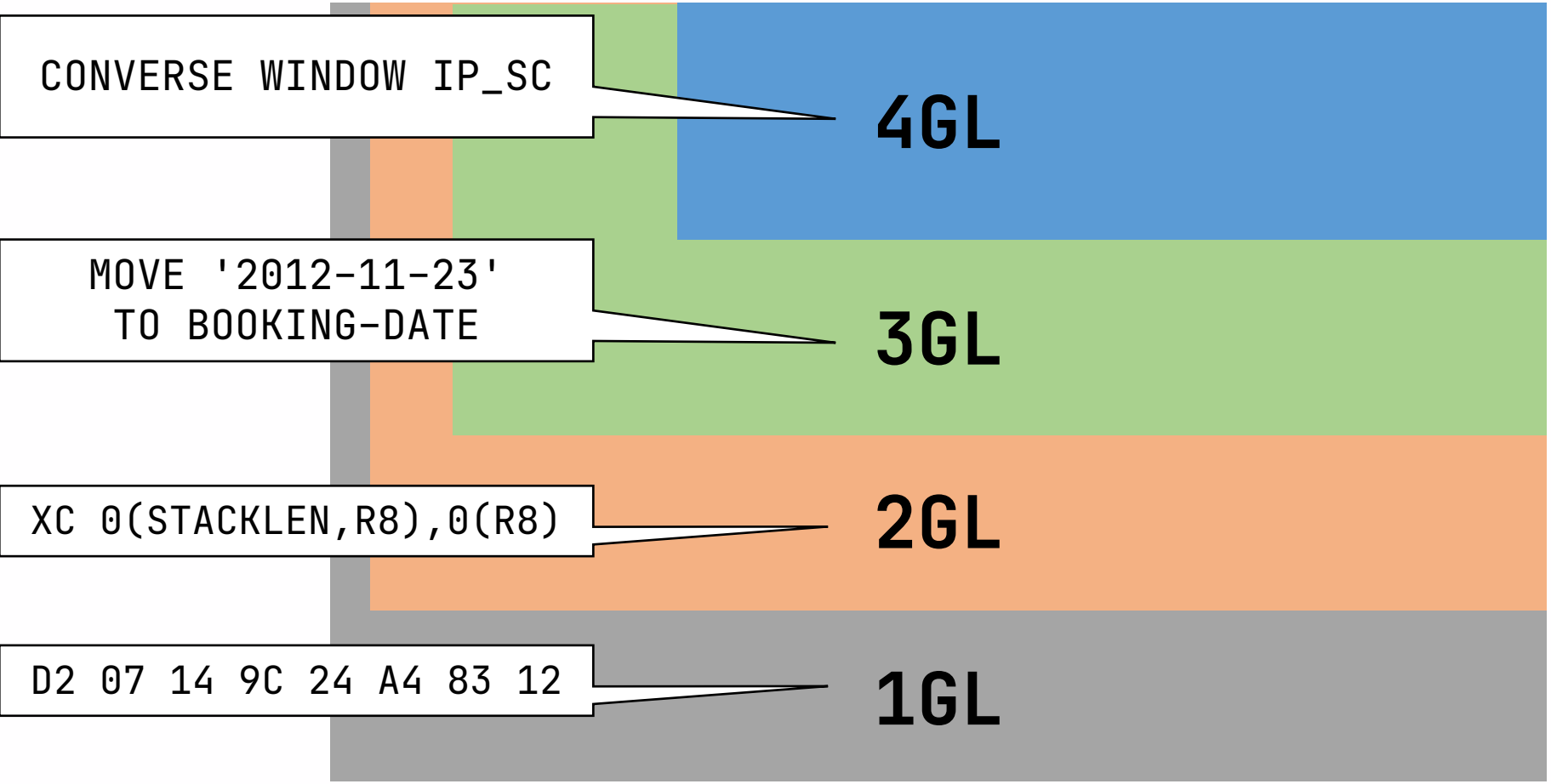
LEGACY LANGUAGES

Software Evolution — L4T1

Dr. Vadim Zaytsev aka @grammarware, February/March 2021

UNIVERSITY OF TWENTE.
Formal Methods & Tools

A Brief History of Languages



Brief 4GL History

- 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

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

Brief 4GL History

- 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 somewhat difficult to learn.

