ELEC 875
Design Recovery
and
Automated Evolution

Week 1 Class 2
Modeling
Papers for next week


Design Recovery Architecture

- Src Code
- Extractor
- Design Model
- Reporter
- Analysis
- Reports
Design Recovery Architecture

- Source Code
- Extractor
- Design Model
- Reporter
- Analysis
- Reports
Modeling - ER

managedBy

Employee

No: int

Department

Name: string
Modeling - Extended ER

managedBy

Employee
No: int

Department
Name: string

Part Time

Full Time
Benefits: xyzzy

Temp
Hours: int
Modeling

• In traditional design (forward engineering), we model the problem domain and incorporate that model into the software in some manner.
  ◊ OOAD
  ◊ SA&D

• In design recovery, the problem domain is software. Our model will consist of entities that represent software artifacts (data is a program)

• Long Term Goal: to tie the model extracted from the code to a traditional problem model
Base Model

- Entities and Relations in the Base Model directly represent software artifacts
  ◊ source code elements

- Example Entities
  ◊ variables
  ◊ procedures
  ◊ types
  ◊ statements
Base Model

• Example Relations
  ◊ calls (procedure calls a procedure)
  ◊ references (procedure references a variable)
  ◊ isFieldOf (field to structure or class)
  ◊ hasType (type of variable or function)
  ◊ ifPart (if statement ⇒ statement)
Base Model - Notes

- some entities have natural names
  - variables
  - procedures
  - types
    - names may be predefined or user defined

- some entities do not have natural names
  - statements
  - blocks
  - constants
Base Model - Example

file main.c
    void printf(char *, ...);
    char * foo(int);
    int main(int argc, char **argv){
        printf("hello world%s", foo(3)
    }

file foo.c
    char * foo(int x){
        return ("!\n");
    }
Base Model - Example

Entities:
- Files: main.c, foo.c
- Functions: foo, main
- Variables: argc, argv, x
- Prototypes: foo, printf
- Constants: “hello world%s”, “!\n”
- Types: void, int, char*, char**, char

Relations:
- Contains: (main.c, printf), (main.c, main), (main.c, foo)
- Calls: (main, foo)
- Parameter: (main, argc), (main, argv), (foo, x)
- Argument: (foo, 3)
- HasType: (main, int), (foo, char*), (argc, int), (argv, char**), (x, int), (printf, void), (foo, char*)
Base Model - Issues

- Unique Naming
  - some entities have the same name
  - scoping
  - name spaces (Java, C, C++)
  - Model is a database, need a key for each entity
  - different entity sets - keys needed only for same entity sets and for entity sets that share relations
  - solutions:
    - unique id for each entity (CPPX, Columbus)
    - name derived from scope (LS/2000)
Base Model - Issues

• Resolution
  ◊ sample model cannot connect arguments to parameters (more than one call? more than one argument?)
  ◊ Return value of foo?

• Organization
  ◊ Database practice - organize database to answer common queries
  ◊ any given organization makes some queries hard, other queries easy
Design Recovery Architecture

Src Code → Extractor → Design Model

Reported → Analysis → Reports
Derived Model

• built on top of the base model
  ◊ derived from information in the base model
  ◊ new relations between entities
  ◊ new entities for existing entity types
  ◊ new entity types
  ◊ new attributes

• Two types of derived information
  ◊ deterministic computed information
    - implementation semantics, storage semantics
  ◊ inferred information (heuristics)
Derived Model - Computed

- storage semantics
  ◊ programmers can and do play storage games

```c
struct xyzzy{
    int x;
    float y;
};
```

- x is at offset 0 and is 4 bytes long
- y is at offset 4 and is 4 bytes long

- Big Endian/Little Endian
Derived Model - Computed

• storage semantics
  ◊ programmers can and do play storage games

union xyzzy{
  int x;
  float y;
};

- x is at offset 0 and is 4 bytes long
- y is at offset 0 and is 4 bytes long

• x and y occupy the same memory
Derived Model - Computed

struct xyzzy{
    int type;
    union {
        struct {
            ...
            int x;
            ...
        } option1;
        struct {
            ...
            int y;
            ...
        } option2;
    } detail;
};

what if fields X and Y have the same offset??
what if the programmer intends them to have the same offset??
Derived Model - Computed

- **BCD** - binary coded decimal
- **COMP-3** - BCD + Sign Nibble

\[
\begin{array}{ccccccccc}
8 & 1 & 0 & 1 & 5 & 2 & 9 & 7 & 3 & 2 \\
8 & 1 & 0 & 1 & 5 & 2 & 9 & 7 & 3 & \pm
\end{array}
\]
Derived Model - Computed

- Cobol Data structures

01 A.
  05 B  PIX XX.
  05 C.
    10 D  PIC X.
    10 FILLER PIC X(3).
  05 F  PIC 9(4).
  05 G REDEFINES F PIC XXXX.
Derived Model - Computed

- BCD - binary coded decimal
- COMP-3 - BCD + Sign Nibble

01 CONV-REC.
   05 NUM-VAL PIC 99 COMP-3.
   05 ALPHA REDEFINES NUM-VAL.
      10 ALPHA-VAL PIC X.
      10 FILLER PIC X.

MOVE INBYTE to ALPHA-VAL.
DIVIDE NUM-VAL BY 10.
Base Model - Resolution Issue

Relations:
Contains: (main.c, printf), (main.c, main), (main.c foo)
Calls: (main, foo)
Parameter: (main, argc, 1), (main, argv, 2), (foo, x, 1)
Argument: (foo, 3, 1)
HasType: (main, int), (foo, char*), (argc, int), (argv, char**), (x, int), (printf, void), (foo, char*)

\[ x = 3 \]
Base Model - Resolution Issue

file main.c
    void printf(char *, ...);
    void bar(int, int);
    int main(int argc, char **argv){
        foo(2, 3);
        foo(atoi(argv[1]), atoi(argv[2]));
    }

file foo.c
    char * foo(int x, int y){
        ...;
    }
Base Model - Resolution Issue

file main.c
    void printf(char *, ...);
    void bar(int, int);
    int main(int argc, char **argv){
        foo(2, atoi(argv[2]));
        foo(atoi(argv[1]), 3);
    }

file foo.c
    char * foo(int x, int y){
        ...
    }
Derived Model - Inferred

• Use other information to infer information about entities.
• Y2K - Dates
  ◊ Names of Variables and Functions
  ◊ Storage Types of Fields
  ◊ Interaction with OS or with known API
  ◊ Domain Dependent Patterns

01 MTGSTD   PIC 9(6).
Derived Model - Inferred

- Use other information to infer information about entities.
- Y2K - Dates
  - Names of Variables and Functions
  - Storage Types of Fields
  - Interaction with OS or with known API
  - Domain Dependent Patterns

01 CURRENT-DATE-YYMMDD PIC 9(6).
01 MTGSTD          PIC 9(6).

IF MTGSTD > CURRENT-DATE-YYMMDD
Derived Model - Inferred

- Move to higher level of abstraction
- Business Rules, Business Types
- Goal:
  - Link to problem model for program
  - Employee Number, Customer Name, Customer Address
  - Where are they used?
  - How are they related?