ELEC 875
Design Recovery
and
Automated Evolution

Week 1
Modelling
Papers for next week

System Evolution

• Real systems evolve over time
  ◊ not just bug fixes
  ◊ environment changes over time
  ◊ new/old features
  ◊ legacy systems

• Design Recovery
  ◊ Recover design level facts about software artifacts

• Automated Evolution
  ◊ semi-automated changes to systems
Course Structure

- 5 weeks of lectures
  - background material (readings)
  - basis
- Midterm (25%)
  - based on lectures
- Advanced Readings
  - reports (30%) and discussion (15%)
- Project (30%)
  - Project Presentation
  - TXL
  - Other technologies (Rascal/Rational/Custom)
Legacy

*noun* A sum of money, or a specified article, given to another by will; anything handed down by an ancestor or predecessor

*adj* associated with something that is outdated or discontinued
Legacy Systems

- Software
  - inherited (more than one generation of developers)
  - valuable
    - significant resources to replace
    - significant risk to replace

- Problems:
  - original developers may not be available
  - older development methods used (outdated?)
  - extensive modifications
  - missing or outdated documentation
  - studies show 50% – 75% of available effort (domain dependent)
Legacy Systems

- Traditionally viewed as old and expensive
  - prohibitively expensive
  - only a matter of time before they must be replaced
  - drain on resources
  - outdated
Legacy Systems

• Alternate View:
  ◊ crown jewels
  ◊ organizations that have not let their legacy systems get out of control (i.e. most large financial institutions) have a significant advantage over other organizations
  ◊ system is working and evolves
Legacy Systems

• Continuous Evolution
  ◊ You own a wooden ship. You replace each board in the ship each time you sail. At what point in time do you have a new ship?
    - Ship of Theseus (Plutarch)
  ◊ Space Shuttle
  ◊ Operating Systems
  ◊ Compilers
  ◊ Financial Systems (systems written in 1962 are still running).
Design Recovery

- Recover Design Information from Source Artifacts.

Source Artifacts:
Design Recovery

• Recover Design Information from Source Artifacts.

Source Artifacts:
  ◊ source code
Design Recovery

• Recover Design Information from Source Artifacts.

Source Artifacts:
◊ source code
◊ database definitions
Design Recovery

• Recover Design Information from Source Artifacts.

Source Artifacts:

◊ source code
◊ database definitions
◊ screen definitions (also web page definitions)
◊ communication definitions
Design Recovery

• Recover Design Information from Source Artifacts.

Source Artifacts:

◊ source code
◊ database definitions
◊ screen definitions (also web page definitions)
◊ communication definitions
◊ stored procedures
Design Recovery

- Recover Design Information from Source Artifacts.

Source Artifacts:

◊ source code
◊ database definitions
◊ screen definitions (also web page definitions)
◊ communication definitions
◊ stored procedures
◊ scripting languages (JCL, TCL, Shell, DOS BAT)
Design Recovery

- Recover Design Information from Source Artifacts.

Source Artifacts:
- source code
- database definitions
- screen definitions (also web page definitions)
- communication definitions
- stored procedures
- scripting languages (JCL, TCL, Shell, DOS BAT)
- some forms of documentation
Design Recovery

• Recover Design Information from Source Artifacts.

Source Artifacts:

◊ source code
◊ database definitions
◊ screen definitions (also web page definitions)
◊ communication definitions
◊ stored procedures
◊ scripting languages (JCL, TCL, Shell, DOS BAT)
◊ some forms of documentation
◊ 4GL languages (application generation)
Resources

- Conferences:
  - IEEE International Conference on Software Maintenance (ICSM)
  - IEEE Working Conference On Reverse Engineering (WCRE)
  - European Conference On Software Maintenance and Reengineering (CSMR)
  - IEEE International Conference on Program Comprehension (ICPC)
  - IEEE International Conference On Software Engineering (ICSE)
  - Foundations on Software Engineering
Resources

