

Practical, Automated Large Scale Software Reengineering with DMS

Jing Zhang

Slides from Dr. Ira D. Baxter

Semantic Designs, Inc.

www.semanticdesigns.com

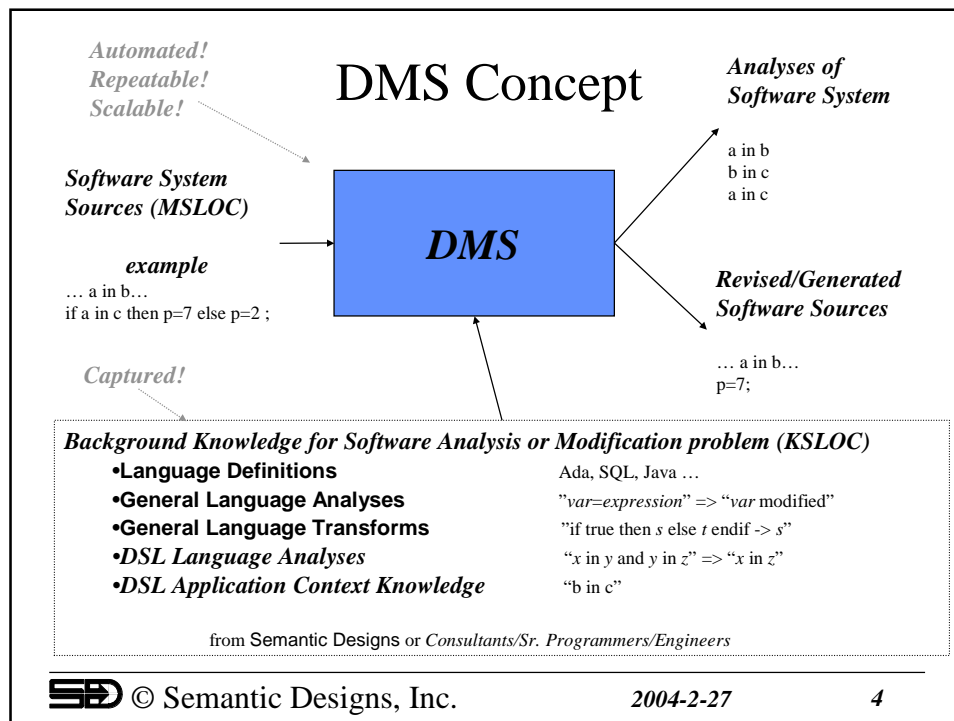
Feb.24th, 2004

Modern Software Engineering

- Large software system in multiple languages
- 80% Maintenance/Enhancements
- Little accurate design documentation
- Largely manual effort
- *How to:*
 - Understand software structure?
 - Reorganize structure to enable change?
 - Make sweeping changes?
 - Make reliable changes?

DMS[®] Software Reengineering Toolkit

- Customized, *automated* analysis, modification, porting or generation
 - For sources for large **scale** software systems
 - Scalable to millions of source lines, tens of thousands of files
 - Parallel processing foundations to support scale
 - Handles many *and mixed* languages simultaneously
 - **C, C++, Java, COBOL, HTML**, Ada, Fortran, SQL, XML, assembler, ...
 - Generalized compiler technology conveniently *integrated*
 - Parsing, Analyzing, *Transforming*, Prettyprinting
 - Enables practical customization for desired automation task
 - Predefined support for standard computer languages



Overview

- **DMS® Software Reengineering Toolkit**
 - Defining notations (“domains”) for specs and legacy systems
 - Parsing and prettyprint
 - Transformation rule mechanics
- **Applications for Software Quality Improvement**
 - C++ preprocessor conditional removal
 - Software Test Coverage
 - Cross Reference and Dead Code
 - Refactoring Java
 - Automatic Code Generation (XML Parsers)
 - Clone Detection/Removal
 - Porting application software to new languages
 - Restructuring Legacy Applications

DMS Domain Parts

- **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**
 - *Optimizations* (how to optimize in the domain)
 - **Refinements** (how to transform IF to another IF)
 - *Analyzers* (how to analyze in the domain)

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!)



