HIDDEN MAINSTREAM: THE MAINFRAME LANGUAGES

DR. VADIM ZAYTSEV

WORKSHOP ON PROGRAMMING RESEARCH IN MAINSTREAM LANGUAGES

(PRIML @ LICS/ICALP 2020)

Dr. Vadim Zaytsev aka @grammarware

- Worked in research (CWI, VU, Koblenz)
  - software evolution
  - software languages (PL+)
  - grammars in a broad sense
- Worked in industry (Raincode, Raincode Labs)
  - legacy systems
  - software migration
  - mainframe to cloud native
- http://grammarware.net, http://twitter.com/grammarware, …

RAINCODE

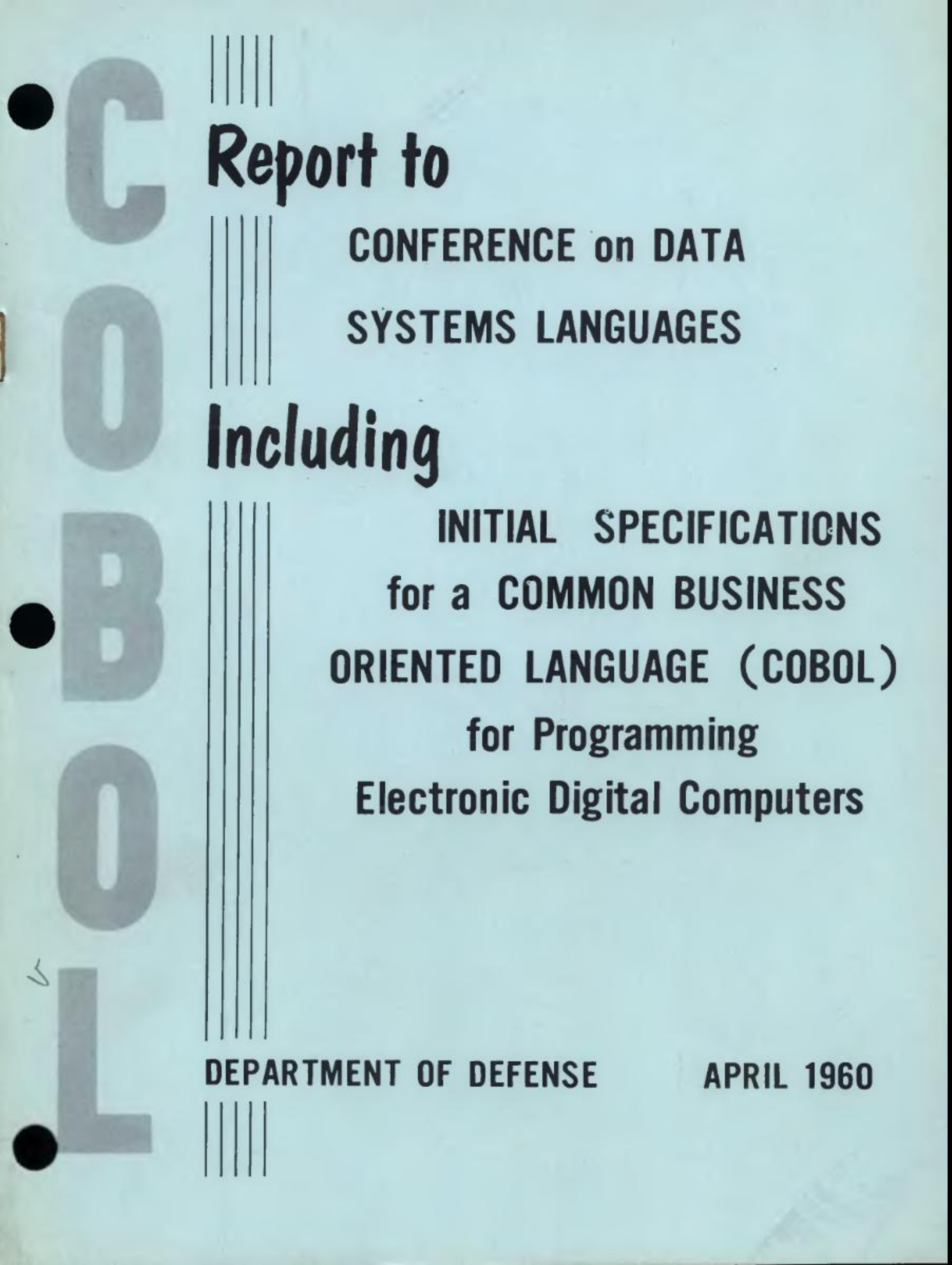
UNIVERSITY OF TWENTE.

IBM S/360

In use since 1964

⇒ Mainframe Languages

UNIVERSITY OF TWENTE.



COBOL

Report to

CONFERENCE on DATA SYSTEMS LANGUAGES

Including

INITIAL SPECIFICATIONS for a COMMON BUSINESS ORIENTED LANGUAGE (COBOL) for Programming Electronic Digital Computers

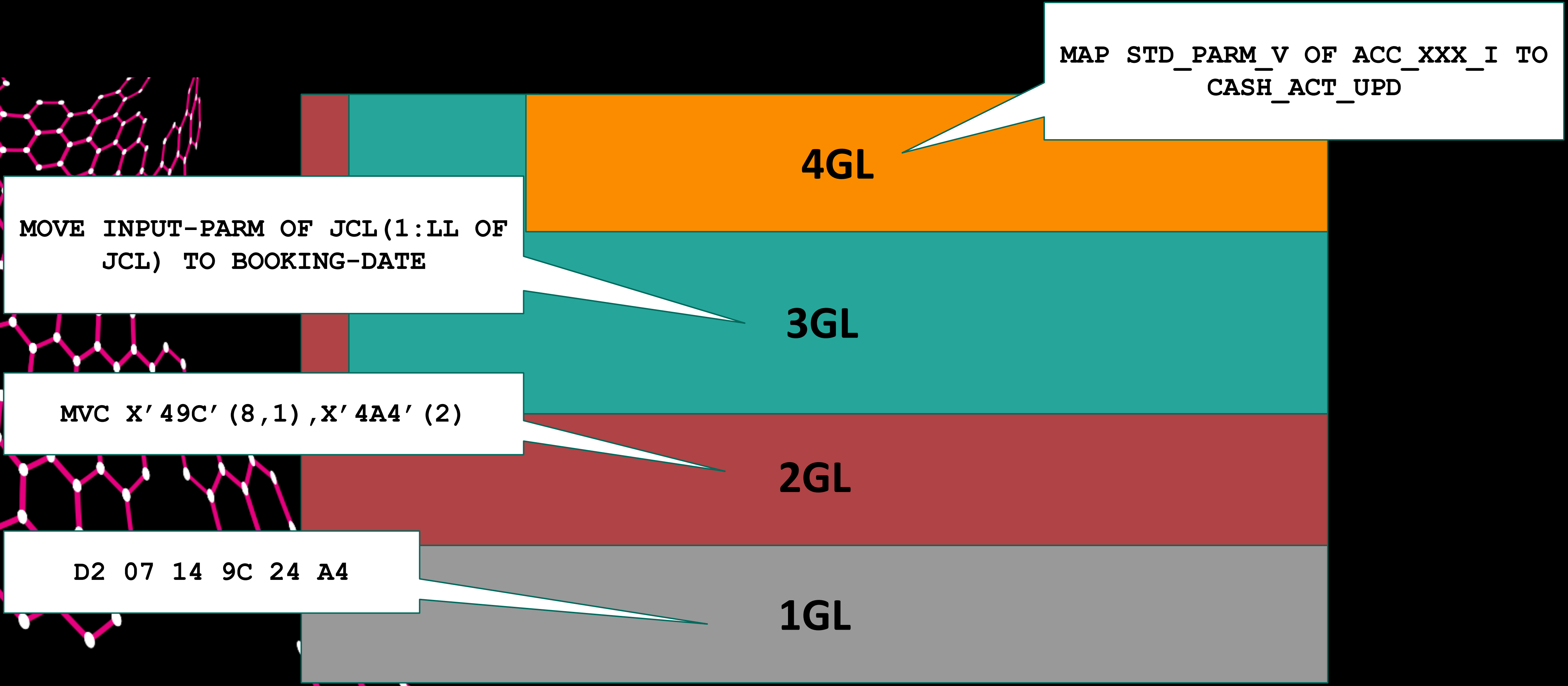
DEPARTMENT OF DEFENSE APRIL 1960

COBOL

- Language designed by committee
  - tech advisor: Grace Murray Hopper
- Based on ideas of Grace Hopper
  - and Bob Bemer (COMTRAN)
- Extremely verbose
  - looks like English
  - surprisingly readable!
- Hard to find empirical fodder
  - little open source
  - closed source portfolios are huge
- Hard to implement
  - large grammar
  - complex semantics

UNIVERSITY OF TWENTE.