• Journals

• Web
  ◊ many researchers in the area
  ◊ citeseer, google scholar, etc.
Design Recovery Architecture

Src Code -> Extractor -> Design Model

Extracted Design Model

Reporter -> Analysis

Analysis & Reporter -> Reports
Design Recovery Architecture
Modelling - ER

managedBy ➤

Employee

No: int

1:∞

0:1

Department

Name: string

1:∞

0:1

member ➤

Employee

No: int

1:∞

0:1

Department

Name: string

1:∞

0:1

member ➤

Employee

No: int

1:∞

0:1

Department

Name: string

1:∞

0:1

member ➤
Modelling

• 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
Two Models - Base Model

- Src Code
- Extractor
- Design Model
- Reporter
- Analysis
- Reports
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)
  - modifies (procedure changes value of a variable)
  - isFieldOf (field to structure or class)
  - hasType (type of variable or function)
  - ifPart (component of a compound 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 ("!
");
}
Base Model - Example

Entities:
- Files: main.c, foo.c
- Functions: foo, main
- Variables: argc, argv, x
- Prototypes: foo, printf
- Constants: “hello world%s”, “!
”
- 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
Two Models - Derived Model

- Src Code
- Extractor
- Design Model
- Reporter
- 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

```c
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

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.
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
Biggerstaff - Introduction

- Seminal Paper
  ◊ Discusses the General Goal
  ◊ Prototype: Desire - first step towards the goal
Already Happens

• “a common, sometimes hidden part of many activities scattered throughout the software life cycle”

• Development
  ◊ Understand similar systems
  ◊ Understand libraries and systems that interact

• Maintenance
  ◊ Understand system before making changes.

• happens in an inherently unstructured way.
Domain Expertise

• Domain Model
  ◊ Human knowledge of the application area.
    - needed for development and maintenance

◊ Tools need to abstract domain knowledge as well.
Biggerstaff

- Design recovery whenever a system is maintained
- Several Steps
  ◊ Program Understanding
    - Modules
    - Key data items
    - Software engineering artifacts
    - Informal design abstractions
    - Relate SE artifacts and informal abstractions to the code
  ◊ Population of Reuse and Recovery Libraries
  ◊ Applying Results of Design Recovery
Identify the Modules

- Not all languages have modules
- Software of any size has modules
- Variety of ways to implement modules
  - Separate files and compilation units
    - module.h module.c
    - No nested modules
    - Smaller modules (one file)
    - May be more than one implementation file
      - E.g. module1.c module2.c
- Naming convention for type, procedure or variable names
Key Data Items

- Most programs are organized around one or more specific data items.
  - Master journal record in transaction systems
  - Master account database
  - Ready, wait and device queues in operating systems
- These data items are some abstraction of the problem domain. What are they?
  - Customer, Sale, Deposit, Process
- How are they related to the modules
  - SA&D vs ADTs
  - Functional Decomposition vs OO
SE Artifacts

- The result of Design Recovery (as expressed by Biggerstaff) are design artifacts
  - dependent on shop
  - PDLs, Dataflow, Data Dictionary
  - UML?
- Does not have to match the artifacts originally used to create the system
- Artifacts must be appropriate for system
  - Consequences of a poor fit?
  - UML for 40 year old transaction system
Informal Design Abstractions

- Informal descriptions of concepts that occur in the code (automatable?)
- Design Rational
- Original Designers are not available, or it may be so long that they do not remember
  ◦ People’s version of history change over the years
  ◦ Guess
  ◦ Source Code Comments
  ◦ Existing Documentation
Relating Abstractions to Code

- Link the recovered design back to the code
- Which functions are part of which module?
- Which files are part of a UML class?
- Which data structure represents a particular informal concept

- Necessary to answer low level questions that have been abstracted out
  - needed in order to use the system
  - not designing systems from scratch, modifying existing systems.
    - modifications to the design imply modifications to particular pieces of code
Reuse and Application

- late 80’s early 90’s - big thing was code reuse
- Identify reusable parts of code
  ◊ generalize to make more reusable
  ◊ factoring and decoupling

- Biggerstaff - not just code reuse, but also design recovery reuse
  ◊ help build similar components
  ◊ help recover similar components from other systems
Analysis

• Informal Information
  ◊ semantics is not the only thing
    - turing computable argument
  ◊ real systems do not contain random code
    - they have to understand it and have some confidence that it actually works
  ◊ naming conventions
  ◊ structural conventions

• One main goal is to help humans
  ◊ don’t underestimate humans
Informal Information

