Automated Porting of Software Systems using DMS

Ira D. Baxter, Ph.D.

Chief Technical Officer

The Software Porting Problem

- Problem: *Critical* application on legacy platform
  - Old technology hurts the application's organization!
    - Old/Proprietary language: hard to find tools, programmers
    - Old platform: hard to get support, low performance
    - Old foundations: hierarchical DB, green screen, custom OS
    - Customer perception of archaic product reduces sales
  - Poor Documentation of System and Application
    - Only have source code, dusty design documents of dubious accuracy
    - Danger of losing hard-won business rules woven into code
  - How to get application on modern platform?
    - Hand translation too slow, too expensive, unreliable
    - Can't find COTS software with equivalent functionality
    - Must account for target platform quirks (OS, screens, 3rd party support)
    - May need to move databases as well as code!
- *Solution: Translate using generalized compiler*
  - DMS Reengineering Toolkit + custom transformations

Typical Porting Scenarios

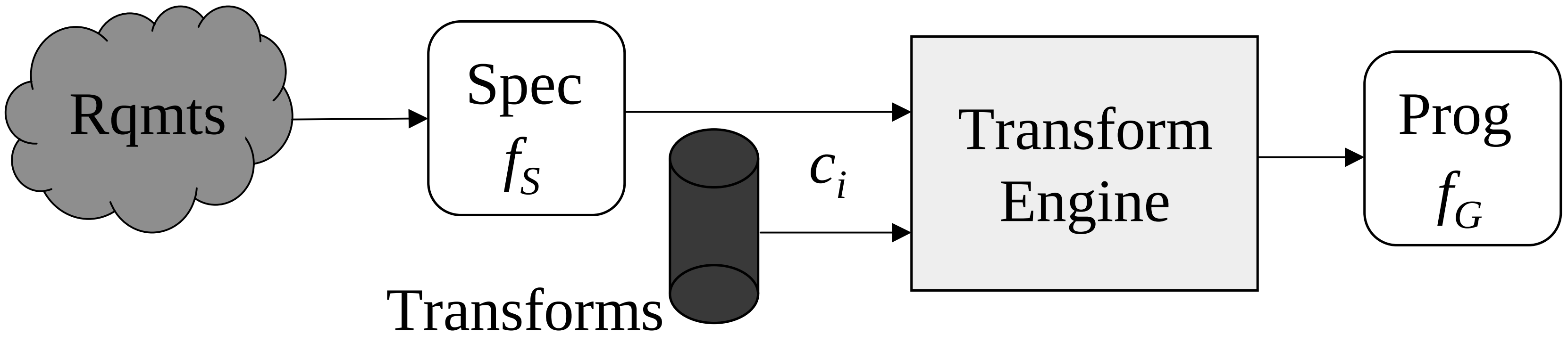
- ***JOVIAL73 on MIL1750 → C on PowerPC***
  - Military Avionics + Weapons management
- COBOL74 + IDMS → COBOL85 + SQL
  - UNISYS 1100 retirement; must move data, too!
- K&R C + custom RTOS → ANSI C + VXworks
  - Microprocessor modernization
- Clipper + green screen → Delphi + GUI
  - Legacy 3GL data processing language
- ColdFusion: 4GL + HTML + Jscript + SQL
  → Jscript + Java + JSP+ SQL + J2EE
  - Early web application modernization
- HP3000 → ?
  - HP drops support for popular hardware/OS

