Software Architecture Recovery from Build Processes

Bram ADAMS
Ghislain Hoffman Software Engineering Lab, INTEC, Ghent University
http://users.ugent.be/~badams

Kris DE SCHUTTER
Lab On REengineering, University of Antwerp
http://faramir.ugent.be/~kdschutt

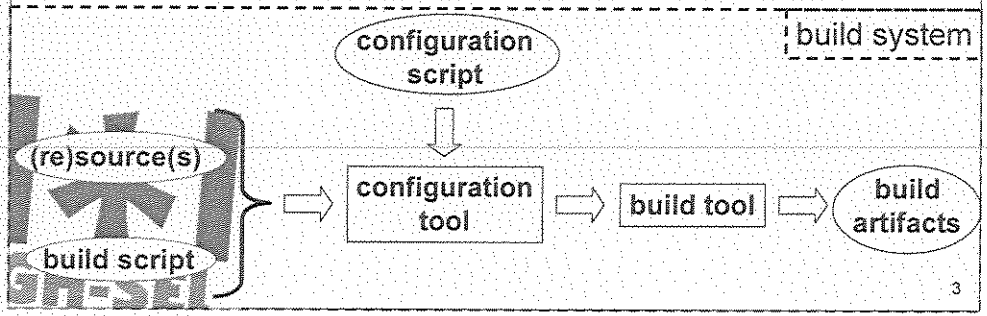
Outline

1. Why Look At Build Systems?
2. Software Architecture Recovery
3. Make
4. MAKAO
5. Rule-Based Approach
6. General Rules
7. Application-Specific Rules
8. Conclusions and Future Work

Build systems

Some history:

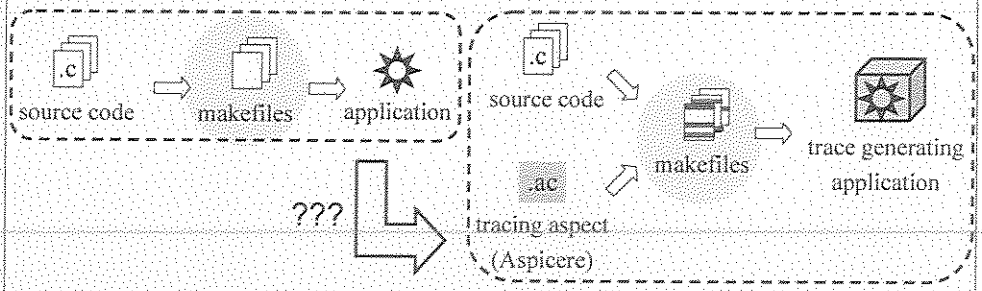
- ...-1977: ad hoc build and install scripts
- 1977: make (Stuart Feldman), most influential build tool
- later:
  - various clones (GNU Make, ...) and alternatives
  - build configuration systems like imake and GBS



3

1. Why Look At Build Systems?

Case study with Aspicere:



More general:

- how to easily modify a build system? — re-engineering
- how to gain quick insight into build process? — reverse-engineering
- how to assess general software architecture? — reverse-engineering

2. Software Architecture Recovery

Software architecture recovery:

- software and build system co-evolve
- assumptions:
  - correct makefiles
  - modular source files (no giant implementation files)

Related work:

- Build-Time Software Architecture View [Tu01]
- Dali (and Rigi) [Kazman99], Portable BookShelf [Finnigan97], and Desire [Biggerstaff89]
- [Bowman99] Linux kernel architecture
  - conceptual architecture ⇒ concrete architecture
  - tedious discovery and population of subsystems

(S. I. Feldman. "Make-a program for maintaining computer programs". Software - Practice and Experience, 1979)

3. Make

Makefile

variable

```
make_OBJECTS = ar.o arscan.o \
        commands.o dir.o ... hash.o
```

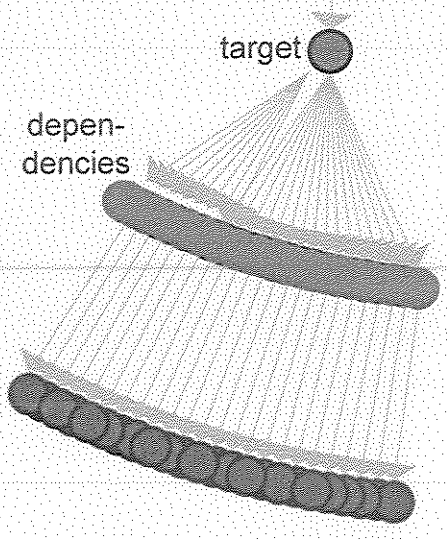
target dependencies

```
make$(EXEEXT): $(make_OBJECTS)
        @rm -f make$(EXEEXT)
        $(LINK) $(make_LDFLAGS) \
          $(make_OBJECTS) \
          $(make_LDADD) $(LIBS)
```