LANGUAGE GENERATIONS



```
MAP STD_PARM_V OF ACC_XXX_I TO
CASH_ACT_UPD
```

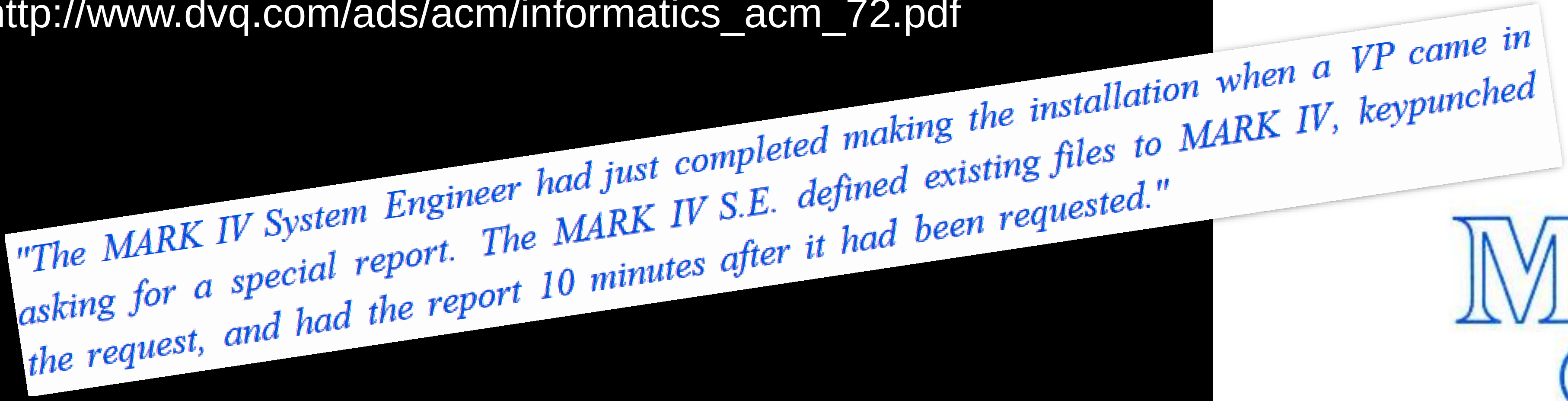
```
MOVE INPUT-PARM OF JCL(1:LL OF
JCL) TO BOOKING-DATE
```

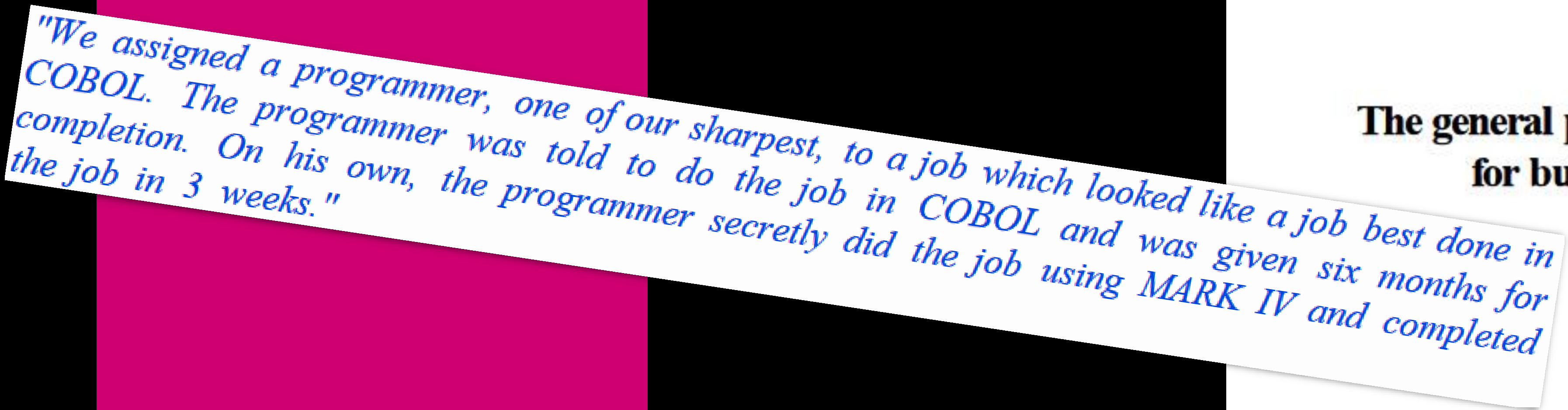
```
MVC X'49C'(8,1),X'4A4'(2)
```

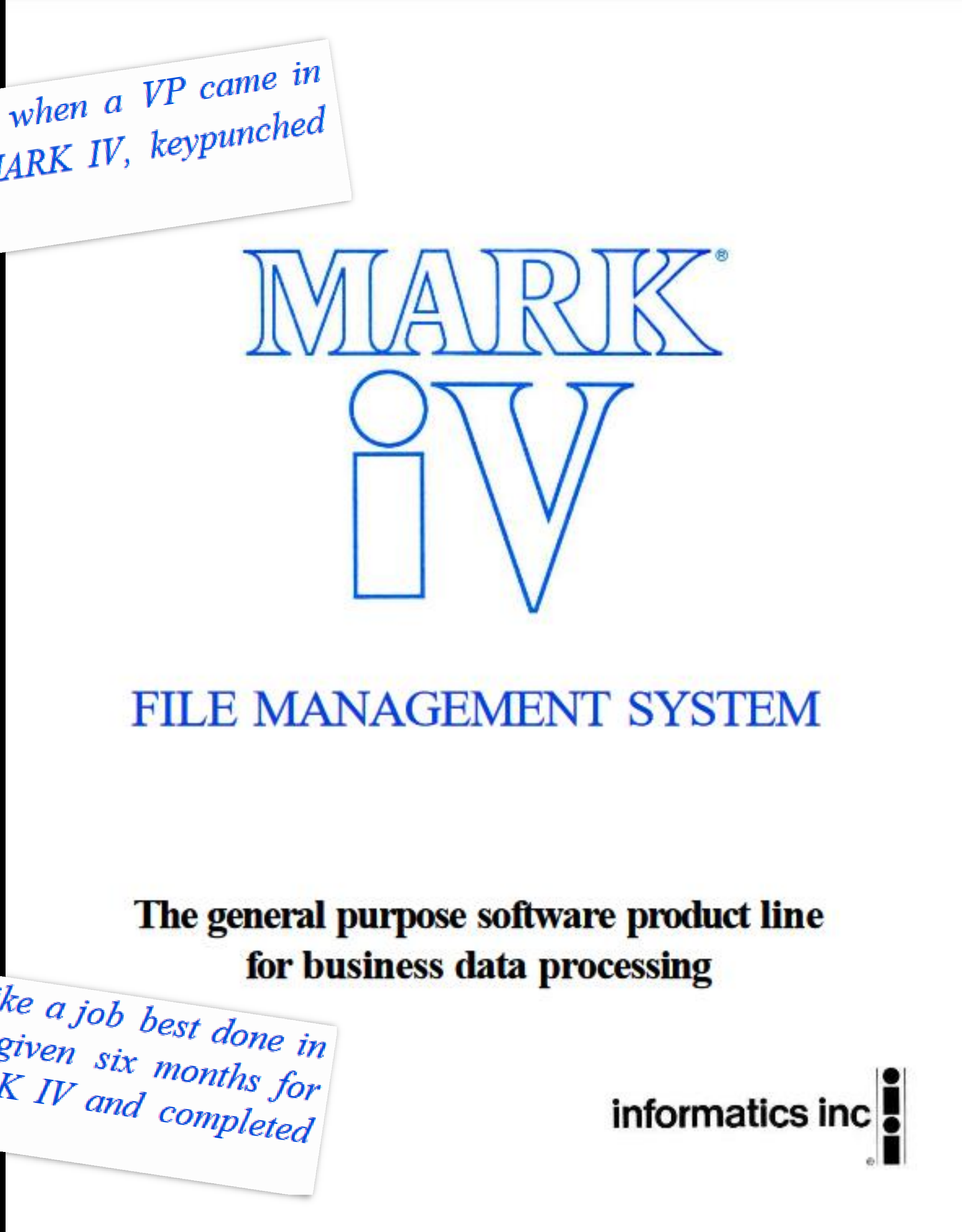
```
D2 07 14 9C 24 A4
```

http://www.dvq.com/ads/acm/informatics_acm_72.pdf

1960s


"The MARK IV System Engineer had just completed making the installation when a VP came in asking for a special report. The MARK IV S.E. defined existing files to MARK IV, keypunched the request, and had the report 10 minutes after it had been requested."


"We assigned a programmer, one of our sharpest, to a job which looked like a job best done in COBOL. The programmer was told to do the job in COBOL and was given six months for completion. On his own, the programmer secretly did the job using MARK IV and completed the job in 3 weeks."


MARK iV®

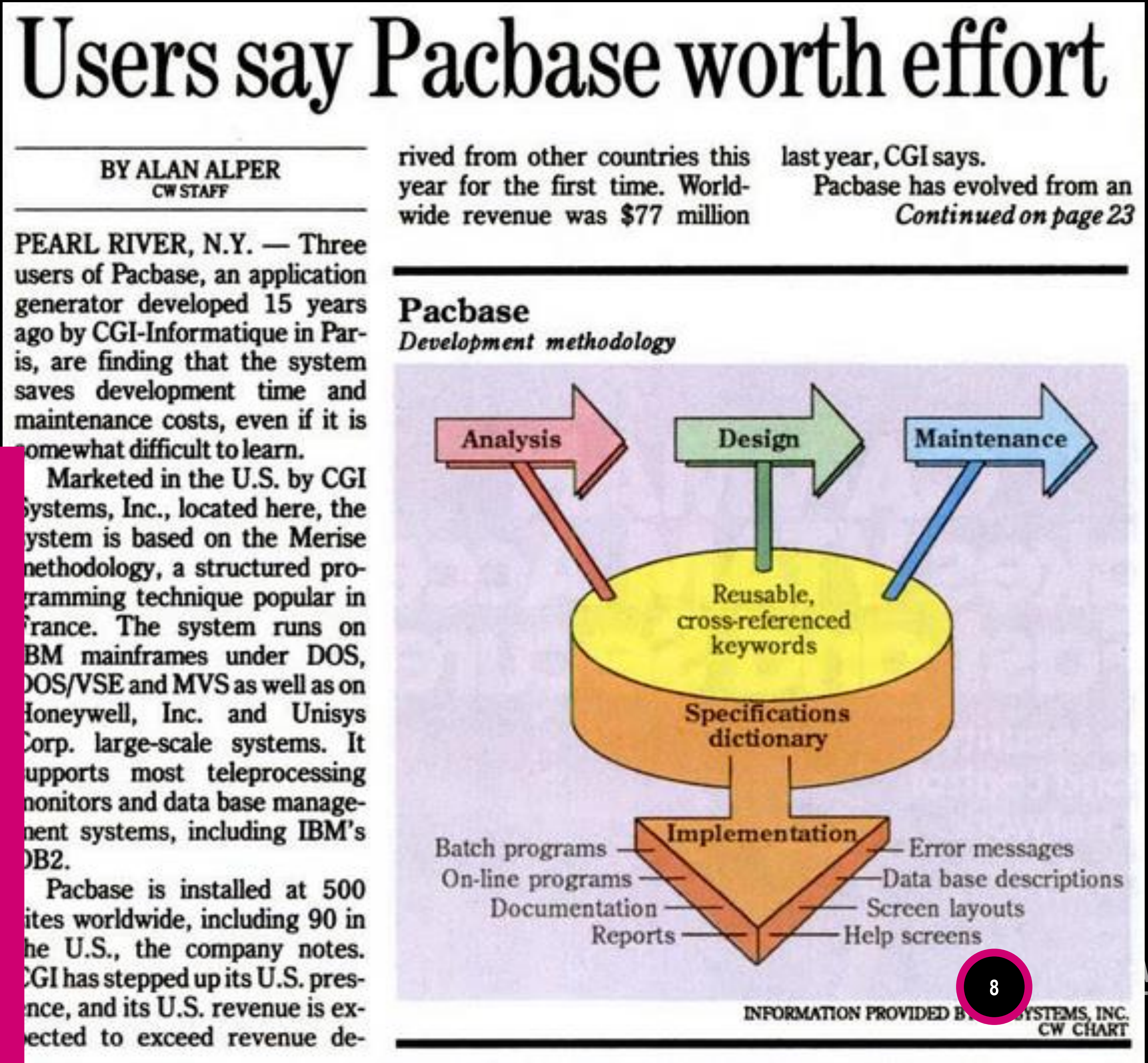
FILE MANAGEMENT SYSTEM

The general purpose software product line for business data processing

informatics inc

Computerworld — Aug 17, 1987

1970s



Users say Pacbase worth effort

BY ALAN ALPER
CW STAFF

PEARL RIVER, N.Y. — Three users of Pacbase, an application generator developed 15 years ago by CGI-Informatique in Paris, are finding that the system saves development time and maintenance costs, even if it is omewhat difficult to learn.