```c
#include <stdio.h>
#include "h0001.h"
#include "h0002.h"
#include "h0003.h"
f0001(a0001)
    unsigned int a0001;
    {
        unsigned int i0001;
        f0002(g0005,d0001,d0002);
        f0002(a0001,d0003,d0002);
        f0003(g0001[a0001].s0001,g0001[a0001].s0002);
        g0006 = a0001;
        i0001 = g0001[a0001].s0003;
        if(!f0004(i0001) && (g0002->g0003)[i0001].s0004 == d0004)
            f0005(i0001);
    }
```

Figure 4. Function with no informal semantic clues.
Informal Information

```c
#include <stdio.h>
#include "proc.h"
#include "window.h"
#include "globdefs.h"

change_window(nw)
    unsigned int nw;
    
    unsigned int pn;
    border_attribute(cwin,NORM_ATTR,INV_ATTR);
    border_attribute(nw,NORMHLIT_ATTR,INV_ATTR);
    move_cursor(wintbl[nw].crow,wintbl[nw].ccol);
    cwin = nw;
    pn = wintbl[nw].pnumb;
    if(!outrange(pn) && (g->proctbl[pn].procstate == SUSPENDED))
        resume(pn);

Figure 5. Function with some informal semantic clues.
```
Informal Information

```c
#include <stdio.h>
#include "proc.h"
#include "window.h"
#include "globdefs.h"

change_window(nw) /*Change current window to window nw*/
    unsigned int nw; /*Number of target window*/
    { unsigned int pn;

    /*Restore border of current window to un-highlighted*/
    border_attribute(cwin,NORM_ATTR,INV_ATTR);

    /*Highlight border of new current window*/
    border_attribute(nw,NORMHLIT_ATTR,INV_ATTR);

    /*Move the physical cursor to the new window where the cursor was
    left, and make nw the current window*/
    move_cursor(wintbl[nw].crow,wintbl[nw].ccol);
    cwin = nw;

    /*Resume the process associated with the new window if it is
    suspended.* /
    pn = wintbl[nw].pnumb;
    if(!outrange(pn) && (g->proctbl)[pn].procstate == SUSPENDED)
    resume(pn);
    }
```
Idioms

• Information frames
  ◊ Conceptual model of domain
  ◊ binds in some way to source code.
    - direct
    - indirect
  ◊ drew a lot from frame based AI techniques popular at the time.
Process number, process name, process state identify elements of a process table
Address, Name, Postal Code indications of a Customer.
Desire

- linguistic patterns - lexical
  ◦ representation of informal information
  ◦ naming convention
- Structural Requirements
  ◦ presence of one component implies another
  ◦ some structures are aggregations of other structures
- Incomplete Match
  ◦ not all systems are created equal
  ◦ manual intervention
Prototype

- lower level
  - functions, files, global data items
  - definition locations, use locations
  - calls uses depends

- Components
  - parser, analysis, view generation
  - links comments to artifacts

- Viewer
  - queries link back to source code
Analysis

• Prototype is lower level
  ◊ starting point is the code
  ◊ may also include comments
• Link Back to Code
  ◊ always important
  ◊ use to modify existing code
  ◊ knowledge of design is important, but only useful if it helps you in the maintenance task
• Manual Intervention
  ◊ Design recovery includes abstract concepts.
    Until real AI is created, human mind is still king.