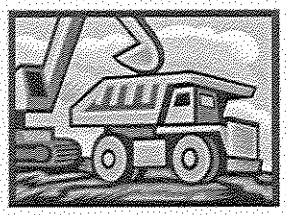
... commands rule

⇒ de facto build tool/process model!

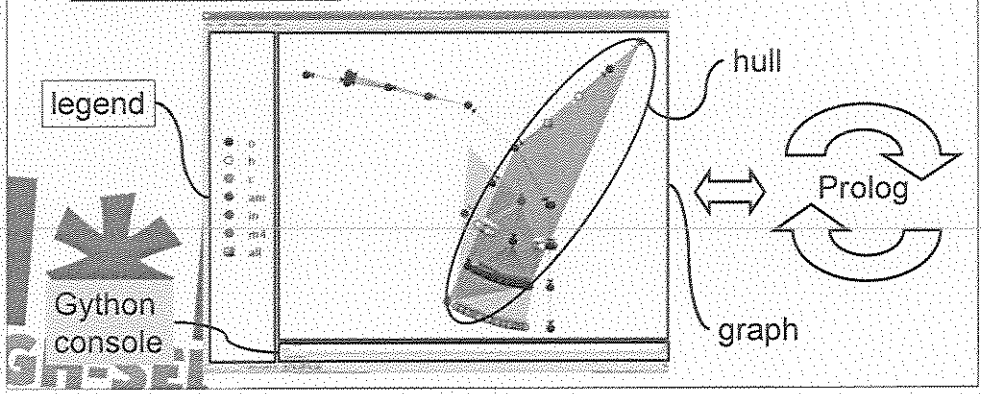
Directed Acyclic Graph (DAG)