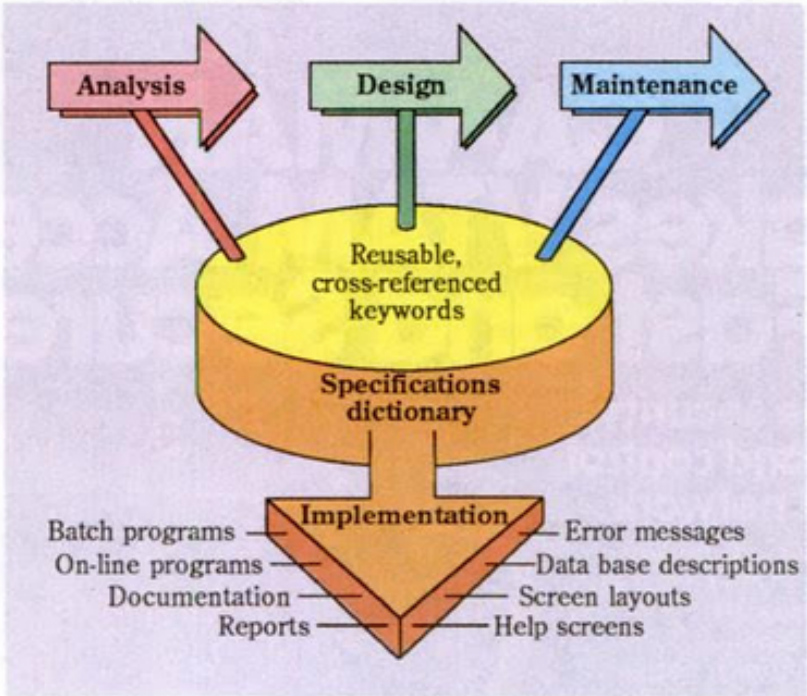
Marketed in the U.S. by CGI Systems, Inc., located here, the system is based on the Merise methodology, a structured programming technique popular in France. The system runs on IBM mainframes under DOS, DOS/VSE and MVS as well as on Honeywell, Inc. and Unisys Corp. large-scale systems. It supports most teleprocessing monitors and data base management systems, including IBM's DB2.

Pacbase is installed at 500 sites worldwide, including 90 in the U.S., the company notes. 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 was $77 million last year, CGI says.

Pacbase has evolved from an

Continued on page 23

Pacbase
Development methodology

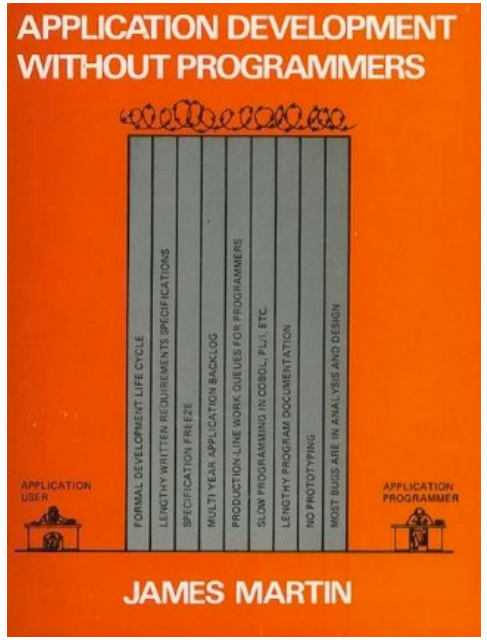


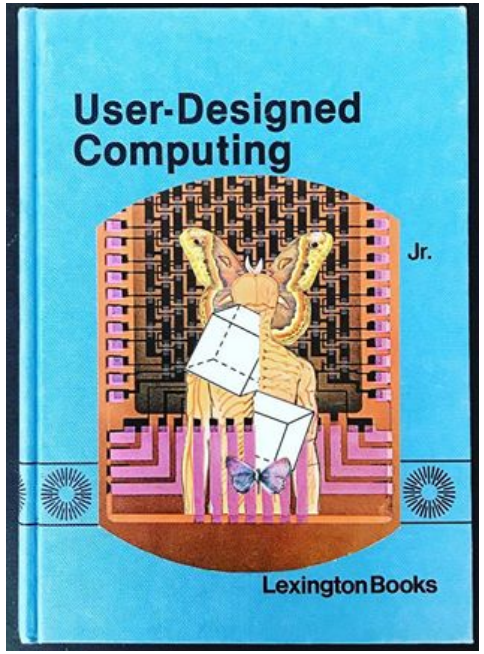
INFORMATION PROVIDED BY CGI SYSTEMS, INC.
CW CHART

Computerworld – Aug 17, 1987

Brief 4GL History

- 1980s:
  - non-procedural
  - high-level
  - specification languages
- Evolution
  - 12 pages of COBOL
  - 2 pages of Mark IV
  - 1 statement in Nomad
- Problem?





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

In 2021…

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 somewhat difficult to learn.

