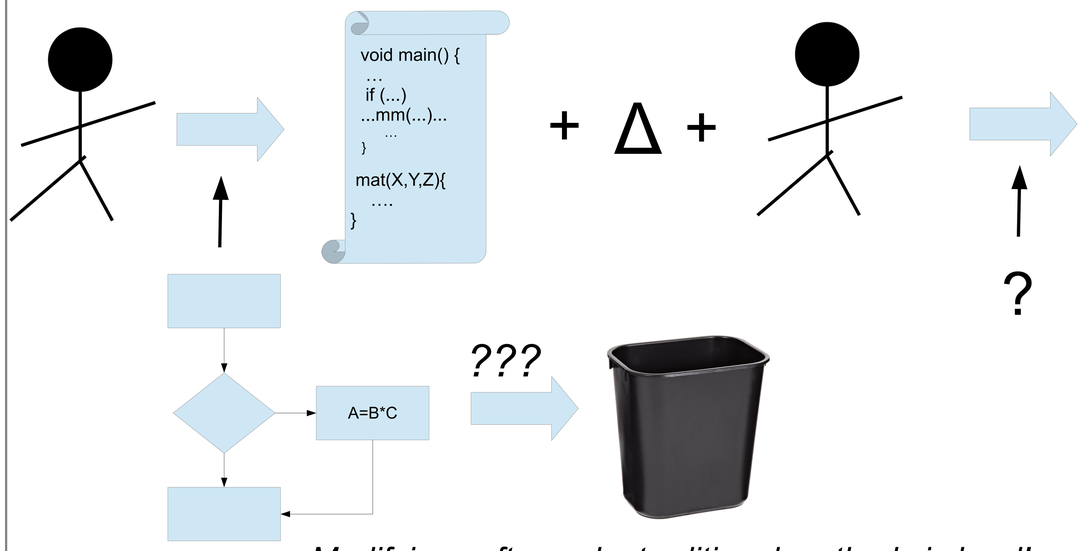
Design Capture as a Paradigm
Design Recovery as a Practical Approach

Ira D. Baxter, Michael Mehlich, Semantic Designs

1



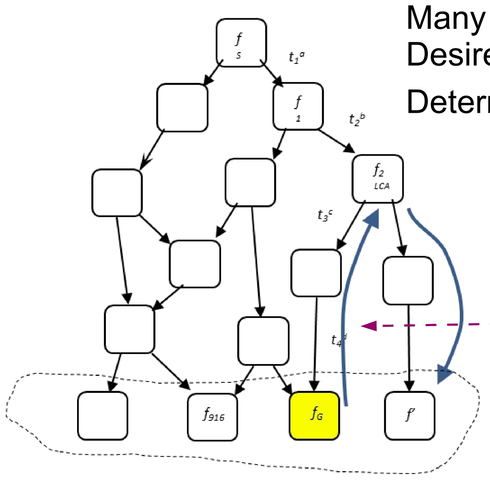
Modifying software by traditional methods is hard!

2

Design space for specification f_s:

Many functionally equivalent implementations
Desired solution f_G has right performance
Determined by chosen decisions t_i