Parsing to Abstract Syntax Trees

A Program Representation analyzable by Computers

- Use DMS grammar domain to define language syntax
- DMS generates lexer/parser automatically
- Parser reads source file(s)
 - Captures comments
 - Carries out lexical conversions (e.g, FP text -> IEEE binary fp)
 - Builds Abstract Syntax Tree
 - Records Position of every node (file, line, col)
- Present capability for the following domains
 - *Specification:* Spectrum, BNF, Rose Models
 - *Technology:* XML, IDL, SQL
 - *Implementation:* C/C++, COBOL, Java, Ada, VB6, Fortran, Verilog



A Simple Java Program

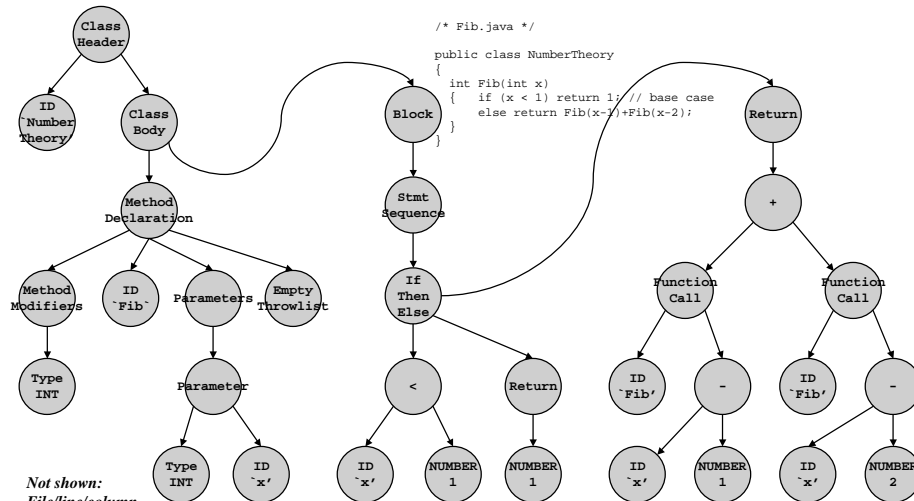
```

001  /* Fib.java */
002
003  public class NumberTheory
004  {
005      int Fib(int x)
006      {   if (x < 1) return 1; // base case
007          else return Fib(x-1)+Fib(x-2);
008      }
009  }
010

```

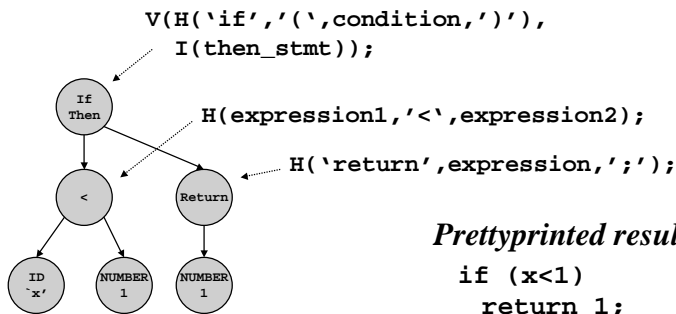
Abstract Syntax Tree (AST) for Fib Class

... free of lexical properties ('text shape') of program ...



PrettyPrinting: “AntiParsing”

- Conversion of AST back to text file
- Handles indentation, comments, literal formats...
- Uses DMS Box language to compose PP fragments



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

DMS transforms work on ASTs, not text

Not fooled by any lexical properties of text!