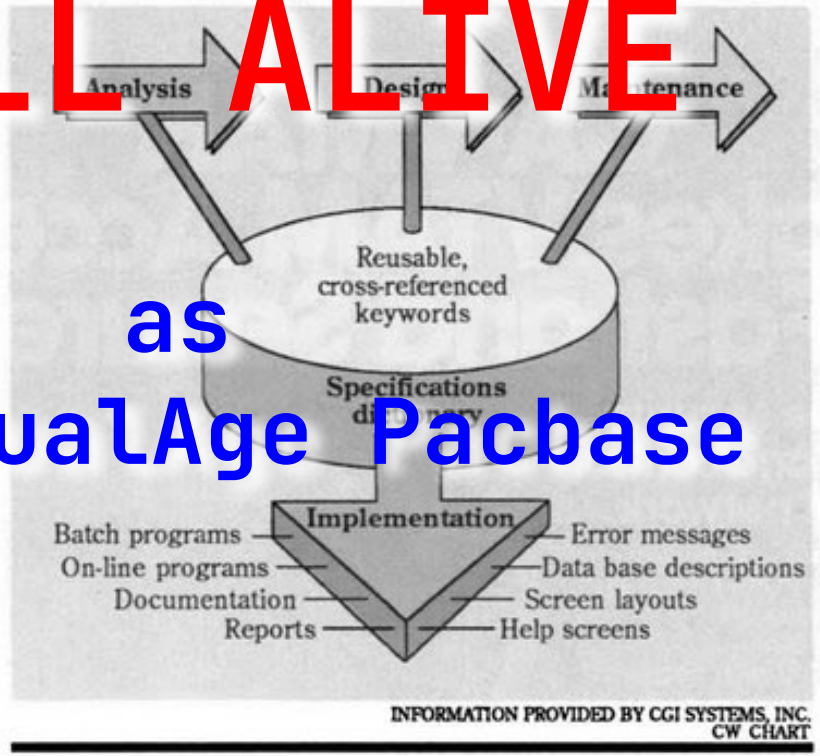
Marketed in the U.S. by CGI Systems, Inc., located here, the system is based on the Merise methodology, a structured programming technique popular in France. The system runs on IBM mainframes under DOS, DOS/VSE and MVS as well as on Honeywell, Inc. and Unisys Corp. computers. It supports most teleprocessing monitors and data base management systems, including IBM's DB2.

Pacbase is installed at 500 sites worldwide, including 90 in the U.S., the company notes. 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 was $77 million last year, CGI says.

Pacbase has evolved from an

Continued on page 23

Pacbase
Development methodology



INFORMATION PROVIDED BY CGI SYSTEMS, INC.
CW CHART

STILL ALIVE as IBM VisualAge Pacbase

MARK IV

FILE MANAGEMENT SYSTEM

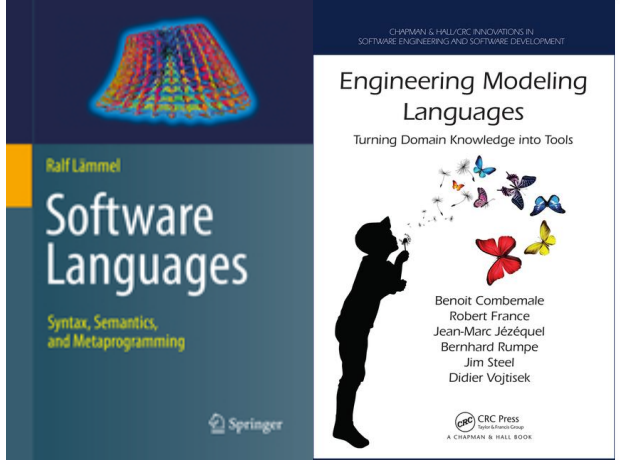
The general purpose software product line for business data processing

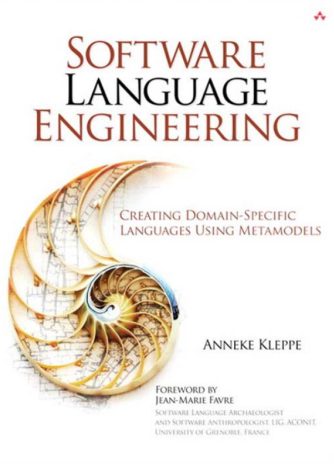
STILL ALIVE as CA VISION:BUILDER

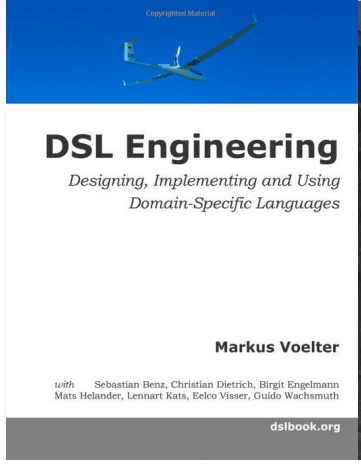
informatics inc

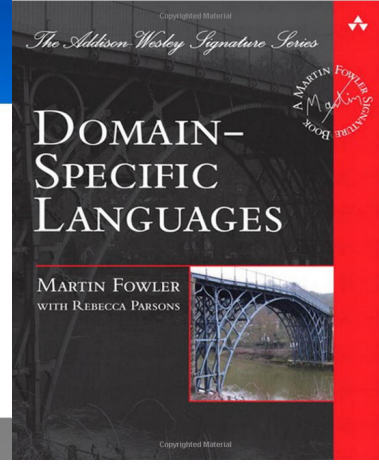
UNIVERSITY OF TWENTE.