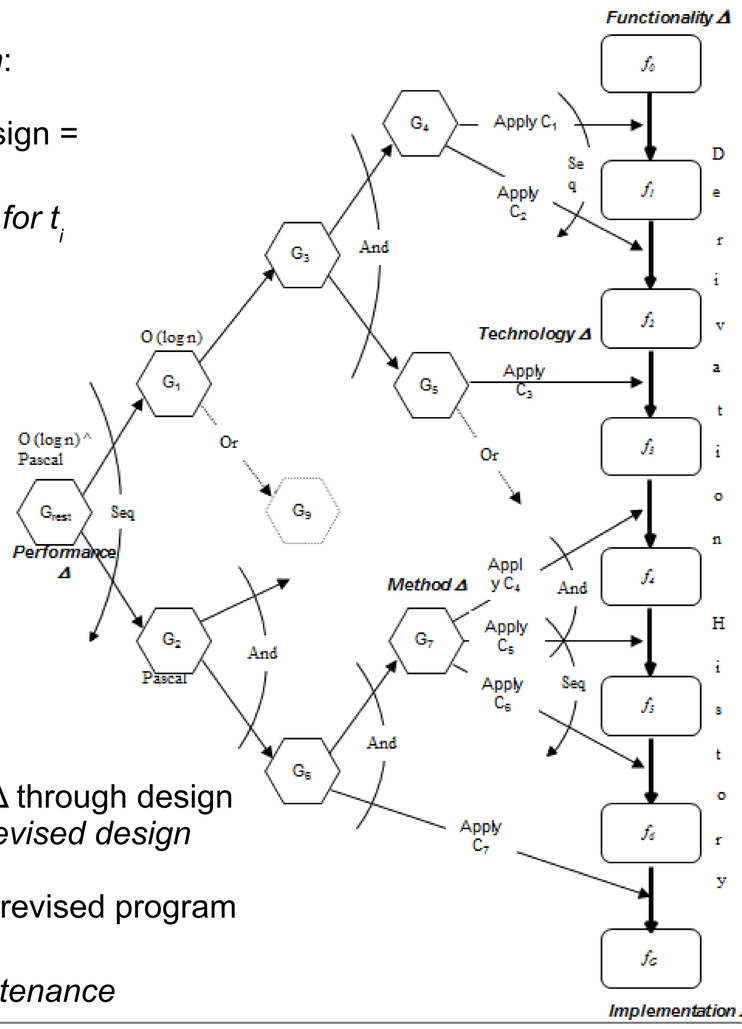
Can achieve with
program transformation systems



3

New paradigm:

1. Capture design = $f_S + t_i$ + *rationale for t_i*



2. Push *typed* Δ through design *to produce revised design*

3. Run t_i to get revised program

= *Design Maintenance*

4

Practicality: What about legacy code with lost design?

design:={};
for each Δ:
augment design by *reverse engineering back to abstraction*
apply "push Δ"
end

5

Abstracting low level code to concepts:

Programs can Represented as Data Flow computations
Code idioms can be represented as patterns of Data Flow computations
Matching idioms = pattern isomorphic match to application sub-graph
Design choices selecting among implementations

```
default base domain C~~GCC4.
public generic type Order
   = enum { `Row Major`, `Column Major` };
public generic type Representation
   = enum { `1D Array`, `2D Array`,
            `Array of Pointers to Arrays` };
public generic type Implementation
   = enum { `Explicit Code`, `BLAS` }.
private data flow pattern ReadElement
   <r:Representation, o: Order>
   (m: IDENTIFIER <~, -- matrix
    r: IDENTIFIER, -- rows in matrix
    c: IDENTIFIER, -- columns in matrix
    i: IDENTIFIER, -- row of interest
    j: IDENTIFIER, -- column of interest
    v: IDENTIFIER ~> -- value read
    ):statement
   = case
       when r == `1D Array` then
         case
           when o == `Row Major` then
             "\v = \m[\i*\c+\j];"
           when o == `Column Major` then
             "\v = \m[\j*\r+\i];"
         esac
       when r == `2D Array` then
         case
           when o == `Row Major` then
             "\v = \m[\i][\j];"
           when o == `Column Major` then
             "\v = \m[\j][\i];"
           esac
       when r == `Array of Pointers to Arrays`
         case
           when o == `Row Major` then
             "\v = (*\m[\i])[\j];"
           when o == `Column Major` then
             "\v = (*\m[\j])[\i];"
         esac
     esac.

private data flow pattern WriteElement
    ...
```

