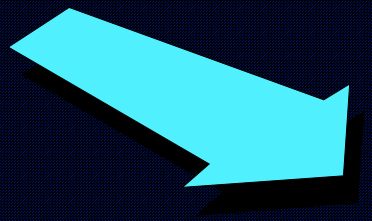
(2,546 x)

REXX language philosophies

- Ground rule: ***A user's time is more important than implementation time or computer time***

Readability

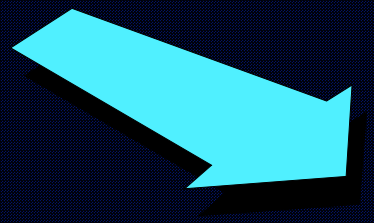
- Perceived legibility: tokens are familiar
  - minimal punctuation and boilerplate
- Free format: layout can be familiar, meaningful, and structured



fewer errors

Few notations

- Keywords and function names are 'real words'
- Most special characters are used conventionally

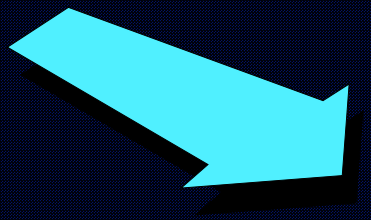


easier to learn

easier to remember

Natural data typing

- Only one data type; strings of characters
  - rich set of string operations and functions
- Nothing to Declare

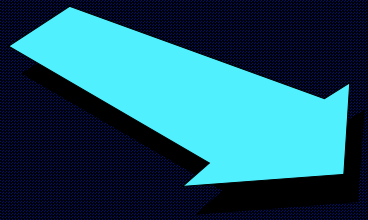


simplifies programming

increases portability

Decimal arithmetic

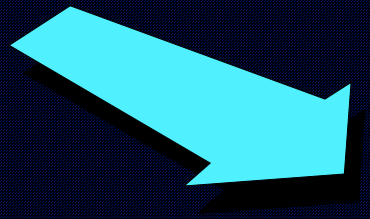
- Matches user model of arithmetic
- No binary artifacts
- Hardware independent



simplifies programming

No limits

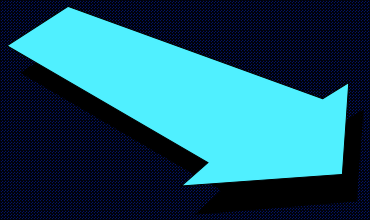
- No language limits on size of strings, size of numbers, or size of programs
- Implementations usually only limited by available memory



simplifies programming

Keep the language small

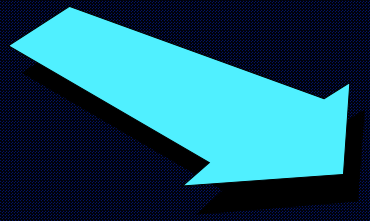
- Few special cases
- Compact documentation



easier to learn and use

Dynamic scoping

- Well matched to our human procedural model
- Easy to modify programs

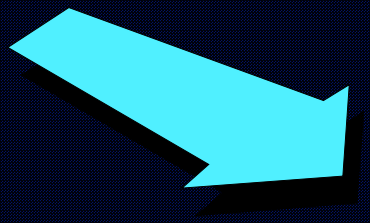


rapid development

low human overhead

No reserved keywords

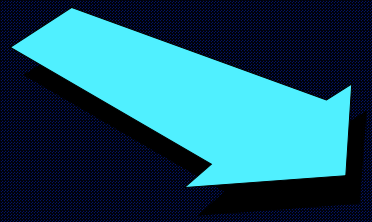
- No need to learn every keyword before you can safely write a program
- Programs, especially macros and scripts, are robust against changes in the language or applications



lower costs

Dealing with reality

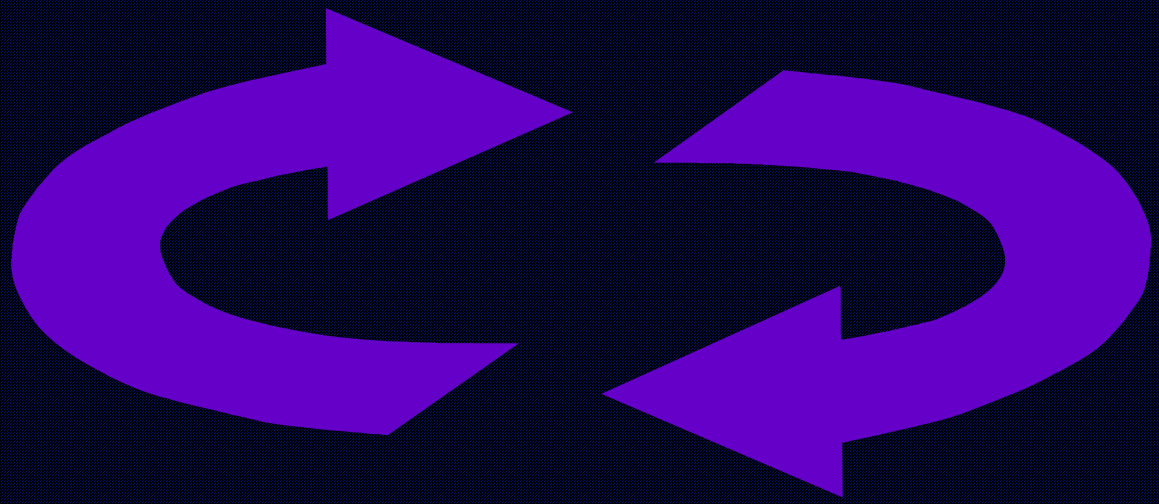
- Usability does ***not*** necessarily follow from elegant design; human expectations must be met
- Optional restrictions support writing robust programs



a tool for real work

REXX development principles

- Ground rule: ***get feedback from users***



Telecommunications

- Designing for people means you need feedback from ***many*** different people
- Only practical electronically

VNET

- Rexx was designed in the UK, with most users in the USA; impossible without the electronic network
- Hundreds of users from the start; rich feedback for problems and changes
- ... but users soon built up an investment in existing programs ...

The user is *always* right

- Simplest to express; hardest to follow
- Any confusion, question, or suggestion shows there is a problem. Not with the user, but with the program or documentation
- Mail review is a powerful technique, rarely used

Documentation first

- Documentation before implementation, ***to final draft quality***
- Problems discovered early
- Improved review process (feedback!)
- Much less influence of implementation on design and documentation

And finally ...

... Rexx gets everywhere

Questions?