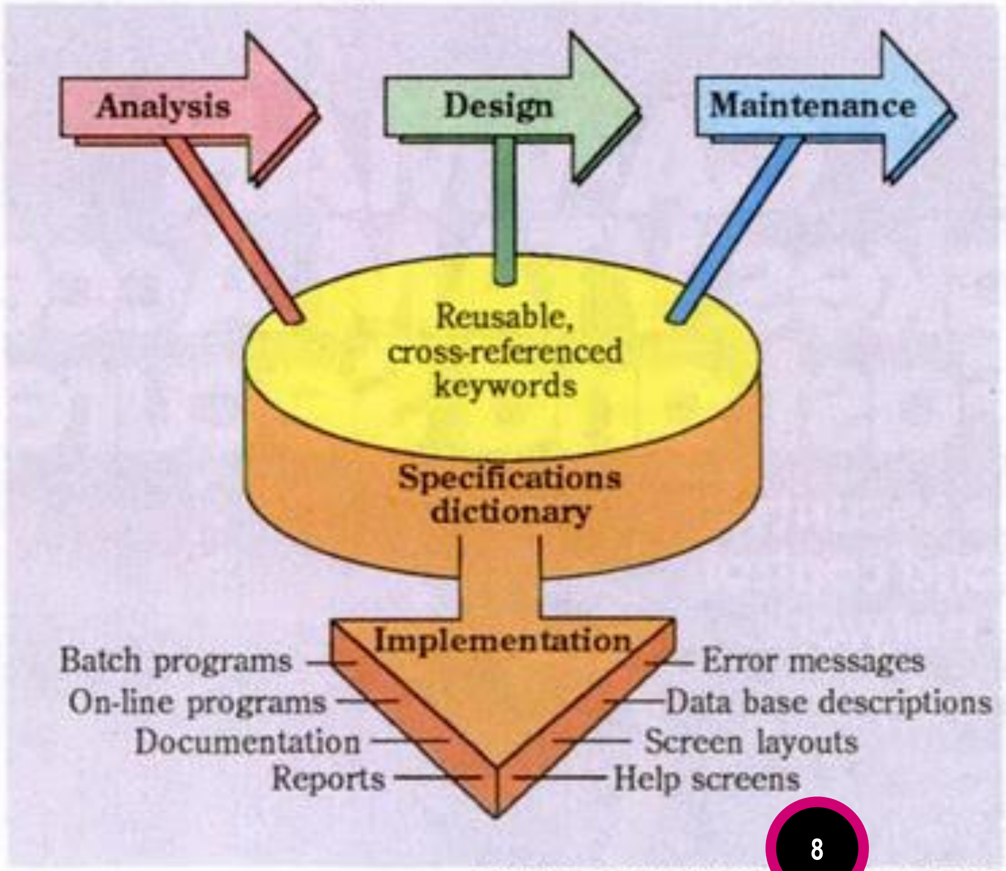
Marketed in the U.S. by CGI ystems, Inc., located here, the ystem is based on the Merise nethodology, a structured proramming technique popular in rance. The system runs on BM mainframes under DOS,)OS/VSE and MVS as well as on Ioneywell, Inc. and Unisys Corp. large-scale systems. It upports most teleprocessing nonitors and data base managenent systems, including IBM's)B2.

Pacbase is installed at 500 ites worldwide, including 90 in he U.S., the company notes. CGI has stepped up its U.S. presnce, and its U.S. revenue is expected to exceed revenue derived from other countries this year for the first time. Worldwide revenue was $77 million last year, CGI says.

Pacbase has evolved from an

Continued on page 23

Pacbase
Development methodology

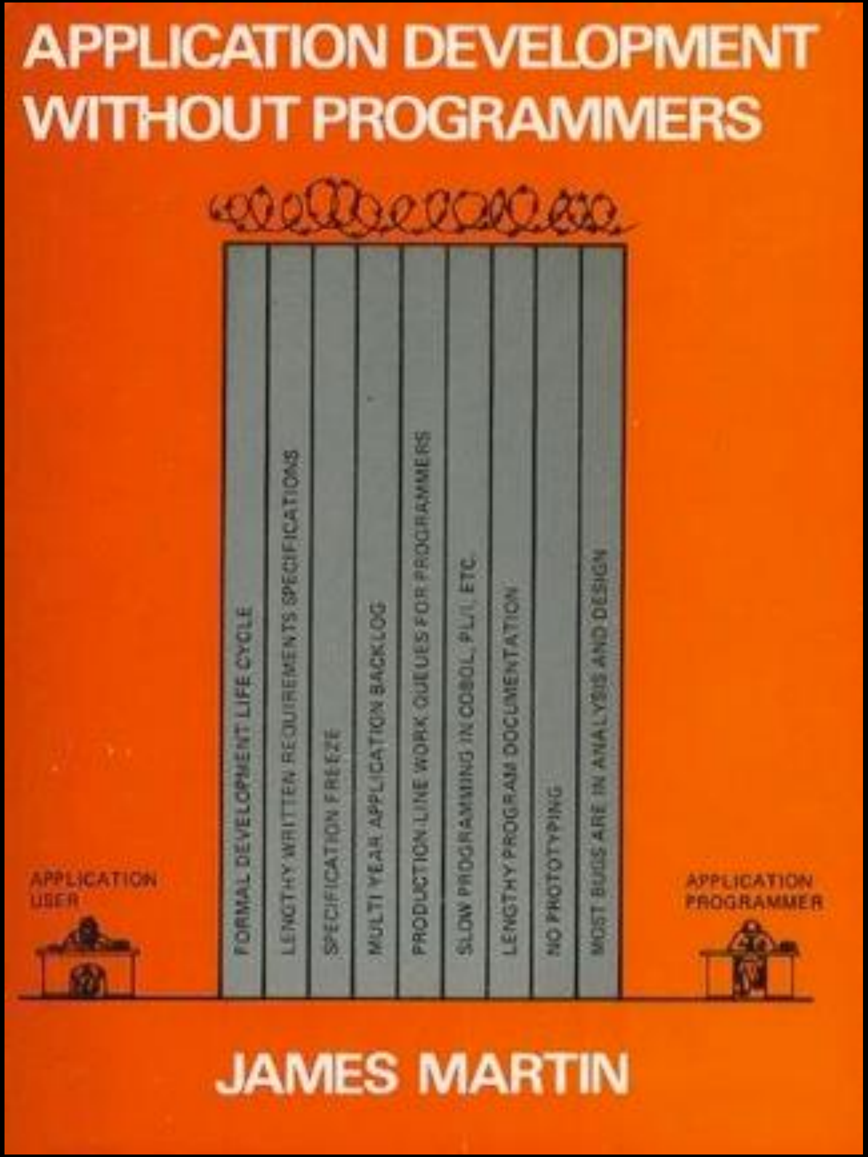


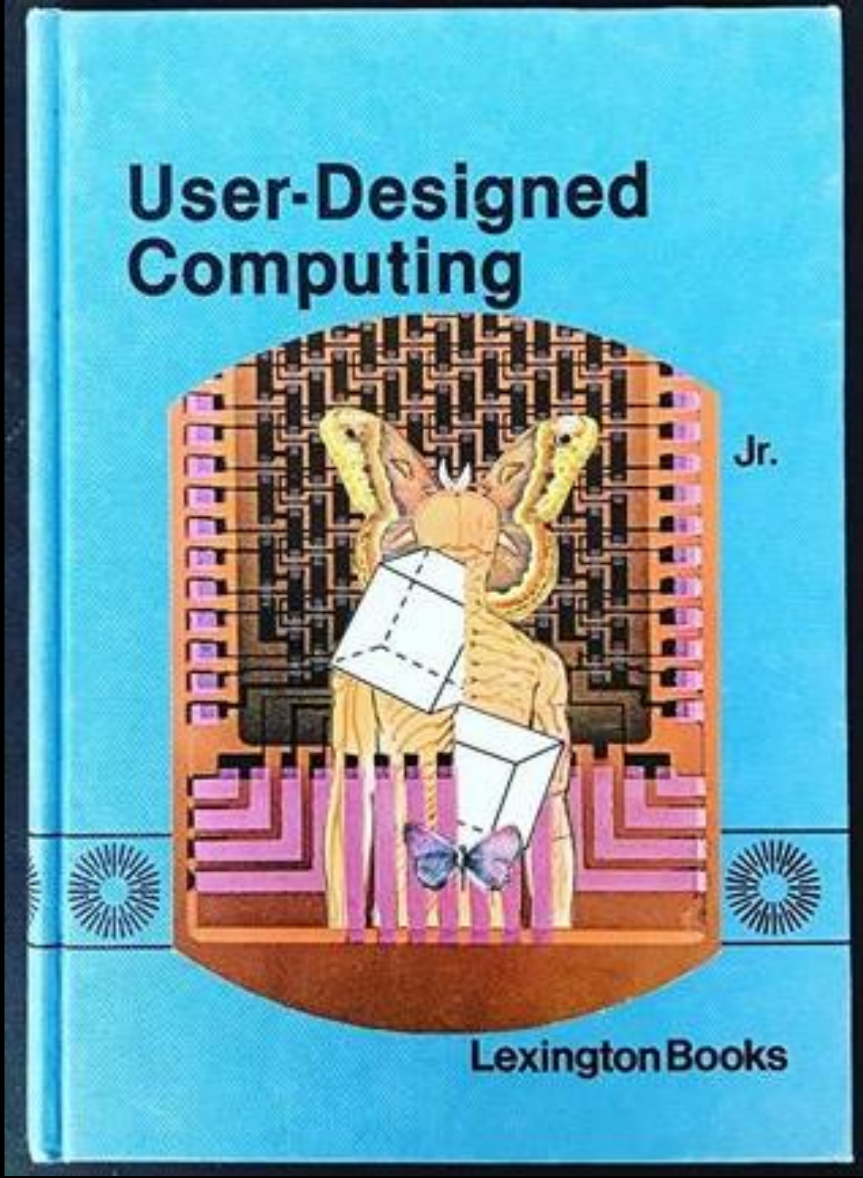
INFORMATION PROVIDED B YSTEMS, INC.
CW CHART

James Martin, Applications Development Without Programmers, 1981
Louis Schlueter, User-Designed Computing: The Next Generation, 1988

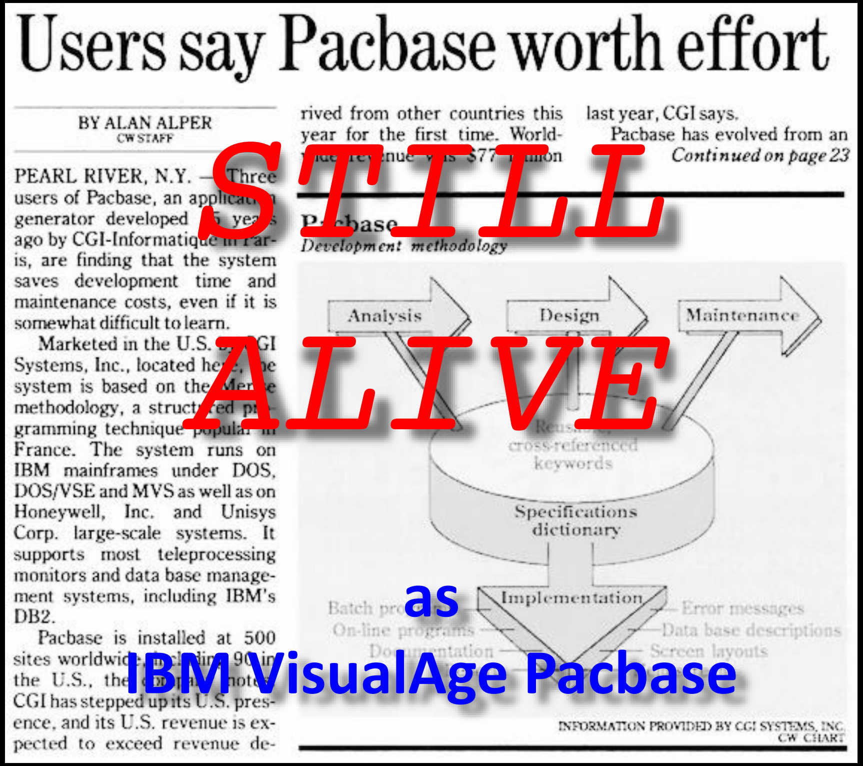
1980s

- 12 pages of COBOL
- 2 pages of Mark IV
- 1 statement in Nomad


APPLICATION DEVELOPMENT WITHOUT PROGRAMMERS
FORMAL DEVELOPMENT LIFE CYCLE
LENGTHY WRITTEN REQUIREMENTS SPECIFICATIONS
SPECIFICATION FREEZE
MULTI YEAR APPLICATION BACKLOG
PRODUCTION-LINE WORK QUEUES FOR PROGRAMMERS
SLOW PROGRAMMING IN COBOL, PL/I, ETC.
LENGTHY PROGRAM DOCUMENTATION
NO PROTOTYPING
MOST BUGS ARE IN ANALYSIS AND DESIGN
APPLICATION USER
APPLICATION PROGRAMMER
JAMES MARTIN



2020?