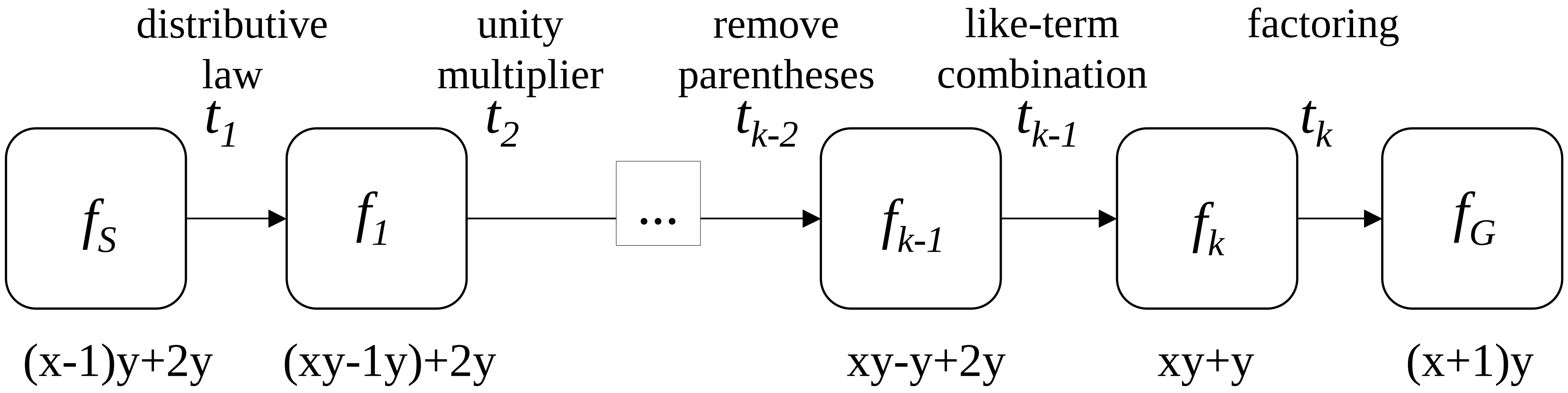
Why porting by hand often *fails*

- Software Engineers are slow and miss details
  - 100 SLOC/day → Too expensive!
    - 50K SLOC takes 2 man-years; 1M SLOC with team of 20: 2.5 years
  - Poor documentation, hard-to-read code → SEs guess at function
    - Bad guesses, miss details in the code
  - Essentially new code → buggy implementation
- Application system changes during process
  - Organization must update function to survive
  - Porting SEs deluged with changes,
    or completed result not up-to-date with legacy application
- Scope creep
  - Temptation to "improve" rather than concentrate on port
  - No good way to test new features
- User training shock on switchover
  - System works differently, missing functions, buggy

Transformation Systems

Stepwise Semiautomatic Conversion of Specs to Code





DMS Software Reengineering Toolkit

- Customized, *automated* analysis, modification or generation of software
  - For sources for large scale software systems
    - Scalable to thousands of files, millions of source lines
  - Handles many *and mixed* languages simultaneously
    - Easily accepts language definitions of arbitrary languages
    - Already defined: C, C++, Java, COBOL, Fortran, SQL, XML, VB …
  - Generalized compiler technology
    - Parsing, analyzing, transforming, Prettyprinting
- Semantic Designs Supporting Services
  - Implementation of DMS customization
  - Consulting & Training on DMS usage

Reengineering Tasks done with DMS

- Prettyprinting sources (many languages)
- Full system hyperlinked cross-reference (COBOL, Java)
- VBScript, Java, C# Metrics
- Restructuring of HTML across website (frame insertion, etc.)
- Automated removal of dead preprocessor conditionals
- Automated detection/removal of duplicate code (C, COBOL, Java)
- Automated detection/removal of dead code (Java)
- Test Coverage/Profiling tools (C, COBOL, Java)
- Factory controller code generation from factory process specs
- Lightning fast XML parser generation from DTDs
- *JOVIAL to C translation*

Automated!
Repeatable!
Scalable!

Porting by DMS

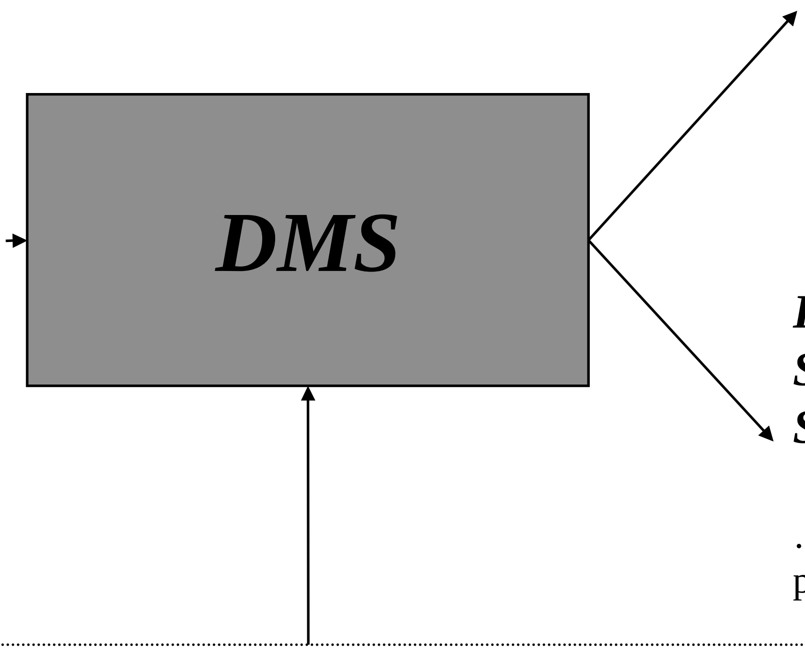
Analyses of Software System

a in b
b in c
a in c

Software System Sources (MSLOC)

example

… a in b…
if a in c then p:=7 else p:=2 endif



Ported Software System Sources

… a in b…
p=7;

Captured!

Background Knowledge for Software Analysis or Modification problem (KSLOC)

•**Language Definitions**	Ada, SQL, Java …
•**General Language Analyses**	”*var:=expression*” => “*var* modified”
•**General Language Transforms**	”if false then *s* else *t* endif -> *t*”
•***Port-specific Language Analyses***	“*x* in *y* and *y* in *z*” => “*x* in *z*”
•***Port-specific Language Transforms***	“BASIC: *l*: if *c* then *s* \goto *l* -> C++: while (*c*) do *s*;”

from Semantic Designs or *Consultants/Sr. Programmers/Engineers*

DMS Knowledge Organized as *Domains*

- Notation
  - External Form (what you can say: string or graphical)
  - Internal Form (How DMS stores it)
  - *Parser* (how to convert external form to internal form)
  - *PrettyPrinter* (how to display the Internal Form)
- Semantics (what the Internal Form *means*)
  - *Optimizations* (how to optimize in the domain)
  - Refinements (how to transform IF to another IF)
  - *Analyzers* (how to analyze in the domain)
  - Attachments (procedures to enhance DMS efficiency)