To modify programs:

1) define transforms

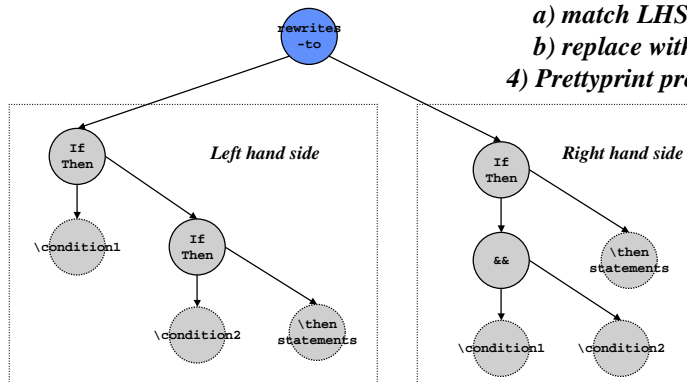
2) Parse program

3) Apply transforms

a) match LHS pattern

b) replace with RHS substitution

4) Prettyprint program



Overview

- **DMS[®] Software Reengineering Toolkit**
 - Defining notations (“domains”) for specs and legacy systems
 - Parsing and prettyprint
 - Transformation mechanics
- **Applications for Software Quality Improvement**
 - C++ preprocessor conditional removal
 - Software Test Coverage
 - Cross Reference and Dead Code
 - Refactoring Java
 - Automatic Code Generation (XML Parsers)
 - Fast HTML generation using XSLT
 - Clone Detection/Removal
 - Porting application software to new languages

Software Test Coverage

- Analysis of code executed by test cases
 - *Non*-executed code likely to have flaws
- 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
- Secondary problem: exercising all parts
 - Exercising individual part
 - Generation of tests from specifications

Test Coverage by Marking visited Blocks

```
bool fibcached[1000];
int fibvalue[1000];

int fib(int i)
{ int t;
  switch (i)
  { case 0:
    { case 1: return 1;
    default:
      if fibcached(i)
        return fibvalue(i);
      else { t=fib(i-1);
            return t+fib(i-2);
          };
    };
};
```

Original “C” program

```
bool fibcached[1000];
int fibvalue[1000];

int fib(int i)
{ int t;
  visited[1]=true;
  switch (i)
  { case 0: visited[2]=true;
    { case 1: visited[3]=true;
      return 1;
    default:
      visited[4]=true;
      if fibcached(i)
      { visited[5]=true;
        return fibvalue(i);
      }
      else { visited[6]=true;
            t=fib(i-1);
            return t+fib(i-2);
          };
    };
};
```

Marking program

DMS transform(s) to mark program

```

default base domain C;

rule mark_function_entry(result:type, name:identifier,
    decls:declaration_list, stmts:statement_sequence) =
    "\result \name { \decls \stmts };"
    rewrites to
    "\result \name { \decls { visited[\place{\stmts\}]=true; \stmts } }.".

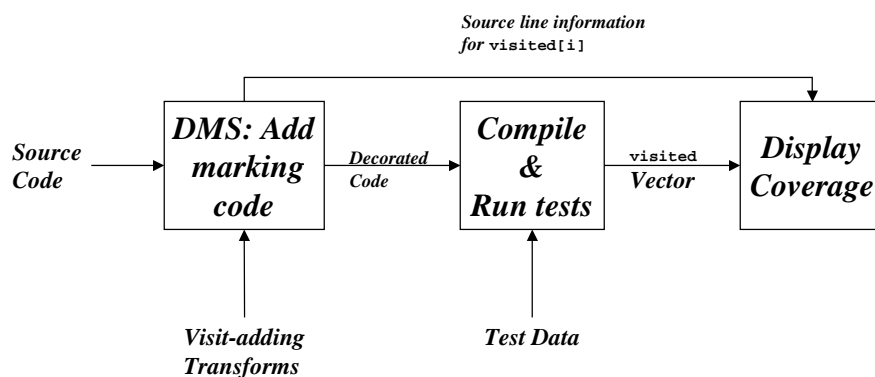
rule mark_if_then_else(condition:expression; tstmt:statement; estmt:statement) =
    "if (\condition)\tstmt else \estmt;"
    rewrites to
    "if (\condition)
    { visited[\place{\tstmt\}]=true; \tstmt}
    else {visited[\place{\estmt\}]=true; \estmt};".

rule mark_while_loop(condition:expression, stmt:statement) =
    "while (\condition) \stmt"
    rewrites to
    "while (\condition) { visited[\place{\stmt\}]=true; \stmt }.".

rule mark_case_clause(e:expression, stmts:statements) =
    "case \e: \stmts"
    rewrites to
    "case \e: { visited[\place{\stmts\}]=true; \stmts }.".