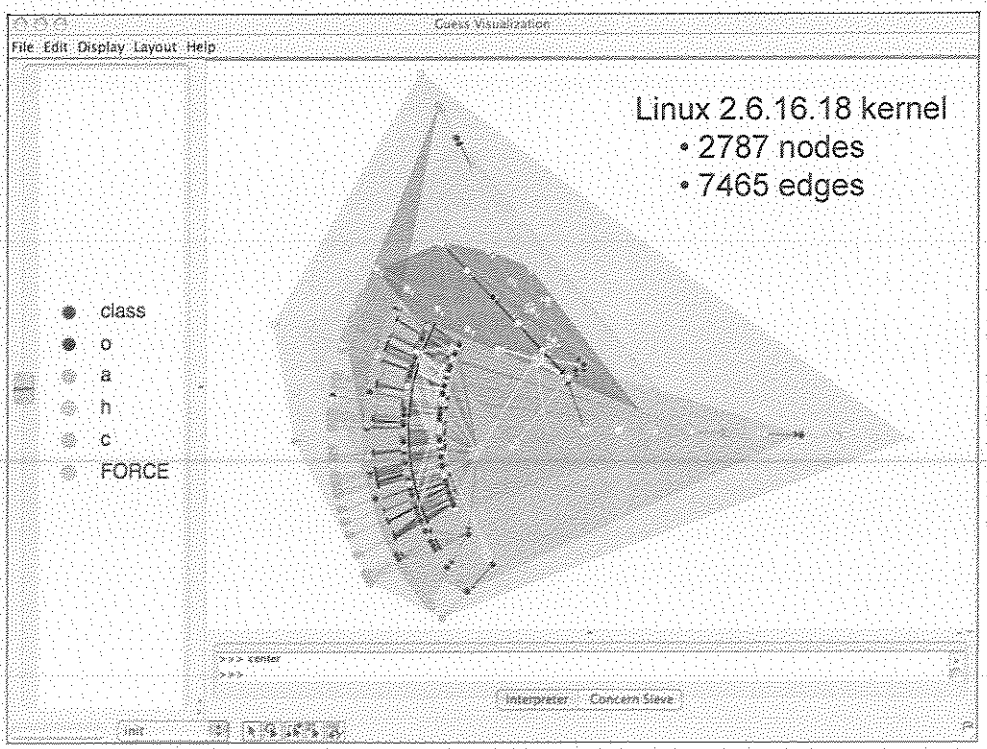


4. MAKAO



Makefile Architecture
Kernel for AO





5. Rule-Based Approach

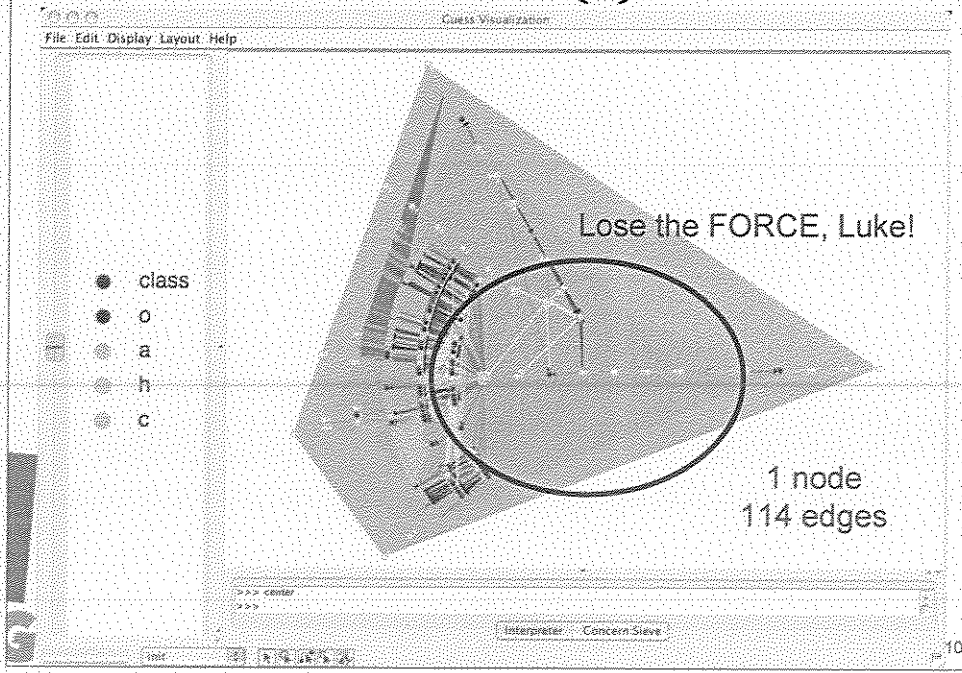
Observations:

- previous slide looks like a mess, even after layouting
- too much detail

Possible solution:

- define rules to modify graph:
  - general vs. application-dependent [Kazman99]
  - semantics-preserving ("cleaning-up") or not
- challenge: don't touch the code ↔ [Bowman99]
- propagate clean-up passes back to build (configuration?) system

6. General Rules (1)

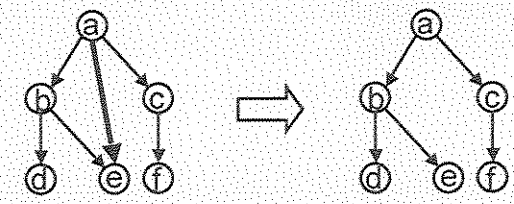


6. General Rules (2)

- semantics-preserving
- faster build
- lose architectural info?

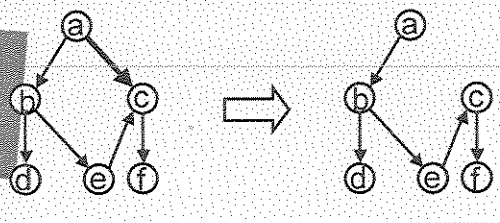
Redundant dependencies:

- simple transitivity



0 nodes
108 edges

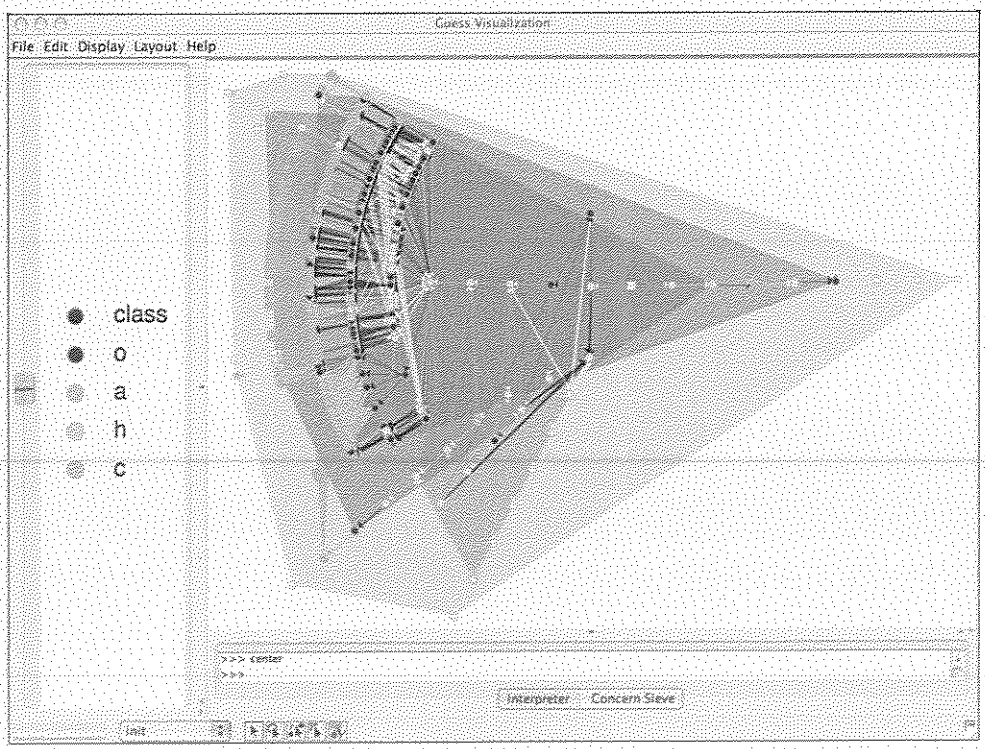
- extended transitivity



not applied

GH-SEI

11

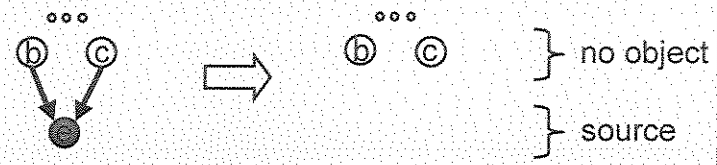


6. General Rules (3)

Redundant dependencies (cont.):