DMS Domain for Java
Parser + Pretty Printer

```
nested_class_declaration = nested_class_modifiers class_header class_body ;
  <<PrettyPrinter>>:  { V(H(nested_class_modifiers,class_header),class_body); }

class_header = 'class' IDENTIFIER ;
  <<PrettyPrinter>>:  { H('class',IDENTIFIER); }
class_header = 'class' IDENTIFIER 'implements' name_list ;
  <<PrettyPrinter>>:  { H('class',IDENTIFIER,'implements',name_list); }
class_header = 'class' IDENTIFIER 'extends' name;
  <<PrettyPrinter>>:  { H('class',IDENTIFIER,'extends',name); }
class_header = 'class' IDENTIFIER 'extends' name 'implements' name_list ;
  <<PrettyPrinter>>:  { H('class',IDENTIFIER,'extends',name,'implements',name_list);

class_body = '{' class_body_declarations '}' ;
  <<PrettyPrinter>>:  { V(H('{',STRING(" "),class_body_declarations),'}'); }

nested_class_modifiers = nested_class_modifiers nested_class_modifier ;
  <<PrettyPrinter>>:  { H(CH(nested_class_modifiers[1]),nested_class_modifier); }
```

… + 300 more rules…(COBOL is 3500!)

Optimization transform for DMS Rewrite Rule Language

```
default base domain Java;

rule merge-ifs(\condition1,
               \condition2,
               \then-statements)
   "if (\condition1)
       if (\condition2)
          { \then-statements
          }
   "
   rewrites to
   "if (\condition1 && \condition2)
       { \then-statements }   ";
```

Domain Name

Domain Syntax

Expertise == Number of Rules

- Mathematics
  - Novice (9th grade algebra): $x+0 \Rightarrow x$
  - Amateur (HS Senior): $\sin^2(x)+\cos^2(x) \Rightarrow 1$
  - Journeyman: (Frosh Calculus) integrals
  - Craftsman: (B.S. Math) Linear Algebra, Group Theory
  - Expert: (Ph.D. Mathematics) Category Theory, Topology, …
- DMS
  - Toy: several rules
  - Useful: 50 rules (simplification/optimization)
  - Powerful: 250 rules (testing, code generation)
  - Indispensable: 2000 rules (massive program translation)

Software Test Coverage

- Analysis of code executed by test cases
  - *Non*-executed code likely to have flaws
  - How can we identify such code?
- Key problem: tracking program control flow
  - Need way to identify possible program parts
  - Capture “executed” status of parts via tests
  - Display execution status of program parts

COBOL Test Coverage - Example

```
MAINLINE.

  OPEN INPUT OldFile.
  OPEN OUTPUT NewFile.
  MOVE SPACES TO FileStatusIndicator.
  PERFORM UNTIL EndOfFile

    READ OldFile
      AT END

        MOVE "Y" TO FileStatusIndicator
      NOT AT END

        PERFORM WriteNewRecord
    END-READ
  END-PERFORM.

  STOP RUN.
WriteNewRecord.

  MOVE CORRESPONDING OldRecord TO NewRecord.
  MOVE CORRESPONDING OldDateFormat OF OldRecord
                TO NewDateFormat OF NewRecord.
  MOVE 19 TO CC OF NewDateFormat OF NewRecord.
  MOVE YY IN OldFile TO YY IN NewFile.
  WRITE NewRecord.
```