Users say Pacbase worth effort

BY ALAN ALPER
CW STAFF

PEARL RIVER, N.Y. — Three users of Pacbase, an applicat[illegible] generator developed [illegible] years ago by CGI-Informatique [illegible] Paris, are finding that the system saves development time and maintenance costs, even if it is somewhat difficult to learn.

Marketed in the U.S. [illegible] CGI Systems, Inc., located he[illegible], [illegible] system is based on the Mer[illegible]e methodology, a struct[illegible]ed pr[illegible]gramming technique popular in France. The system runs on IBM mainframes under DOS, DOS/VSE and MVS as well as on Honeywell, Inc. and Unisys Corp. large-scale systems. It supports most teleprocessing monitors and data base management systems, including IBM's DB2.

Pacbase is installed at 500 sites worldwid[illegible] 90 in the U.S., the [illegible] not[illegible] CGI has stepped up its U.S. presence, and its U.S. revenue is expected to exceed revenue derived from other countries this year for the first time. Worldwide revenue [illegible] $77 [illegible] last year, CGI says.

Pacbase has evolved from an

Continued on page 23

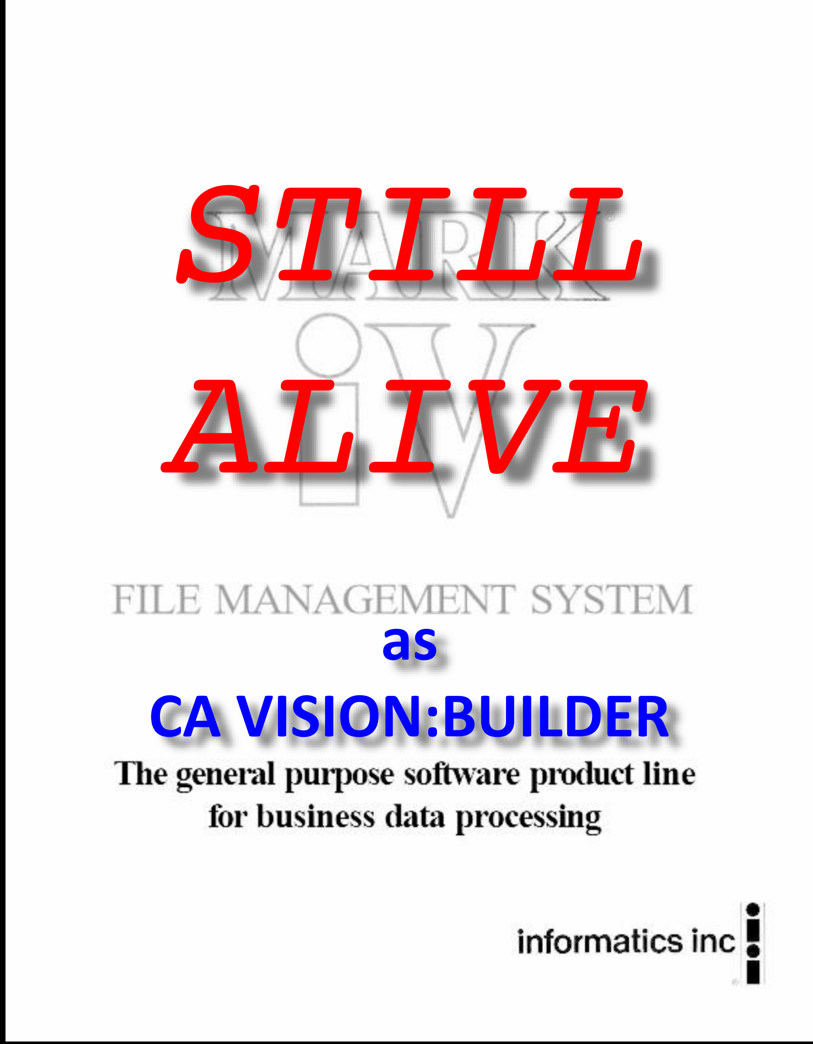
P[illegible]base
Development methodology

INFORMATION PROVIDED BY CGI SYSTEMS, INC.
CW CHART

STILL ALIVE

as

IBM VisualAge Pacbase



STILL ALIVE

FILE MANAGEMENT SYSTEM

as

CA VISION:BUILDER

The general purpose software product line for business data processing

informatics inc

UNIVERSITY OF TWENTE.

What about COBOL?

- 43% of banking systems built on COBOL [Reuters 2017]
- 15% of new applications built in COBOL [Gartner 2003]
- 75% of business data processed by COBOL [Gartner 2003]
- 80% of in-person transactions run COBOL [Reuters 2017]
- 95% ATM swipes rely on COBOL code [Reuters 2017]
- 180–200 billion LOC of COBOL in use [Gartner 2003]
  - 220 billion LOC in use [Reuters 2017]
  - one codebase up to 250 MLOC [Bankia'20], 343 MLOC [NYMellon'12]
- Replacement costs at $25 per line [Tactical Strategy Group]
- 1 COBOL app costs $5M/year [Micro Focus 2003]
  - 4000 MIPS will cost $6–16M/year [Raincode 2020]

https://thenewstack.io/cobol-everywhere-will-maintain/ (2017),
https://beyondparsing.com/interview-with-vadim-zaytsev/ (2020), etc

COBOL

Report to
CONFERENCE on DATA
SYSTEMS LANGUAGES

Including
INITIAL SPECIFICATIONS
for a COMMON BUSINESS
ORIENTED LANGUAGE (COBOL)
for Programming
Electronic Digital Computers

DEPARTMENT OF DEFENSE APRIL 1960

COBOL QUOTES:

- "it's only a matter of time before all the existing COBOL programmers die of old age" [Yourdon 1996]
  - *"maybe it will all be outsourced to some part of the world where COBOL maintenance programming is considered a pleasant alternative to growing rice or raising pigs"* [Yourdon 1996]

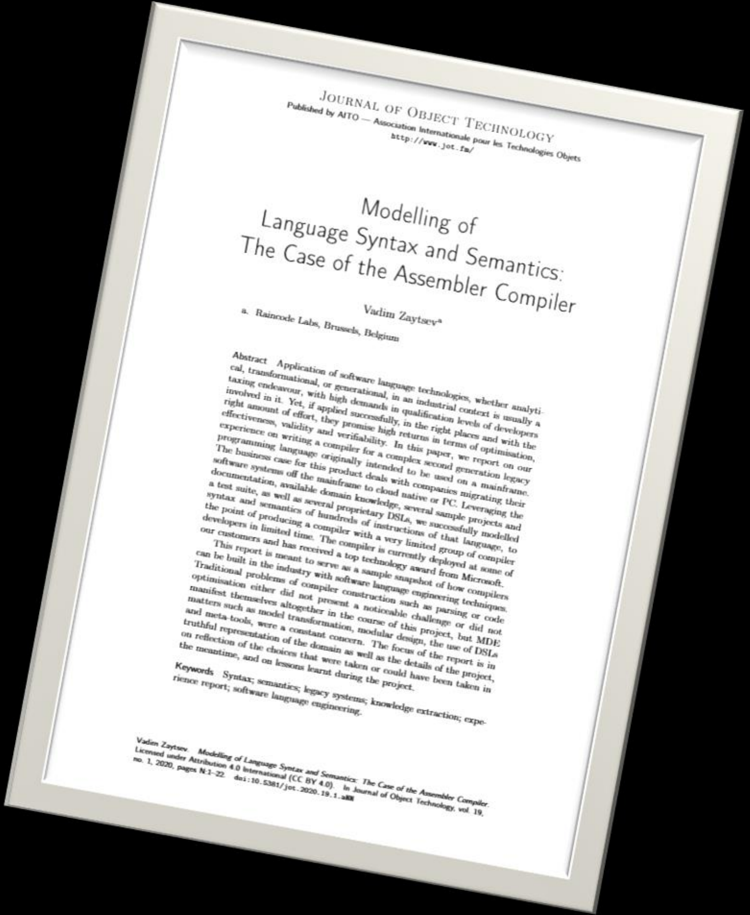
- "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offence." [Dijkstra 1975]

- However, research on COBOL is not! [Lämmel 2004]

cf.: https://www.cs.vu.nl/Cobol/stop-bashing-cobol.pdf

Takeaway #1:

- Legacy languages matter
- Software written in them runs your life
- 2GLs like HLASM
- 3GLs like COBOL, PL/I, CLIST, REXX, RPG, FORTRAN
- 4GLs like Pacbase, AppBuilder, IDEAL, VISION:BUILDER



Journal of Object Technology
Published by AITO — Association Internationale pour les Technologies Objets
http://www.jot.fm/

Modelling of
Language Syntax and Semantics:
The Case of the Assembler Compiler

Vadim Zaytsev[a]
a. Raincode Labs, Brussels, Belgium

Abstract Application of software language technologies, whether analytical, transformational, or generational, in an industrial context is usually a taxing endeavour, with high demands in qualification levels of developers involved in it. Yet, if applied successfully, in the right places and with the right amount of effort, they promise high returns in terms of optimisation, effectiveness, validity and verifiability. In this paper, we report on our experience on writing a compiler for a complex second generation legacy programming language originally intended to be used on a mainframe. The business case for this product deals with companies migrating their software systems off the mainframe to cloud native or PC. Leveraging the documentation, available domain knowledge, several sample projects and a test suite, as well as several proprietary DSLs, we successfully modelled syntax and semantics of hundreds of instructions of that language, to the point of producing a compiler with a very limited group of compiler developers in limited time. The compiler is currently deployed at some of our customers and has received a top technology award from Microsoft.

This report is meant to serve as a sample snapshot of how compilers can be built in the industry with software language engineering techniques. Traditional problems of compiler construction such as parsing or code optimisation either did not present a noticeable challenge or did not manifest themselves altogether in the course of this project, but MDE matters such as model transformation, modular design, the use of DSLs and meta-tools, were a constant concern. The focus of the report is in truthful representation of the domain as well as the details of the project, on reflection of the choices that were taken or could have been taken in the meantime, and on lessons learnt during the project.

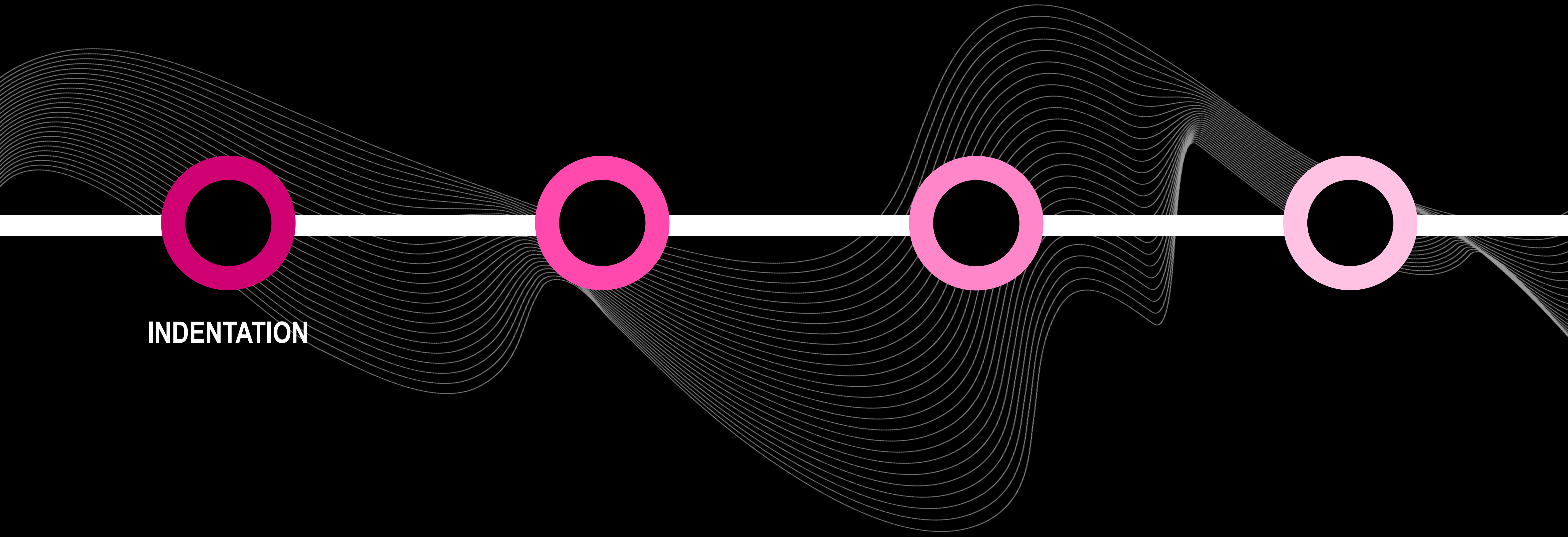
Keywords Syntax; semantics; legacy systems; knowledge extraction; experience report; software language engineering.