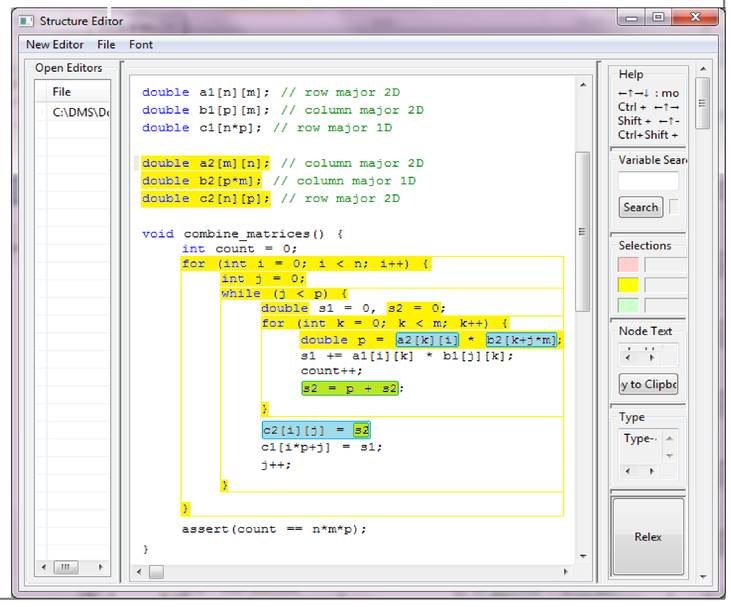
Dataflow Pattern for MatrixMultiply in C (1)

```
private data flow pattern AddInto
    (t: IDENTIFIER, -- target being updated
     s: IDENTIFIER   -- value to add to target
     ):statement
     =  "\t += \s;" ? "\t = \t + \s;".
public data flow pattern MatrixMultiply
    <i: Implementation,
     ra: Representation, oa: Order,
     rb: Representation, ob: Order,
     rc: Representation, oc: Order>
    (n: IDENTIFIER <~, -- in: matrix size parameter
     m: IDENTIFIER <~, -- in: matrix size parameter
     p: IDENTIFIER <~, -- in: matrix size parameter
     a: IDENTIFIER <~, -- in: source matrix
     b: IDENTIFIER <~, -- in: source matrix
     c: IDENTIFIER ~> -- out: target matrix
    ):statement
    = case
        when i == `Explicit Code` then
           [i: IDENTIFIER, j: IDENTIFIER, k: IDENTIFER,
            s: IDENTIFIER,
            ta: IDENTIFIER, tb: IDENTIFIER, tc: IDENTIFIER.
            "for (\i=0; \i<\n; \i++)
               for (\j=0; \j<\p; \j++) {
                  \s=0;
                  for (\k=0; \k<\m; \k++) {
            \ReadElement\<ra,oa>\(\a\,\n\,\m\,\i\,\k\,\ta\)
            \ReadElement\<rb,ob>\(\b\,\m\,\p\,\k\,\j\,\tb\)
            \tc = \ta * \tb;
            \AddInto\(\s\,tc\)
                                        }
            \WriteElement\<rc,oc>\(\c\,\n\,\p\,\i\,\j\,\s\)
            }"
           ]
        when i == `BLAS` then
           ...
        esac.
```

Dataflow Pattern for MatrixMultiply in C (2)
Represents 3456 variants with decisions

Mockup at right shows 1 matched idiom
in code with two fused multiplies

DF Matching in use now for Dow Chemical
Reverse engineer code to Process Control Design
Applications of size 50K SLOC
111 different Data Flow patterns
One pattern ~~ 10K SLOC with 17 design choices
One patttern ~~ 3K SLOC with 35 design choices
Expected to use in 1200 factory control migrations



G. Arango, I. Baxter, P. Freeman and C. Pidgeon, *TMM: Software Maintenance by Transformation*, IEEE Software, May, 1986.
I. D. Baxter, *Design Maintenance Systems*, Communications of the ACM 35(4) ACM, April 1992.
I. D. Baxter and M. Mehlich, *Reverse Engineering is Reverse Forward Engineering*, Working Conf. on Reverse Engineering, IEEE, 1997.
I. D. Baxter, **Transformational Maintenance by Reuse of Design Histories**, Ph.D. dissertation, CS. Dept., UC Irvine, Irvine, CA, 1990
E. Kant, *Synthesis of mathematical-modeling software*, IEEE Software 10(3), May 1993.
C. Rich and C. Waters, *The Programmer's Apprentice: A Research Overview*, IEEE Press Computer 21(11), Nov 1988
H. Shrobe, B. Katz, Randall Davis, *Towards A Programmer's Apprentice (Again)*, Proc. 29th AAAI Conference on Artificial Intelligence
www.semanticdesigns.com/Products/DMS/DMSToolkit.html

© Semantic Designs