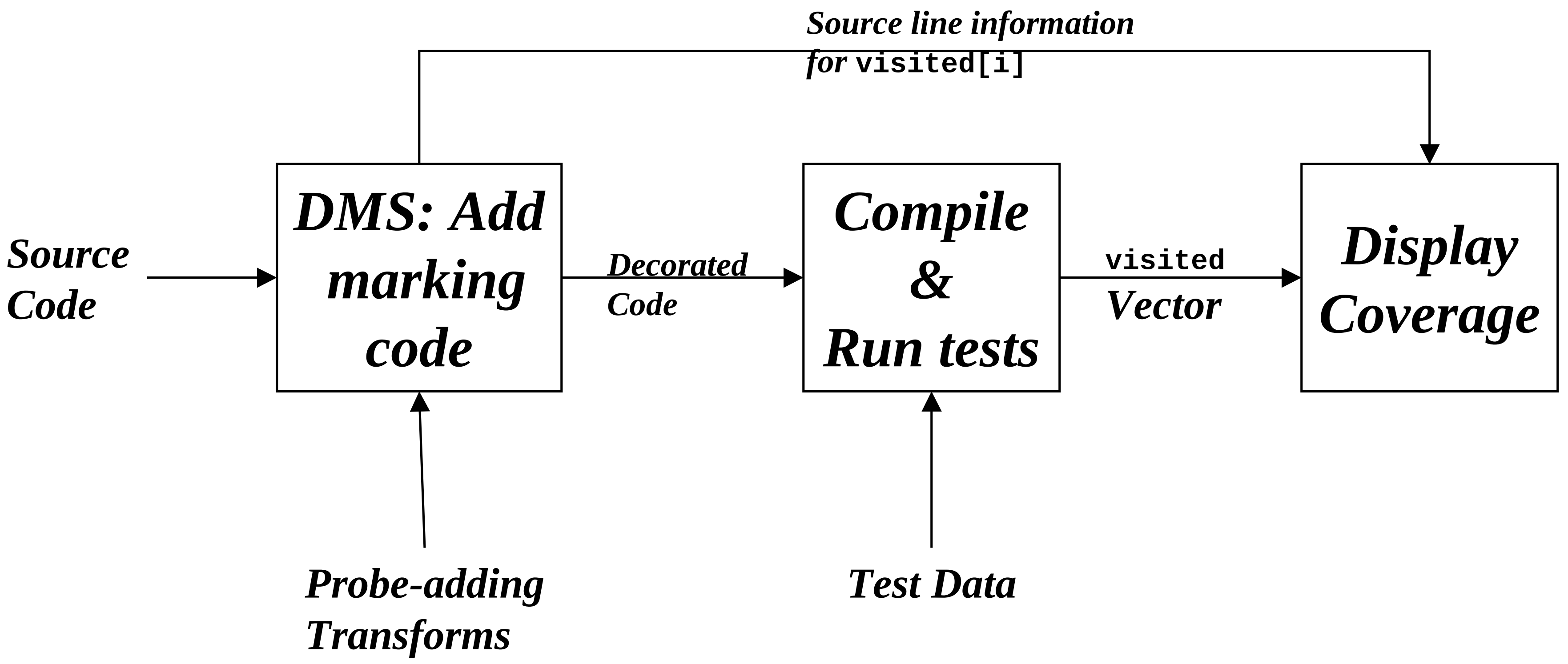
Y2K data conversion program

```
TestCoverage-Start-Main SECTION.
  MAINLINE.
    SET TestCoverage-Covered (5) TO TRUE.
    OPEN INPUT OldFile.
    OPEN OUTPUT NewFile.
    MOVE SPACES TO FileStatusIndicator.
    PERFORM UNTIL EndOfFile
      SET TestCoverage-Covered (4) TO TRUE
      READ OldFile
        AT END
          SET TestCoverage-Covered (3) TO TRUE
          MOVE "Y" TO FileStatusIndicator
        NOT AT END
          SET TestCoverage-Covered (2) TO TRUE
          PERFORM WriteNewRecord
      END-READ
    END-PERFORM.
    PERFORM TestCoverage-Dump IN TestCoverage-Code
    STOP RUN.
  WriteNewRecord.
    SET TestCoverage-Covered (1) TO TRUE.
    MOVE CORRESPONDING OldRecord TO NewRecord.
    MOVE CORRESPONDING OldDateFormat OF OldRecord
                TO NewDateFormat OF NewRecord.
    MOVE 19 TO CC OF NewDateFormat OF NewRecord.
    MOVE YY IN OldFile TO YY IN NewFile.
    WRITE NewRecord.
TestCoverage-FallThru-Finalize SECTION.
  TestCoverage-Finalize.
    PERFORM TestCoverage-Dump IN TestCoverage-Code.
```

COBOL Test Coverage - Rules

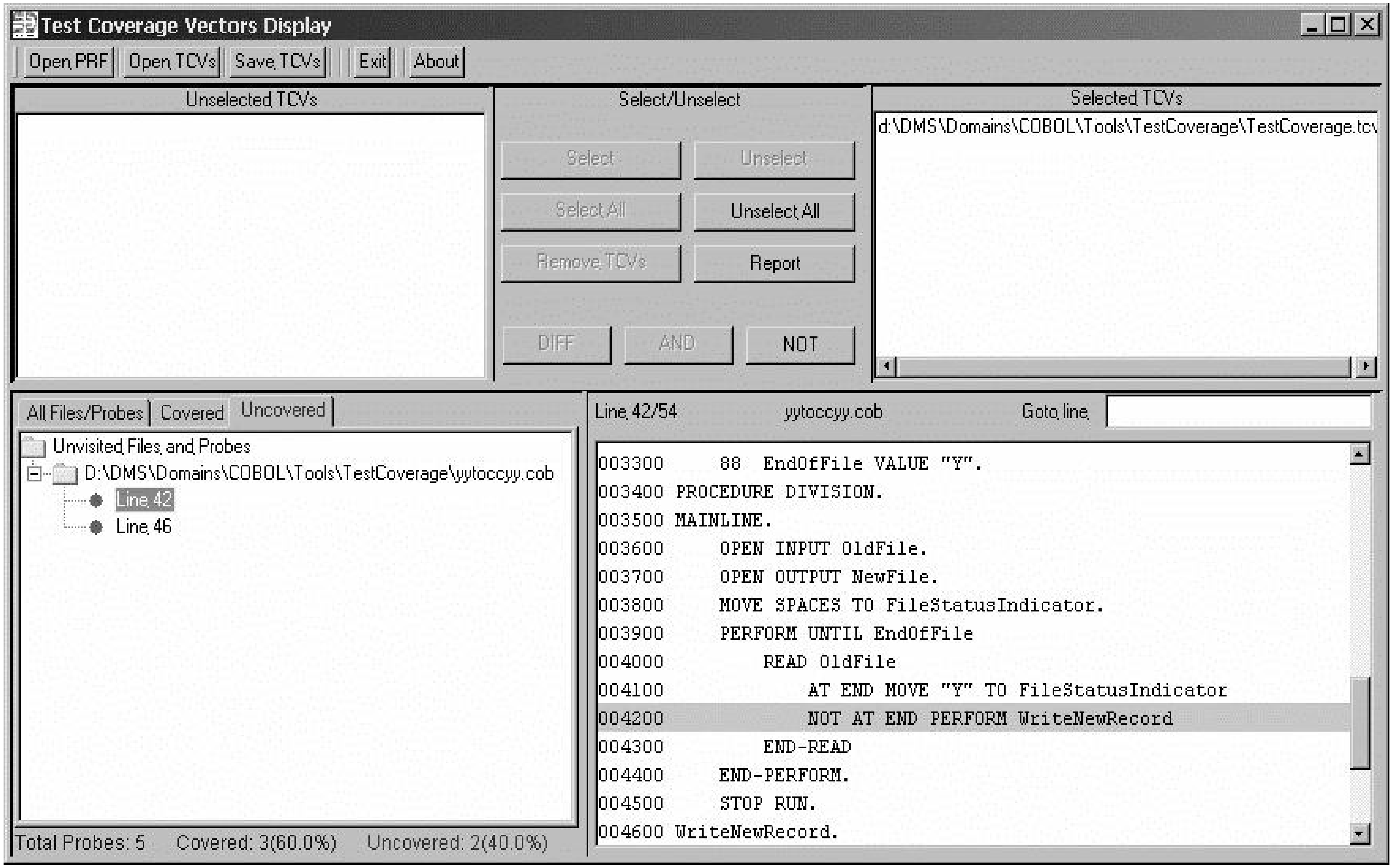
```
external pattern create_test_coverage_index (for_construct:*@*):unsigned_integer_number
    = 'TestCoverage/CreateTestCoverageIndex'.


private pattern probe(for_construct:*@*):imperative_statement
    = "SET TestCoverage-Covered (\create_test_coverage_index\(\for_construct\)) TO TRUE".


private rule install_in_at_end_phrase_single(i:imperative_statement)
                                                                :at_end_phrase->at_end_phrase
    = "AT END \i"
   -> "AT END \probe\(\i\) \i".


private rule install_in_perform_statement_single(pp:perform_phrase,i:imperative_statement)
                                                        :perform_statement->perform_statement
    = "PERFORM \pp \i END-PERFORM"
   -> "PERFORM \pp \probe\(\i\) \i END-PERFORM".


private condition is_probe_sentence(s:sentence)
    = [i:unsigned_integer_number. s <: "\:sentence \probe\(\i\)."].


private rule install_in_paragraph_multiple(n:defining_paragraph_or_section_name,s:sentence,
                                                       sl:sentence_list):paragraph->paragraph
    = "\n. \s \sl\:sentence"
   -> "\n. \probe\(\n. \s \sl\:sentence\). \s \sl\:sentence"
   if ~is_probe_sentence(s).
```

