ELEC 875
Design Recovery
and
Automated Evolution

Week 1 Class 1
Introduction
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 (Rational Rose)
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adj associated with something that is outdated or discontinued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  - older development methods used (outdated?)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  - missing or outdated documentation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  ◦ outdated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

  - 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 now SANER))
  ◊ European Conference On Software Maintenance and Reengineering (CSMR now SANER)
  ◊ IEEE International Conference on Program Comprehension (ICPC)
  ◊ IEEE International Conference On Software Engineering (ICSE)
  ◊ Foundations on Software Engineering
Resources

• Journals

• Web
  ◊ Authors Web Pages:
    – Dr. Timothy Lethbridge (SITE, U of Ottawa)
    – Dr. Hausi Müller (CS, U of Victoria)
    and many others (check references in articles)

◊ http://citeseerx.ist.psu.edu/
◊ google scholar
Biggerstaff - Introduction

• Seminal Paper
  ◊ Discusses the General Goal
  ◊ Prototype: Desire - first step towards the goal
• Design Recovery Already Happens
  ◊ “a common, sometimes hidden part of many activities scattered throughout the software life cycle”
• Domain Expertise - Domain Model
  ◊ 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
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
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].so001, g0001[a0001].sOOO2);
    go006 = a0001;
    i0001 = g0001[a0001].s0003;
    if( !f0004(i0001)&&(gOOO2->gOOO3)[iOOOl].sOOO4 == d0004)
        f0005(i0001);
}
```

ELEC 875 – Design Recovery and Automated Evolution
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,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);
}```
Example Curses Screen (Debian)

At the moment, only the core of Debian is installed. To tune the installation to your needs, you can choose to install one or more of the following predefined collections of software. Experienced users may prefer to select packages manually.

Choose software to install:

- Desktop environment
- Web server
- Print server
- DNS server
- File server
- Mail server
- SQL database
- Manual package selection
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.
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