```

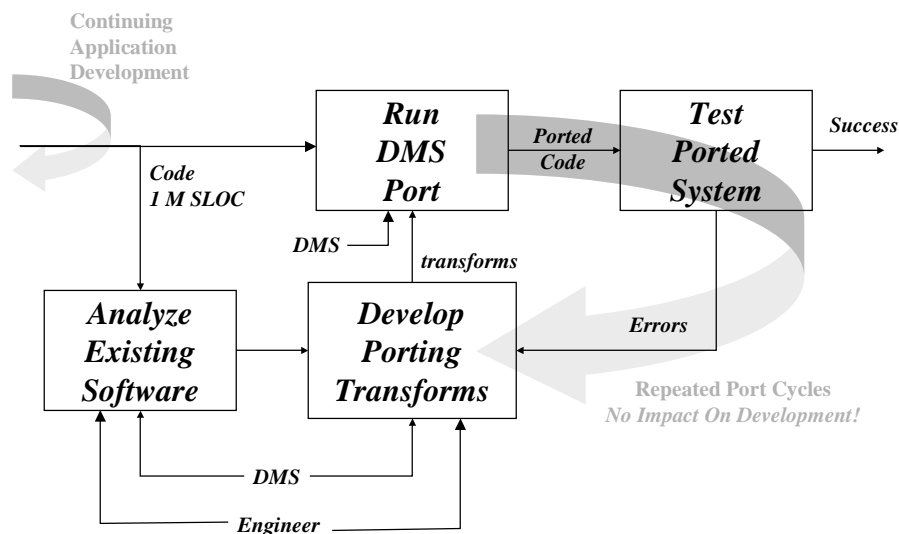
Test Coverage Tool Flow



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
- MODCOMP ASM → C
 - Defense Radar modernization; 12 computer languages!
- Verilog → VHDL
 - Reuse of Chip Design in new context

Porting Process



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).
```

Annotations:

- Domain Name (points to Jovial)
- Pattern Variables (points to n1, n2)
- Source Domain Syntax (points to the rule parameters)
- Target Domain Syntax (points to the C code output)

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
~1500 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"

```

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

```
#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)*



DMS: Conclusion

- Useful to automate analysis/modification of programs
 - Many possible custom reengineering possibilities
 - A key technology for software quality improvement
- Need generalized compiler-like infrastructure
 - Definable parsers, prettyprinters, transforms
 - Must *scale* to application systems with MSLOCs

www.semanticdesigns.com/Products/DMS/DMSToolkit.html

WhyDMSForSoftwareQuality.pdf



Where can *DMS* be applied?

- Program modification
 - *Application evolution*
 - Functionality change
 - Performance change
 - Technology change
 - Massive changes
 - Porting: new language, target APIs, ...
 - Restructuring: *Clone removal*, Y2K fix, ...
 - Optimization: Dead code, parallelize, ...
 - Customize reusable component in new context
- Program Analysis
 - Metrics
 - SLOC, conditional, complexity
 - Organization style checking
 - Programming information extraction
 - Clones, Slices, Call Graphs, Side effects
 - Domain information extraction
 - Business rules
 - Idiom recognition
 - Semantic Faults
 - erroneous/dead/useless code
- Domain-specific *program generation*
 - Partial Differential Equation solvers
 - Factory control synthesis
 - Entity-Relationship compilers
 - DB conversion generators
 - Protocol compilers
 - Automated Test Generation
- Legacy code reverse engineering
 - design recovery to domain abstractions
 - aid code understanding
 - enable application evolution
 - *Incremental* design capture
 - reusable component extraction
 - component extraction for domains
 - Legacy mergers
 - Unify data schemas
 - Modify applications
 - Convert existing data
 - Business rule extraction
 - Make explicit, easy to read/change