Using Design Recovery Techniques to Transform Legacy Systems
Design Recovery & Evolution

• Design Recovery from source
  - Recover a limited design model from software
  - Base model contains entities and relations that represent software artifacts
  - Extend the model with derived relations directed to a specific task

• Automated Evolution
  - Transformation is guided by the model constraints
  - Transformation of software artifacts to meet new constraints
  - The transformation is guided by the model

Design Recovery & Evolution
The hard part of Y2K is finding the dates!!

Changes to the software are local

Remediation is simple once the dates are known

Evolution, Why?

Ideal problem for design recovery and automated evolution

Design Recovery

Automated Evolution

Global analysis required

Why?

Year 2000 Problem
ICSM 2001 © 2001 Thomas R. Dean

- LS/2000
- Legasys solution to Y2K
- Architecture adapted to post Y2K problems
- More than 3.3 billion lines of code
- Less than 40 analysts running the software
- Licensed to several Y2K vendors worldwide
- COBOL, PL/I and RPG
- Legasys solution to Y2K

1S/2000
Lessons Learned

• Compiler Manuals are incomplete
  do not document obsolete language features still recognized by the compiler (backward compatibility)
  implemented but not documented
  Only real reference is the compiler itself
  Not a fatal error!
  does to next statement
  Cobol compiler ignores statements with errors
  Real programs contain errors

• Tools must handle errors
  Not a fatal error!!
  scans to next statement
Lessons Learned

- Design recovery is more effective when it is task directed!
- The base model extracted from code was the information necessary for finding dates!
- Other information was ignored!
- Information was necessary for finding dates!
- Reduce storage and I/O for design model facts!
- Automate most common cases!
- Don't over automate!
- Human help to resolve ambiguities!
Lessons Learned

• Data flow analysis not necessary for many problems
  • Reference flow sufficient

```c
if (a < b) {
    x = y;
} else {
    x = z;
}
```

• Condition `a < b` is not germane to the data types of `x`, `y`, and `z`.

• Data flow analysis not necessary for many problems
Lessons Learned

- Hot Spot Reports clearly exposed the results to the client.
- Elided version of the program.
  - Only lines with date issues were printed.
- Clients used as checklist.
  - Examined each hot spot.
  - Used to drive testing.
  - Client.
- Printed with highlighter on hot lines.
- Most common fixed, others left for client.
- Client.
- Hot Spot Reports clearly exposed the results to the client.
Conclusions

- Very successful Y2K tool
- Successfully applied to other evolution tasks
- Hot Spot Reports
- Task Directed
- Lessons Learned

Don't Over Automate