DSL = ?









- Wikipedia
  - software language specialised to an application domain
    - in contrast to a GPL
- SLEBoK
  - abstractions targeted at a problem domain
    - in contrast to a library
- *(many variations)*
- 4GL:
  - higher level of abstraction
  - programmer-friendly, powerful, versatile

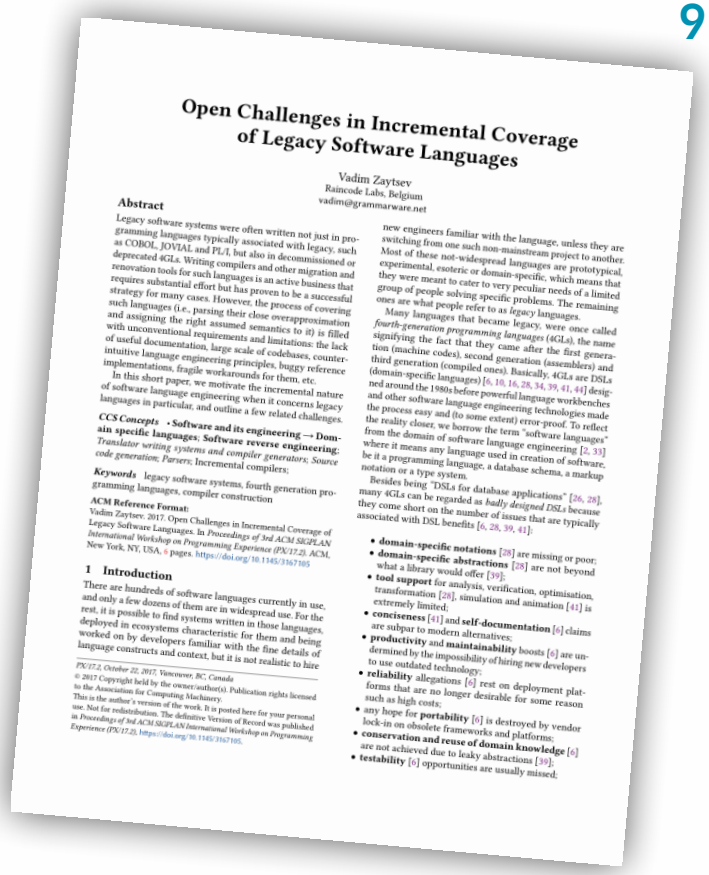
DSL Example: Rascal

```
module Idiomatic

import lang::java::\syntax::Java15;
import IO;
import ParseTree;

CompilationUnit idiomatic(CompilationUnit unit) = innermost visit(unit) {
   case (Stm) `if (!<Expr cond>) <Stm a> else <Stm b>` =>
        (Stm) `if (<Expr cond>)  <Stm b> else <Stm a>`

   case (Stm) `if (<Expr cond>) <Stm a>` =>
        (Stm) `if (<Expr cond>) { <Stm a> }`
     when (Stm) `<Block _>` !:= a

   case (Stm) `if (<Expr cond>) <Stm a> else <Stm b>` =>
        (Stm) `if (<Expr cond>) { <Stm a> } else { <Stm b> }`
     when (Stm) `<Block _>` !:= a

   case (Stm) `if (<Expr cond>) { return true; } else { return false; }` =>
        (Stm) `return <Expr cond>;`
};
```

https://www.rascal-mpl.org/#_Transformations

Advantages of a DSL

- domain-specific notations
- domain-specific abstractions, reuse
- tool support
- conciseness, self-documentation
- productivity, maintainability
- reliability
- portability
- testability
- lifespan
- configurability per customer



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;

PX/17.2, October 22, 2017, Vancouver, BC, Canada

Conclusion

- Banks, governments, airlines, etc own **legacy** code
  - in production
- Legacy 2GLs (HLASM), 3GLs (COBOL, PL/I, JOVIAL, ADA…), 4GLs (Pacbase, Coolgen, AppBuilder), CICS, SQL, JCL…
- Retirement, costs, death of hardware, databases, devs
- **DSL**s vs **4GL**s
- Q&A Sessions @ Canvas
  - ⇒ v.zaytsev@utwente.nl
  - ⇒ https://discord.gg/n7VQAPNBPD