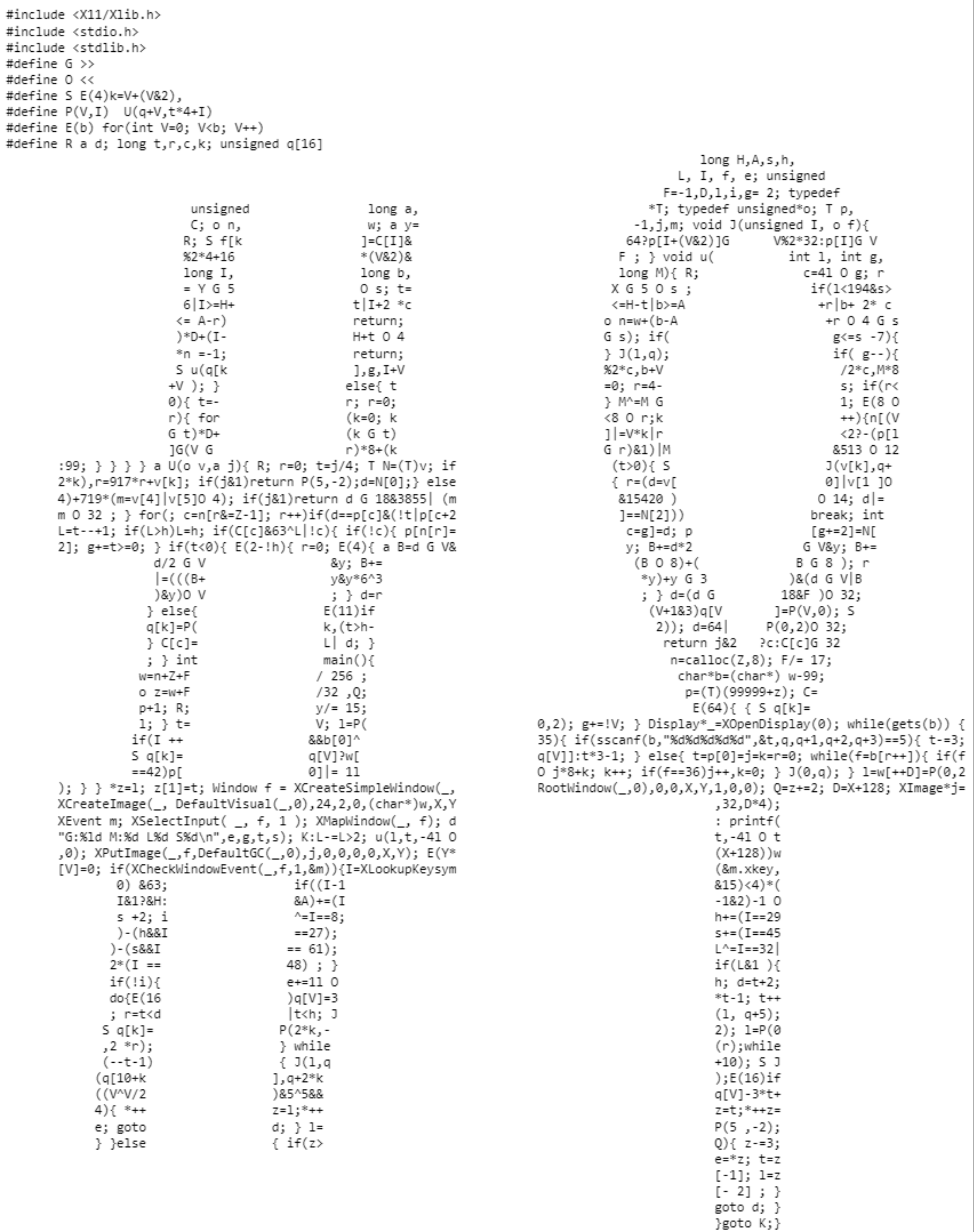
Vadim Zaytsev. *Modelling of Language Syntax and Semantics: The Case of the Assembler Compiler.* Licensed under Attribution 4.0 International (CC BY 4.0). In Journal of Object Technology, vol. 19, no. 1, 2020, pages N:1–22. doi:10.5381/jot.2020.19.1.a[illegible]

⇒ Language features to be afraid of

UNIVERSITY OF TWENTE.

Language features to be afraid of:





```c
#include <X11/Xlib.h>
#include <stdio.h>
#include <stdlib.h>
#define G >>
#define O <<
#define S E(4)k=V+(V&2),
#define P(V,I)  U(q+V,t*4+I)
#define E(b) for(int V=0; V<b; V++)
#define R a d; long t,r,c,k; unsigned q[16]



                         unsigned                long a,
                         C; o n,                 w; a y=
                        R; S f[k                ]=C[I]&
                        %2*4+16                 *(V&2)&
                        long I,                 long b,
                        = Y G 5                 O s; t=
                        6|I>=H+                t|I+2 *c
                       <= A-r)                 return;
                       )*D+(I-                 H+t O 4
                       *n =-1;                 return;
                       S u(q[k                 ],g,I+V
                      +V ); }                 else{ t
                      0){ t=-                 r; r=0;
                      r){ for                 (k=0; k
                      G t)*D+                 (k G t)
                      ]G(V G                  r)*8+(k
       :99; } } } } a U(o v,a j){ R; r=0; t=j/4; T N=(T)v; if
       2*k),r=917*r+v[k]; if(j&1)return P(5,-2);d=N[0];} else
       4)+719*(m=v[4]|v[5]O 4); if(j&1)return d G 18&3855| (m
       m O 32 ; } for(; c=n[r&=Z-1]; r++)if(d==p[c]&(!t|p[c+2
       L=t--+1; if(L>h)L=h; if(C[c]&63^L|!c){ if(!c){ p[n[r]=
       2]; g+=t>=0; } if(t<0){ E(2-!h){ r=0; E(4){ a B=d G V&
                    d/2 G V                 &y; B+=
                    |=(((B+                 y&y*6^3
                    )&y)O V                 ; } d=r
                   } else{                 E(11)if
                   q[k]=P(                 k,(t>h-
                   } C[c]=                 L| d; }
                   ; } int                 main(){
                  w=n+Z+F                 / 256 ;
                  o z=w+F                 /32 ,Q;
                  p+1; R;                 y/= 15;
                  l; } t=                 V; l=P(
                 if(I ++                 &&b[0]^
                 S q[k]=                 q[V]?w[
                 ==42)p[                 0]|= 11
       ); } } *z=l; z[1]=t; Window f = XCreateSimpleWindow(_,
       XCreateImage(_, DefaultVisual(_,0),24,2,0,(char*)w,X,Y
       XEvent m; XSelectInput( _, f, 1 ); XMapWindow(_, f); d
       "G:%ld M:%d L%d S%d\n",e,g,t,s); K:L-=L>2; u(l,t,-41 O
       ,0); XPutImage(_,f,DefaultGC(_,0),j,0,0,0,0,X,Y); E(Y*
       [V]=0; if(XCheckWindowEvent(_,f,1,&m)){I=XLookupKeysym
               0) &63;                 if((I-1
               I&1?&H:                 &A)+=(I
               s +2; i                 ^=I==8;
               )-(h&&I                 ==27);
              )-(s&&I                 == 61);
              2*(I ==                 48) ; }
              if(!i){                 e+=11 O
              do{E(16                 )q[V]=3
              ; r=t<d                 |t<h; J
             S q[k]=                 P(2*k,-
             ,2 *r);                 } while
             (--t-1)                 { J(l,q
            (q[10+k                 ],q+2*k
            ((V^V/2                 )&5^5&&
            4){ *++                 z=l;*++
            e; goto                 d; } l=
            } }else                 { if(z>
```

```c
                      long H,A,s,h,
                   L, I, f, e; unsigned
                 F=-1,D,l,i,g= 2; typedef
               *T; typedef unsigned*o; T p,
             -1,j,m; void J(unsigned I, o f){
            64?p[I+(V&2)]G      V%2*32:p[I]G V
           F ; } void u(          int l, int g,
           long M){ R;              c=41 O g; r
          X G 5 O s ;                if(l<194&s>
          <=H-t|b>=A                  +r|b+ 2* c
         o n=w+(b-A                    +r O 4 G s
         G s); if(                      g<=s -7){
         } J(l,q);                      if( g--){
         %2*c,b+V                        /2*c,M*8
         =0; r=4-                        s; if(r<
         } M^=M G                        1; E(8 O
         <8 O r;k                        ++){n[(V
         ]|=V*k|r                        <2?-(p[l
         G r)&1)|M                      &513 O 12
          (t>0){ S                     J(v[k],q+
          { r=(d=v[                    0]|v[1 ]O
           &15420 )                   O 14; d|=
           ]==N[2]))                 break; int
            c=g]=d; p                [g+=2]=N[
            y; B+=d*2               G V&y; B+=
             (B O 8)+(             B G 8 ); r
              *y)+y G 3           )&(d G V|B
              ; } d=(d G         18&F )O 32;
               (V+1&3)q[V       ]=P(V,0); S
                2)); d=64|     P(0,2)O 32;
                 return j&2   ?c:C[c]G 32
                  n=calloc(Z,8); F/= 17;
                   char*b=(char*) w-99;
                    p=(T)(99999+z); C=
                     E(64){ { S q[k]=
0,2); g+=!V; } Display*_=XOpenDisplay(0); while(gets(b)) {
35){ if(sscanf(b,"%d%d%d%d%d",&t,q,q+1,q+2,q+3)==5){ t-=3;
q[V]]:t*3-1; } else{ t=p[0]=j=k=r=0; while(f=b[r++]){ if(f
O j*8+k; k++; if(f==36)j++,k=0; } J(0,q); } l=w[++D]=P(0,2
RootWindow(_,0),0,0,X,Y,1,0,0); Q=z+=2; D=X+128; XImage*j=
                        ,32,D*4);
                        : printf(
                        t,-41 O t
                        (X+128))w
                        (&m.xkey,
                        &15)<4)*(
                        -1&2)-1 O
                        h+=(I==29
                        s+=(I==45
                        L^=I==32|
                        if(L&1 ){
                        h; d=t+2;
                        *t-1; t++
                        (l, q+5);
                        2); l=P(0
                        (r);while
                        +10); S J
                        );E(16)if
                        q[V]-3*t+
                        z=t;*++z=
                        P(5 ,-2);
                        Q){ z-=3;
                        e=*z; t=z
                        [-1]; l=z
                        [- 2] ; }
                        goto d; }
                        }goto K;}
```

Free formatting

- C
- C++
- C#
- Java
- JavaScript
- …
- https://www.ioccc.org/2019/dogon/prog.c [Dogon, IOCCC 2019]