Test Coverage Tool Flow



Note: incrementing **`visited`** *rather then setting true changes this to profiler tool!*

COBOL Test Coverage Tool Display



d:\DMS\Domains\COBOL\Tools\TestCoverage\TestCoverage.tcv

Unvisited Files and Probes
D:\DMS\Domains\COBOL\Tools\TestCoverage\yytoccyy.cob
Line 42
Line 46

Total Probes: 5 Covered: 3(60.0%) Uncovered: 2(40.0%)

Line 42/54 yytoccyy.cob

```
003300     88  EndOfFile VALUE "Y".
003400 PROCEDURE DIVISION.
003500 MAINLINE.
003600     OPEN INPUT OldFile.
003700     OPEN OUTPUT NewFile.
003800     MOVE SPACES TO FileStatusIndicator.
003900     PERFORM UNTIL EndOfFile
004000         READ OldFile
004100             AT END MOVE "Y" TO FileStatusIndicator
004200             NOT AT END PERFORM WriteNewRecord
004300         END-READ
004400     END-PERFORM.
004500     STOP RUN.
004600 WriteNewRecord.
```

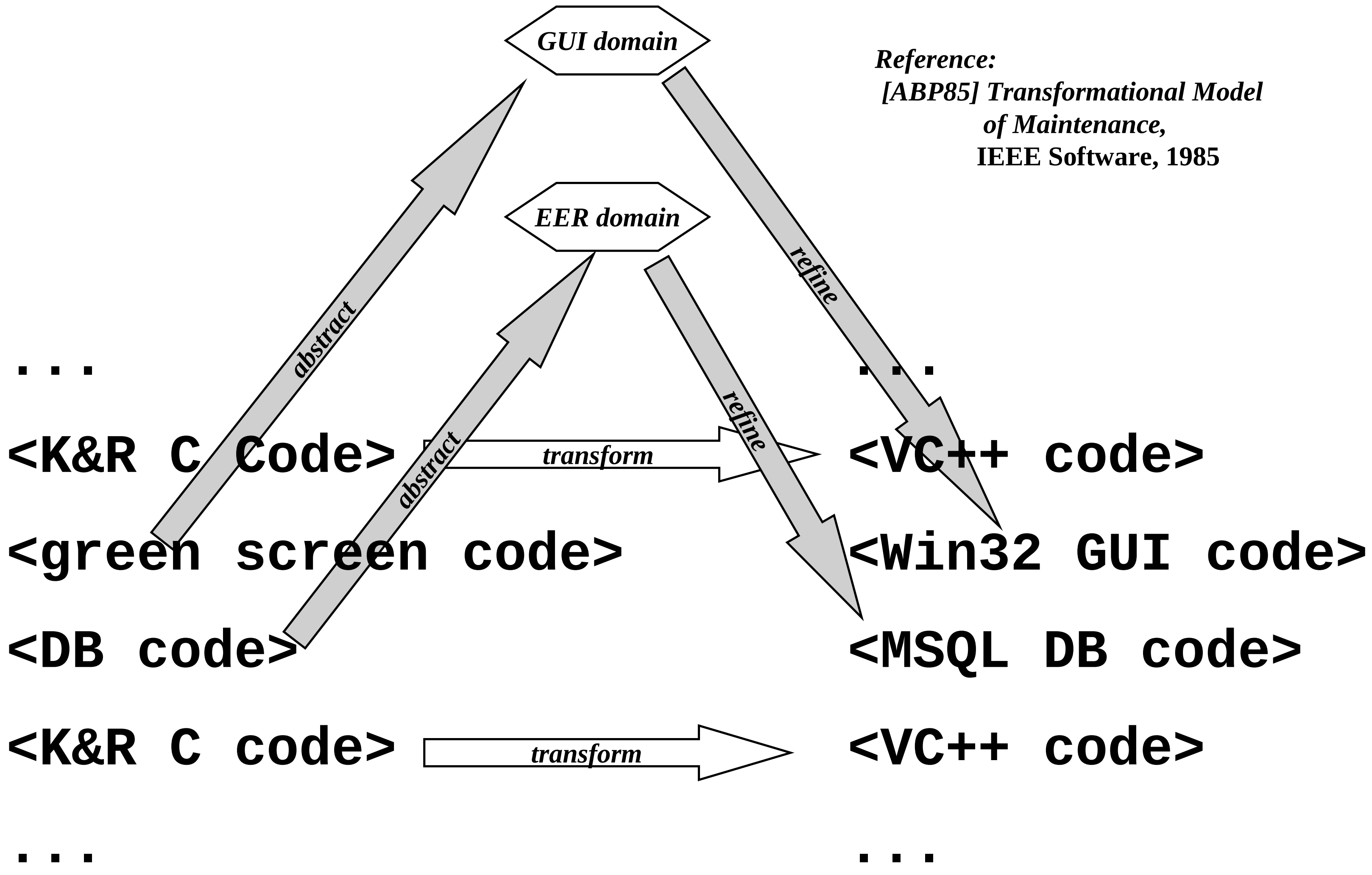
Typical porting issues

- How many source programming languages, REALLY?
  - Which ones to translate automatically? …manually? …ignore?
- How many target programming languages?
  - Definition of programming languages?
    - Trustworthy, detailed documentation for AlphaBasic?
    - Expert interpretation of IDMS?
- How to translate language and environmental idioms?
  - Pointer arithmetic and data aliases
  - Operating system and library calls (open_file, string_copy, WAIT)
- Translation technology: what? from where? scalability?
- Moving application data
- Testing the ported application
- Managing transition from legacy to modern state
  - Impact on application development team
  - Impact on user community

How DMS handles Porting

- Accepts definitions of source, target and *design* languages
  - Syntax, Semantics, Optimization Transforms and Analysis rules
- Accepts specifications of (porting) transformations
  - Written in terms of the language syntax, conditioned by analyses
  - Source-language idioms often map directly to Target-language idioms
  - Transforms for Complex idioms/OS/Library calls
    abstracted to design languages, then refined to target languages
- Parses entire source system (thousands of files!)
- Apply Porting transformations, then Optimizing transforms
- PrettyPrints the results in compilable target language form
- Test Result using Application Regression Test
- Revise transforms and repeat till done

Porting Transforms: Direct and Indirect



Reference:
[ABP85] Transformational Model of Maintenance,
IEEE Software, 1985

A few DMS porting transforms

Jovial to C

```
default source domain Jovial;
default target domain C;

private rule refine_data_reference_dereference_NAME
                (n1:identifier@C,n2:identifier@C)
                   :data_reference->expression
  = "\n1\:NAME @ \n2\:NAME" -> "\n2->\n1".

private rule refine_for_loop_letter_2
               (lc:identifier@C,f1:expression@C,
                f2:expression@C,s:statement@C)
                     :statement->statement
  = "FOR \lc\:loop_control :
         \f1\:formula BY \f2\:formula; \s\:statement“
    ->
       "{ int \lc = (\f1);
          for(;;\lc += (\f2)) { \s } }“
        if is_letter_identifier(lc).
```

Domain Name

Pattern Variables

Source Domain Syntax

Target Domain Syntax

Porting Transforms in Action

Jovial to C

JOVIAL Source:

```
FOR i: j*3 BY 2 ;
    x@mydata = x@mydata+I;
```

Translated C Result:

```
{ int i = j*3;
  for (;;i+=2)
     { mydata->x = mydata->x + i}
```

Typically lots of small transforms for full translation

~2500 rules to translate full Jovial

A More Complex Example

Jovial to C

```
START
  TABLE TFP'D'TWRDET (1:109,12:37);
    BEGIN
      % Main status boolean %
      ITEM TFP'G'TWRDET STATUS (V(YES),V(NO));
    END
    TYPE TFP'D'TWRDET'TABLE TABLE (7:23) W 3;
      BEGIN
        ITEM TFP'ITM S 3 POS(0,3); "cube axis"
      END

  %begin proc%
  PROC PROC'A(c1) S;
    BEGIN
      ITEM match'count U 6;
        %an item%
      ITEM c1 C 5; "parameter value"
      ITEM c2 C 7;
      IF c1 <= c2 AND c2 > c1;
         match'count = UBOUND(TFP'D'TWRDET,0) +
                       UBOUND(TFP'D'TWRDET'TABLE,0);
      "result off by 1 so adjust"
      match'count = match'count+1;
      BEGIN
        match'count=match'count/2;
        PROC'A = match'count; % return answer %
      END "cleanup and exit";
    END "end proc"

TERM
```

***packed tables with bit offsets,
typedefs, functions,
string operations***

```
#include "jovial.h"
static struct
  { /* Main status boolean */
    enum { V(yes$OF$tfp_g_twrdet$OF$tfp_d_twrdet),
           V(no$OF$tfp_g_twrdet$OF$tfp_d_twrdet) }
                          tfp_g_twrdet _size_as_word;
          } tfp_d_twrdet[109][26];
typedef union
  { W(3);
    struct
      { POS(0, 3) S(3) tfp_itm:4 _align_to_bit; /* cube axis */
      };
   } tfp_d_twrdet_table[17];

static S proc_a(C(5) c1);
/* begin proc */
static S proc_a(C(5) c1)
{ __typeof__(proc_a(c1)) RESULT(proc_a);
  _main:
    { U(6) match_count;
      C(7) c2;
      if (CHARACTER_COMPARE(BYTE_CONVERT(C(7), c1, 7), c2) <= 0
          && CHARACTER_COMPARE(c2, BYTE_CONVERT(C(7), c1, 7)) > 0)
        match_count = UBOUND(tfp_d_twrdet, 2, 0) + 16;
      /* result off by 1 so adjust */
      match_count = (S(6))match_count + 1;
      { match_count = (S(6))match_count / 2;
        RESULT(proc_a) = (S(6))match_count; /* return answer */
      } /* cleanup and exit */
      ;
    }
  _return:
    return RESULT(proc_a);
} /* end proc */
```

***Equivalent C
(used with hand-coded macro library)***

JOVIAL to C port statistics

- Realtime Avionics System for Exotic military airplane
  - 16 bit CPU to 32 bit CPU
- Input: 350K SLOC, 480 files
  - Not counting RTOS
- Translation time: 20 CPU hours Pentium 4
  - *Every* file translated
- Output: 400K SLOC
  - RTOS reimplemented by COTS
  - Added 16 hand-written files
- Manual changes: 15K SLOC ~~ 3%
  - Made to input files
  - 50% deletions
- Application now running in ground simulation tester
- Customer planning to translate ~~4 *million* SLOC