UNIVERSITY OF TWENTE.

```
f x = let a = w x
      in if cond1 x
         then a
         else if cond2 x
              then g a
              else f (h x a)
```

ALIGN HOMOGENEOUS PARTS

- Haskell
- Kotlin

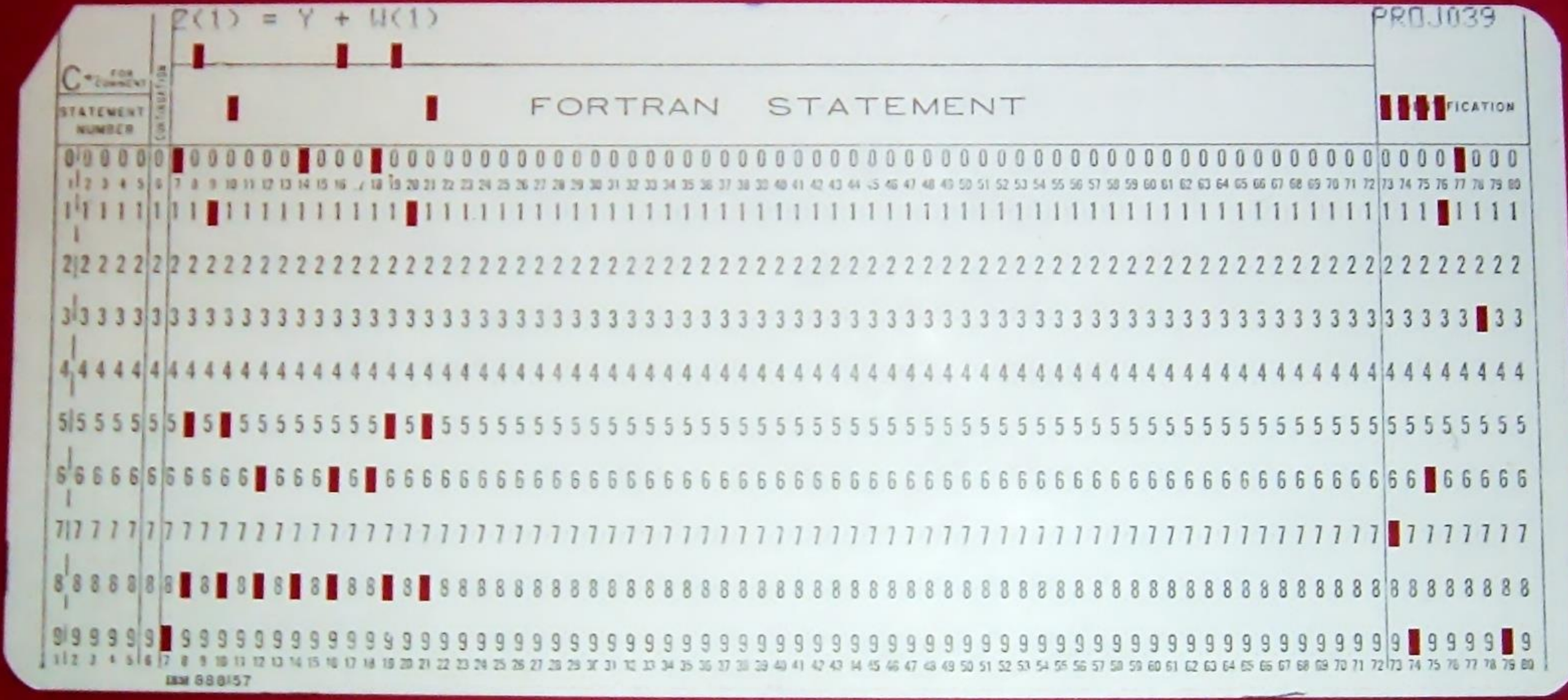
- [Landin 1966]

```
class Pony(flyer: CanFly, walker: CanWalk) :
        CanFly by flyer,
        CanWalk by walker
```

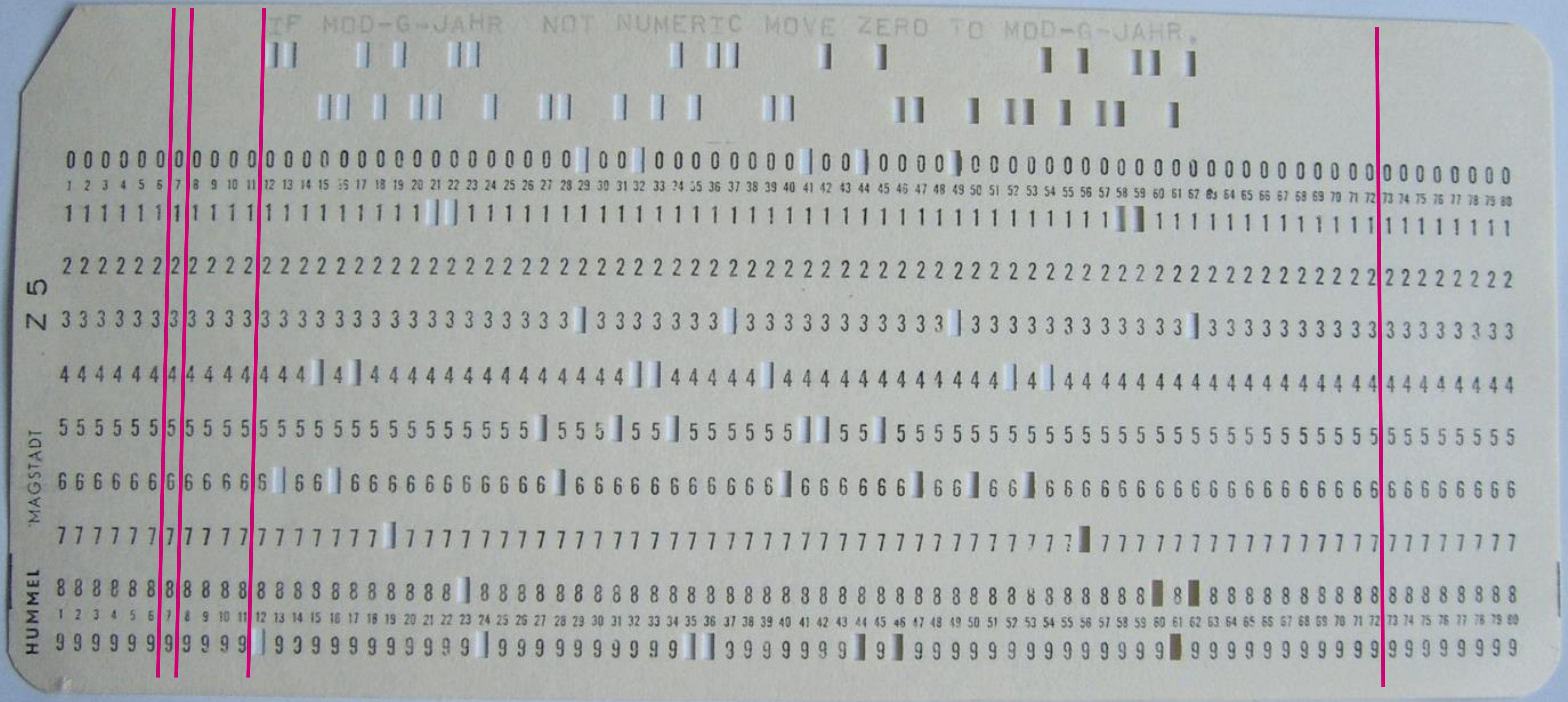
```
i = 1
while i < 6:
  print(i)
  i += 1
else:
  print("error")
```

ALIGN ALL BLOCKS

- Python



Arnold Reinhold, CC-BY-SA, 2006,
https://commons.wikimedia.org/wiki/File:FortranCardPROJ039.agr.jpg



Rainer Gerhards, CC-BY-SA, 2006,
https://commons.wikimedia.org/wiki/File:Punch-card-cobol.jpg

UNIVERSITY OF TWENTE.

Line continuations in HLASM

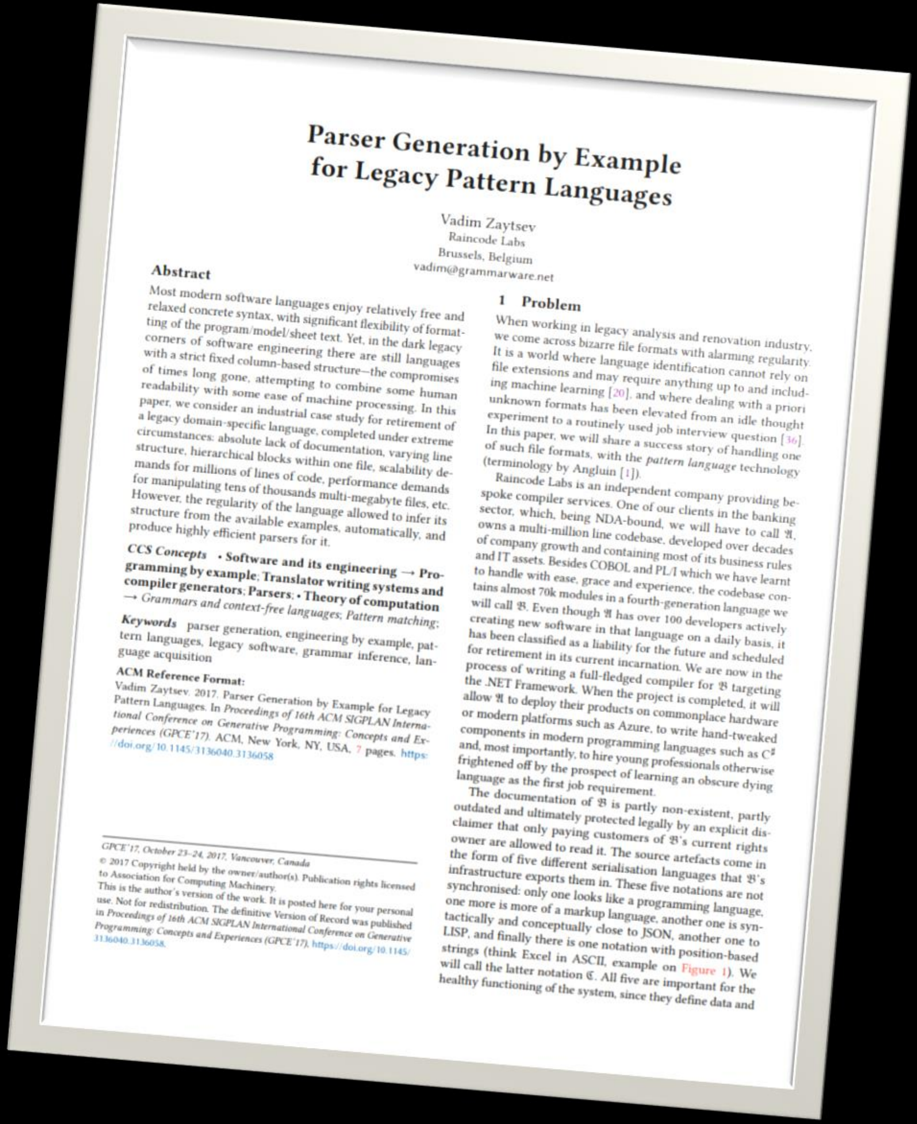
```
 MACRO
         LCLC &KAF
         LCLA &COS
         LCLA &TZT
&N1      MOVE &T,&F
         AIF (T'&T NE T'&F).END
&KAF     SETC 'fHUx'.(3)'X'.'th8LFqnqiexQvPD12RwUA'.'7PVUPZABNmQwOCk r'*
              (9,13)
&TZT     SETA 872+354/469
         AIF (T'&T NE 'F').END
&COS     SETA &TZT
&N2      ST 2,SAVEAREA
         L 2,&F
         LCLB &EOT
         ST 2,&T
         L 2,SAVEAREA
         ANOP
.END     MEND
```