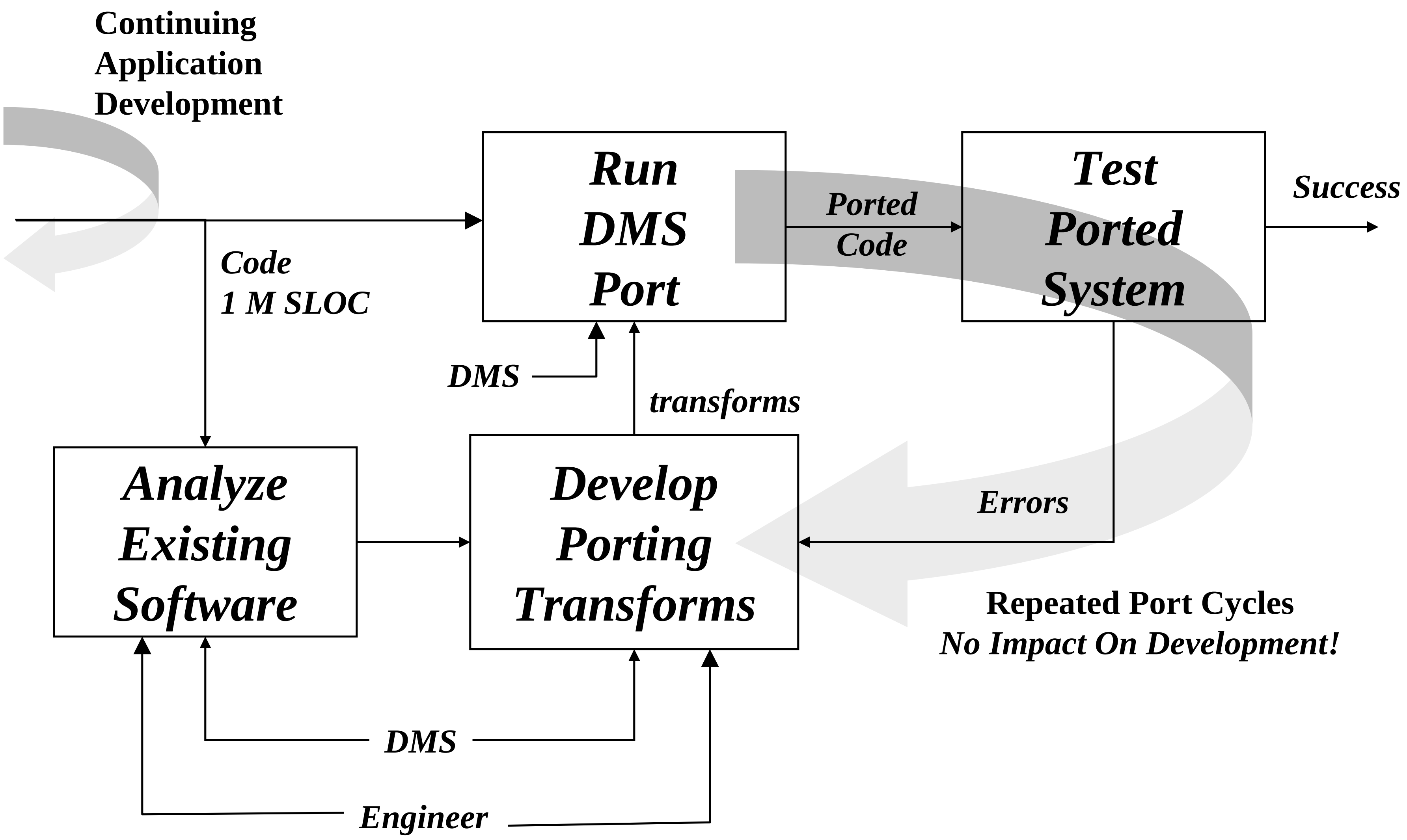
Assessing Porting Costs

Use DB of historical indicators

DMS approach: count instances of RSL patterns

```
default base domain Cpp;

pattern pointer_arithmetic(t:type,
                           e1:expression,
                           e2:expression)=
       “(unsigned int)(\t *)\e + \e2”.

pattern green_screen_call(s:string_literal)=
       “puts(\s);” if contains_termcap_codes(s).
```

Porting Process



Additional Porting Support

- Can construct full system hyperlinked crossreference
  - Programmers spend 50% of time just *looking* at code
  - For original and for translated system
- Dead/duplicate code detection/removal
  - Typically 10-20% redundant code in a system
  - Removing it → 10-20% less code to port
- Custom code reorganization as needed
  - Collection/optimization of DB calls
  - System Refactorization/Remodularization
- Need to test, test, test the ported system
  - Given tests, what part of system is tested?
  - Code coverage answers that question
  - Can build coverage tools for arbitrary languages
- Synthesize database translation code handling format changes
  - Heirarchical to Relational
  - Relational table cleanup

Advantages of Automated Porting

- Faster, better, more reliable conversion
  - Automated translation on scale
  - Uses results of all lessons learned while porting
- No loss of application function
  - Rebuilding from scratch loses useful features
  - No re-training of user base
- App. Development team undisturbed until switch
  - Organization can continue to meet dev. needs
- Incremental testing
  - Progress indication during project, not at end
- Easy distinction of feature creep and porting
  - Helps control conversion time frame

Advantages of Porting with DMS

- Can handle arbitrary languages and dialects
  - Define syntax (variants) and transforms
  - Predefined domains for common languages
- Can handle mixed languages in or out
- Can be customized to for source/target APIs
- Can be customized for needed changes
  - Green screen, DataBase changes, restructuring
- Can handle very big applications
  - Tens of thousands of source files

Porting by DMS is practical

- Enabled by generalized compiler technology
  - Requires:
    - Specification of source, target and design languages
    - Specification of *inspectable* transforms
  - Automates:
    - Source file parsing and prettyprinting of results
    - Application of sets of transforms
  - Scalable, fast, repeatable
    - Transform thousands of files/millions of lines in one day
    - Iterative development/testing of porting transforms/result
- Organizational benefits
  - Development team not disturbed by porting team
  - Application functionality preserved → happy users!
  - Far more cost effective than hand translation

When to Port using DMS

- When manual conversion too expensive or long
- When application functionality *must* be preserved
  - Automated tool carries out transforms reliably
  - Your team can inspect transforms for correctness
- Using languages not handled by other vendors
  - Different dialect
- Using language mixes not handled by other vendors
  - Need mixed language application translated as a unit
- Need to make massive source change while porting
  - Using custom 3rd party packages in source/target
  - Source code must change to match
    - Heirarchical DB -> SQL
    - Green Screens to GUI Screens

SD can enable a Successful Port

- Consulting on porting process
  - Assessment, Planning, Execution
- Supplying DMS to porting team
- Supplying language definitions
  - C, C++, Ada, Fortran, VisualBasic, XML, …
- DMS Training
- Contract implementation of parts of port
  - Design languages, transforms, custom steps

www.semanticdesigns.com

DMS Domain for HP Transact Parser Rules

```
PUT_statement = 'PUT' data_item_target ',' 'LIST' '=' arithmetic_expression ';' ;

PUT_statement = 'PUT' UPDATE_destination PUT_options ';' ;
PUT_statement = 'PUT' '(FORM)' FORM_selector PUT_FORM_options ';' ;

PUT_options = ;
PUT_options = ',' PUT_option PUT_options ;
PUT_option = 'STATUS' ;
PUT_option = list_range_option ;
PUT_option = error_handler_option ;
PUT_option = 'LOCK' ;
PUT_option = 'NOMSG' ;
PUT_option = record_number_option ;

PUT_FORM_options = PUT_FORM_option ;
PUT_FORM_options = PUT_FORM_options ',' PUT_FORM_option ;
PUT_FORM_option = 'STATUS' ;
PUT_FORM_option = list_range_option ;
PUT_FORM_option = FORM_option ;
PUT_FORM_option = 'CURRENT' ;
PUT_FORM_option = function_key_transfer ;
PUT_FORM_option = wait_for_function_key ;
```

~~ 1200 Rules