Line continuations in COBOL

```
|...+.*..1....+....2....+....3....+....4....+....5....+....6....+....7..
000001               "AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE
      -              "GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKK
      -    "LLLLLLLLLLMMMMMMMMMM"
000003              N"AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE
      -              "GGGGGGGGGG"
000005     "AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE
      -    "GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKK
      -    "LLLLLLLLLLMMMMMMMMMM"
000010     "AAAAAAAAAABBBBBBBBBBCCCCCCCCCCDDDDDDDDDDEEEEEEEEEE"
      -    "GGGGGGGGGGHHHHHHHHHHIIIIIIIIIIJJJJJJJJJJKKKKKKKKKK"
      -    "LLLLLLLLLLMMMMMMMMMM"
```

TAKEAWAY #2:

- Parsing is “solved”
- Indentation is solved ad hoc
- There is no line continuation calculus
- Example @ GPCE 2017 ⇒



Parser Generation by Example for Legacy Pattern Languages

Vadim Zaytsev
Raincode Labs
Brussels, Belgium
vadim@grammarware.net

Abstract

Most modern software languages enjoy relatively free and relaxed concrete syntax, with significant flexibility of formatting of the program/model/sheet text. Yet, in the dark legacy corners of software engineering there are still languages with a strict fixed column-based structure—the compromises of times long gone, attempting to combine some human readability with some ease of machine processing. In this paper, we consider an industrial case study for retirement of a legacy domain-specific language, completed under extreme circumstances: absolute lack of documentation, varying line structure, hierarchical blocks within one file, scalability demands for millions of lines of code, performance demands for manipulating tens of thousands multi-megabyte files, etc. However, the regularity of the language allowed to infer its structure from the available examples, automatically, and produce highly efficient parsers for it.

CCS Concepts • **Software and its engineering → Programming by example; Translator writing systems and compiler generators; Parsers;** • **Theory of computation** → *Grammars and context-free languages; Pattern matching;*

Keywords parser generation, engineering by example, pattern languages, legacy software, grammar inference, language acquisition

ACM Reference Format:
Vadim Zaytsev. 2017. Parser Generation by Example for Legacy Pattern Languages. In *Proceedings of 16th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE'17).* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3136040.3136058

1 Problem

When working in legacy analysis and renovation industry, we come across bizarre file formats with alarming regularity. It is a world where language identification cannot rely on file extensions and may require anything up to and including machine learning [20], and where dealing with a priori unknown formats has been elevated from an idle thought experiment to a routinely used job interview question [36]. In this paper, we will share a success story of handling one of such file formats, with the *pattern language* technology (terminology by Angluin [1]).

Raincode Labs is an independent company providing bespoke compiler services. One of our clients in the banking sector, which, being NDA-bound, we will have to call 𝔄, owns a multi-million line codebase, developed over decades of company growth and containing most of its business rules and IT assets. Besides COBOL and PL/I which we have learnt to handle with ease, grace and experience, the codebase contains almost 70k modules in a fourth-generation language we will call 𝔅. Even though 𝔄 has over 100 developers actively creating new software in that language on a daily basis, it has been classified as a liability for the future and scheduled for retirement in its current incarnation. We are now in the process of writing a full-fledged compiler for 𝔅 targeting the .NET Framework. When the project is completed, it will allow 𝔄 to deploy their products on commonplace hardware or modern platforms such as Azure, to write hand-tweaked components in modern programming languages such as C♯ and, most importantly, to hire young professionals otherwise frightened off by the prospect of learning an obscure dying language as the first job requirement.

The documentation of 𝔅 is partly non-existent, partly outdated and ultimately protected legally by an explicit disclaimer that only paying customers of 𝔅's current rights owner are allowed to read it. The source artefacts come in the form of five different serialisation languages that 𝔅's infrastructure exports them in. These five notations are not synchronised: only one looks like a programming language, one more is more of a markup language, another one is syntactically and conceptually close to JSON, another one to LISP, and finally there is one notation with position-based strings (think Excel in ASCII, example on Figure 1). We will call the latter notation ℭ. All five are important for the healthy functioning of the system, since they define data and

UNIVERSITY OF TWENTE.

Language features to be afraid of:

INDENTATION

NAMING

Typed names — implicit typing in FORTRAN

```
IMPLICIT REAL (A-Z)
IMPLICIT INTEGER (I-N)
ADJUSTMENT = 1.0E5
NAME = 42
```

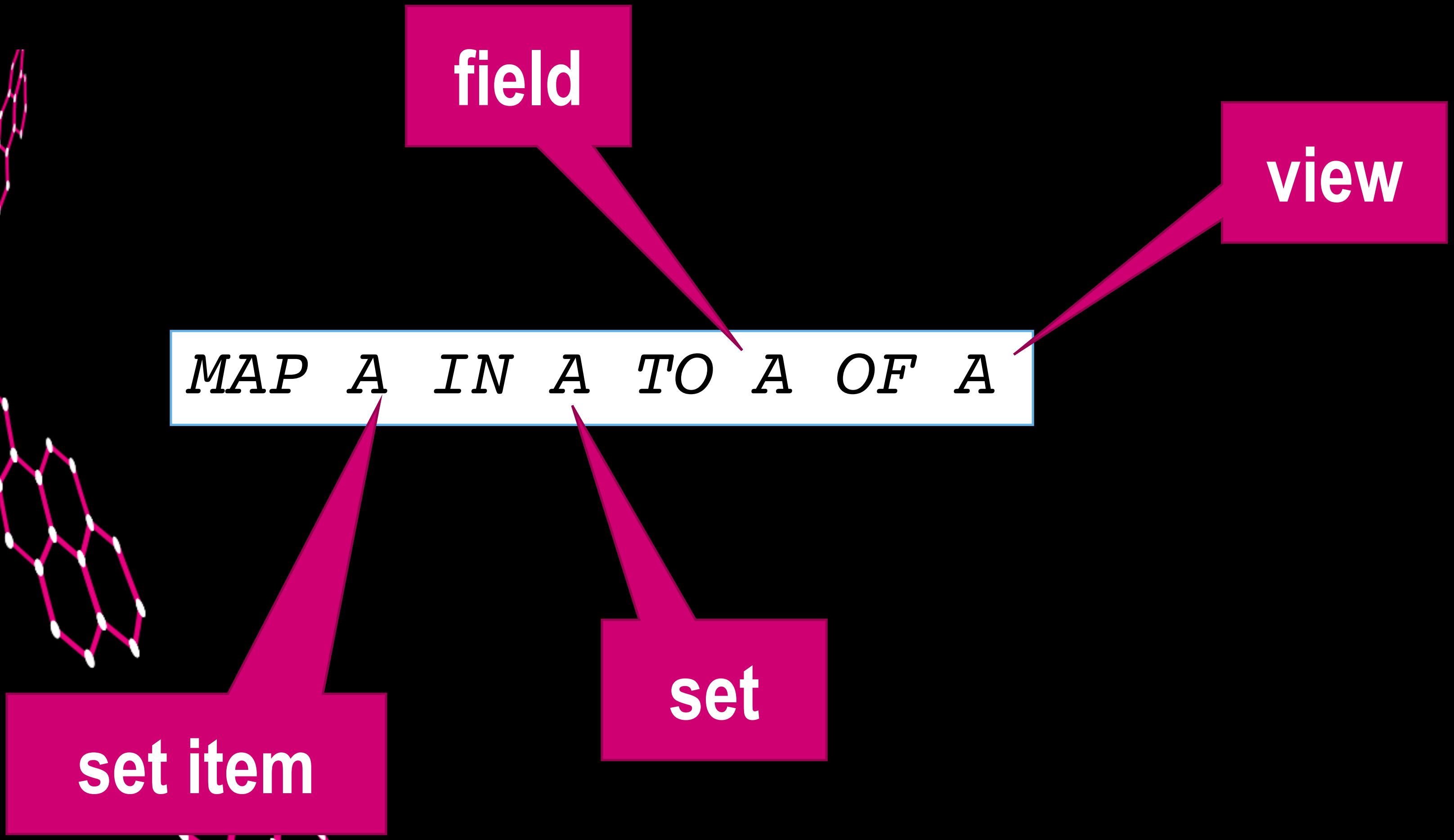
DEFAULT NAME-BASED VALUES & DROP IN REXX

```
hole. = "empty"
hole.9 = "full"
hole.rat = "full"
rat = "cheese"
drop hole.rat
say hole.1 hole.mouse hole.9 hole.rat
```

```
empty empty full HOLE.cheese
```

UNIVERSITY OF TWENTE.

NAMES ARE NOT UNIQUE IN APPBUILDER



```
MAP A IN A TO A OF A
```

KEYWORDS AND NOT RESERVED — PL/I

```
IF THEN = ELSE
THEN ELSE = IF
ELSE IF = THEN
END;
```

Contractions in COBOL

```
Y.X
```

```
MOVE 42 TO X OF Y
```

```
(N.)*Y.(N.)*X
```

```
IF X > 0 AND = Y OR Z THEN DISPLAY X END.
```

```
IF X > 0 AND X = Y OR X = Z THEN DISPLAY X END.
```

Language features to be afraid of:

INDENTATION

NAMING

LEXICAL IMPORTS

Imports in Haskell

```
import Data.Maybe
import qualifying Data.Maybe
import Data.Maybe hiding maybeToList
```

Imports in Python

```
import os.path
import lxml.etree as ET
from library import *
```

Imports in COBOL

```
COPY PRIMLLI
     REPLACING == STD-FSSHH-I == BY == STD-FSSHH-INIT ==.
COPY PRIMLLI
     REPLACING == STD-FSSHH-I == BY == STD-FSSHH-CHKPT ==.
COPY PRIMLLI
     REPLACING == STD-FSSHH-I == BY == STD-FSSHH-END ==.
COPY PRIMLLI
     REPLACING == STD-FSSHH-I == BY == STD-FSSHH-ROLLBACK ==.
COPY PRIMLLI
     REPLACING == STD-FSSHH-I == BY == STD-FSSHH-COMMIT ==.
```

Language features to be afraid of:

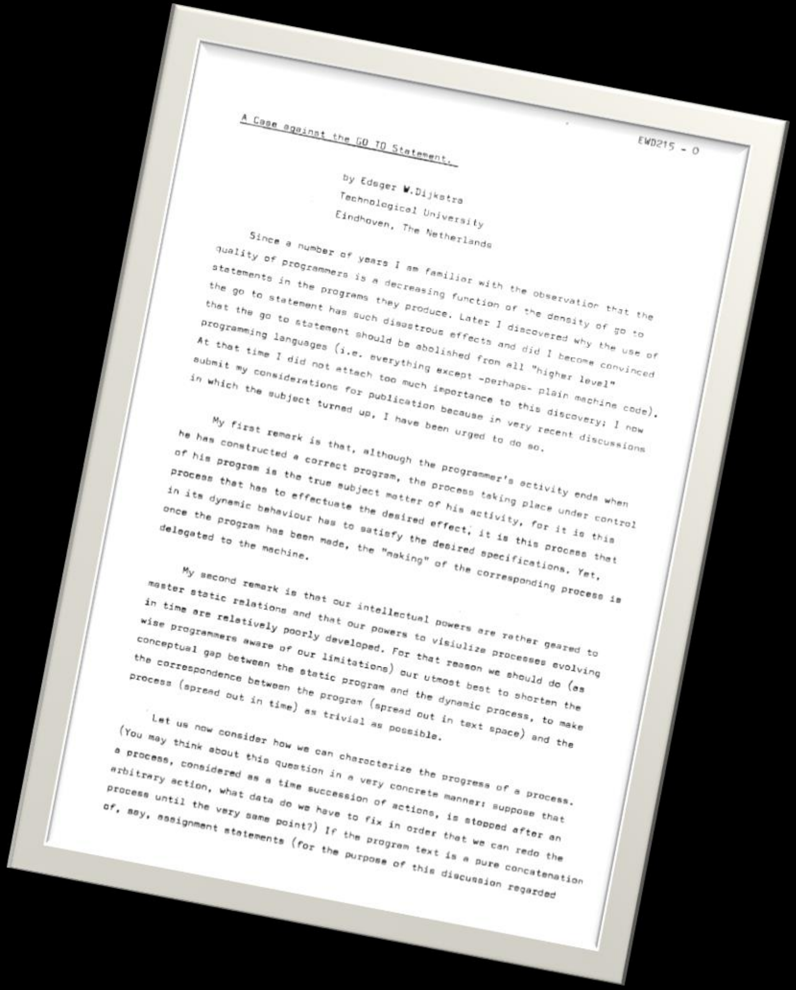
INDENTATION

NAMING

LEXICAL IMPORTS

TRANSFER OF CONTROL

GO TO CONSIDERED HARMFUL? TRY ALTER!



A Case against the GO TO Statement.

EWD215 - 0

by Edsger W.Dijkstra
Technological University
Eindhoven, The Netherlands

Since a number of years I am familiar with the observation that the quality of programmers is a decreasing function of the density of go to statements in the programs they produce. Later I discovered why the use of the go to statement has such disastrous effects and did I become convinced that the go to statement should be abolished from all "higher level" programming languages (i.e. everything except -perhaps- plain machine code). At that time I did not attach too much importance to this discovery; I now submit my considerations for publication because in very recent discussions in which the subject turned up, I have been urged to do so.

My first remark is that, although the programmer's activity ends when he has constructed a correct program, the process taking place under control of his program is the true subject matter of his activity, for it is this process that has to effectuate the desired effect, it is this process that in its dynamic behaviour has to satisfy the desired specifications. Yet, once the program has been made, the "making" of the corresponding process is delegated to the machine.

My second remark is that our intellectual powers are rather geared to master static relations and that our powers to visiulize processes evolving in time are relatively poorly developed. For that reason we should do (as wise programmers aware of our limitations) our utmost best to shorten the conceptual gap between the static program and the dynamic process, to make the correspondence between the program (spread out in text space) and the process (spread out in time) as trivial as possible.

Let us now consider how we can characterize the progress of a process. (You may think about this question in a very concrete manner: suppose that a process, considered as a time succession of actions, is stopped after an arbitrary action, what data do we have to fix in order that we can redo the process until the very same point?) If the program text is a pure concatenation of, say, assignment statements (for the purpose of this discussion regarded

UNIVERSITY OF TWENTE.

GO TO CONSIDERED HARMFUL? TRY ALTER!

```
PROCEDURE DIVISION.


. . .


EXIT-ON-ERROR.
   GO TO EXIT-UPDATE-RECORD.
EXIT-UPDATE-RECORD.
   . . .
EXIT-ROLLBACK-RECORD.
   . . .
```

GO TO CONSIDERED HARMFUL? TRY ALTER!

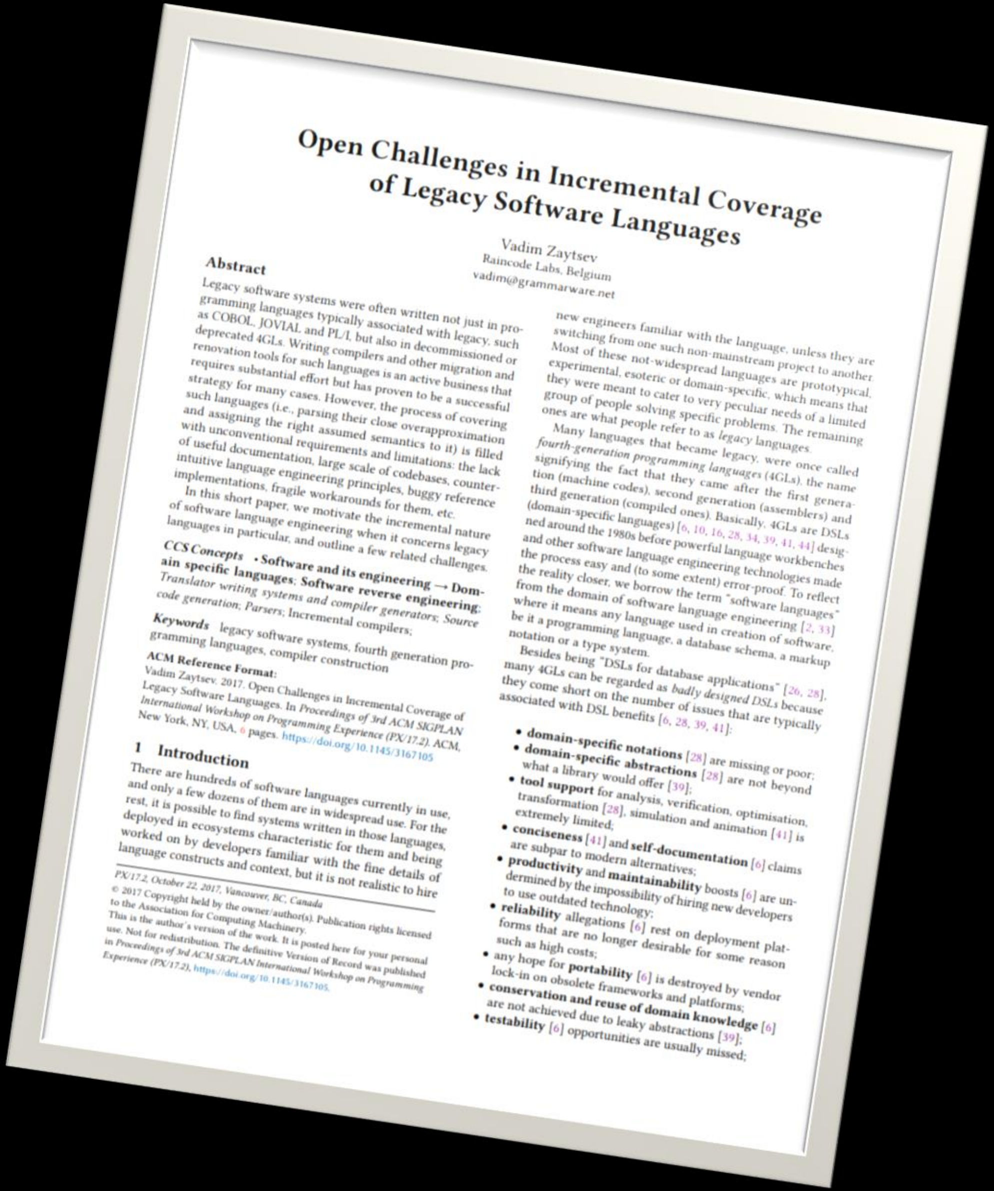
```
PROCEDURE DIVISION.


ALTER EXIT-ON-ERROR TO EXIT-ROLLBACK-RECORD.


EXIT-ON-ERROR.
    GO TO EXIT-ROLLBACK-RECORD.
EXIT-UPDATE-RECORD.
    . . .
EXIT-ROLLBACK-RECORD.
    . . .
```

WANT MORE?

- Semi-structured transfer of control
  - *`GO TO + ALTER`* (COBOL)
  - *`GO TO (…), X`* (FORTRAN)
  - *`EX`* (HLASM)
- Structured transfer of control
  - *`NEXT SENTENCE`* (COBOL)
  - *`PERFORM THRU`* (COBOL)
  - *`DO`* (AppBuilder)
  - *`SIGNAL`* (REXX)

TAKEAWAY #3:

- Read the documentation carefully!
- Expect the unexpected
  - and still be surprised
- More new challenges = good!
- Example @ PX/17.2 ⇒



Open Challenges in Incremental Coverage of Legacy Software Languages

Vadim Zaytsev
Raincode Labs, Belgium
vadim@grammarware.net

Abstract

Legacy software systems were often written not just in programming languages typically associated with legacy, such as COBOL, JOVIAL and PL/I, but also in decommissioned or deprecated 4GLs. Writing compilers and other migration and renovation tools for such languages is an active business that requires substantial effort but has proven to be a successful strategy for many cases. However, the process of covering such languages (i.e., parsing their close overapproximation and assigning the right assumed semantics to it) is filled with unconventional requirements and limitations: the lack of useful documentation, large scale of codebases, counter-intuitive language engineering principles, buggy reference implementations, fragile workarounds for them, etc.

In this short paper, we motivate the incremental nature of software language engineering when it concerns legacy languages in particular, and outline a few related challenges.

CCS Concepts • **Software and its engineering → Domain specific languages**; **Software reverse engineering**; *Translator writing systems and compiler generators; Source code generation; Parsers;* Incremental compilers;

Keywords legacy software systems, fourth generation programming languages, compiler construction

ACM Reference Format:
Vadim Zaytsev. 2017. Open Challenges in Incremental Coverage of Legacy Software Languages. In *Proceedings of 3rd ACM SIGPLAN International Workshop on Programming Experience (PX/17.2).* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3167105

1 Introduction

There are hundreds of software languages currently in use, and only a few dozens of them are in widespread use. For the rest, it is possible to find systems written in those languages, deployed in ecosystems characteristic for them and being worked on by developers familiar with the fine details of language constructs and context, but it is not realistic to hire new engineers familiar with the language, unless they are switching from one such non-mainstream project to another. Most of these not-widespread languages are prototypical, experimental, esoteric or domain-specific, which means that they were meant to cater to very peculiar needs of a limited group of people solving specific problems. The remaining ones are what people refer to as *legacy* languages.

Many languages that became legacy, were once called *fourth-generation programming languages* (4GLs), the name signifying the fact that they came after the first generation (machine codes), second generation (assemblers) and third generation (compiled ones). Basically, 4GLs are DSLs (domain-specific languages) [6, 10, 16, 28, 34, 39, 41, 44] designed around the 1980s before powerful language workbenches and other software language engineering technologies made the process easy and (to some extent) error-proof. To reflect the reality closer, we borrow the term "software languages" from the domain of software language engineering [2, 33] where it means any language used in creation of software, be it a programming language, a database schema, a markup notation or a type system.

Besides being "DSLs for database applications" [26, 28], many 4GLs can be regarded as *badly designed DSLs* because they come short on the number of issues that are typically associated with DSL benefits [6, 28, 39, 41]:

- **domain-specific notations** [28] are missing or poor;
- **domain-specific abstractions** [28] are not beyond what a library would offer [39];
- **tool support** for analysis, verification, optimisation, transformation [28], simulation and animation [41] is extremely limited;
- **conciseness** [41] and **self-documentation** [6] claims are subpar to modern alternatives;
- **productivity** and **maintainability** boosts [6] are undermined by the impossibility of hiring new developers to use outdated technology;
- **reliability** allegations [6] rest on deployment platforms that are no longer desirable for some reason such as high costs;
- any hope for **portability** [6] is destroyed by vendor lock-in on obsolete frameworks and platforms;
- **conservation and reuse of domain knowledge** [6] are not achieved due to leaky abstractions [39];
- **testability** [6] opportunities are usually missed;

UNIVERSITY OF TWENTE.

Conclusion

- Your life is run by COBOL
  - banking, booking, ordering, …
- BabyCobol is a WIP
  - http://slebok.github.io/baby
- Mainframe languages are great
  - hide many challenges
- Follow @grammarware!