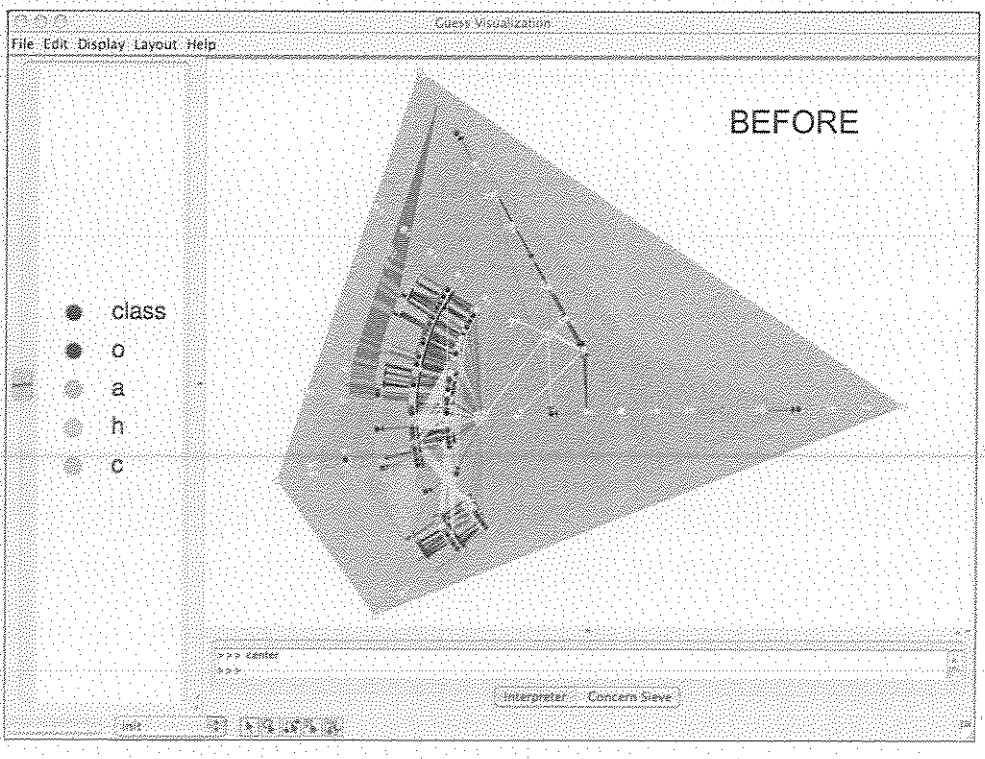
- obsoleteness

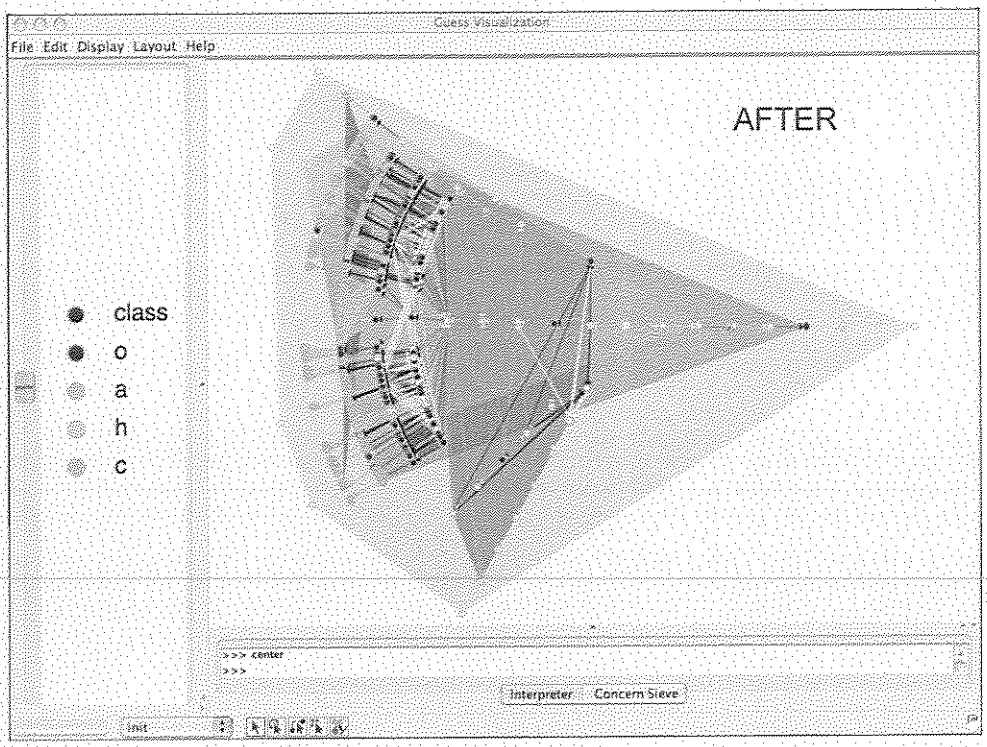
11 nodes
17 edges



☺
- semantics-preserving if no commands tied to source node
- faster build

GH-SEI

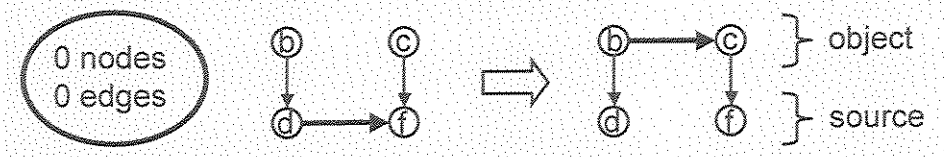




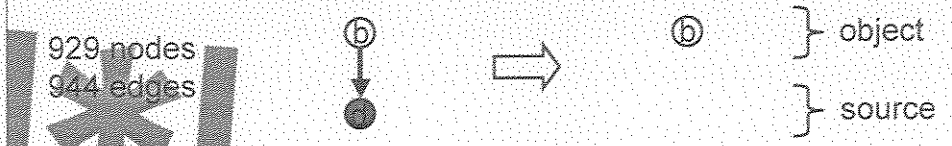
6. General Rules (4)

Raising level of abstraction:

- pulling up source file relations

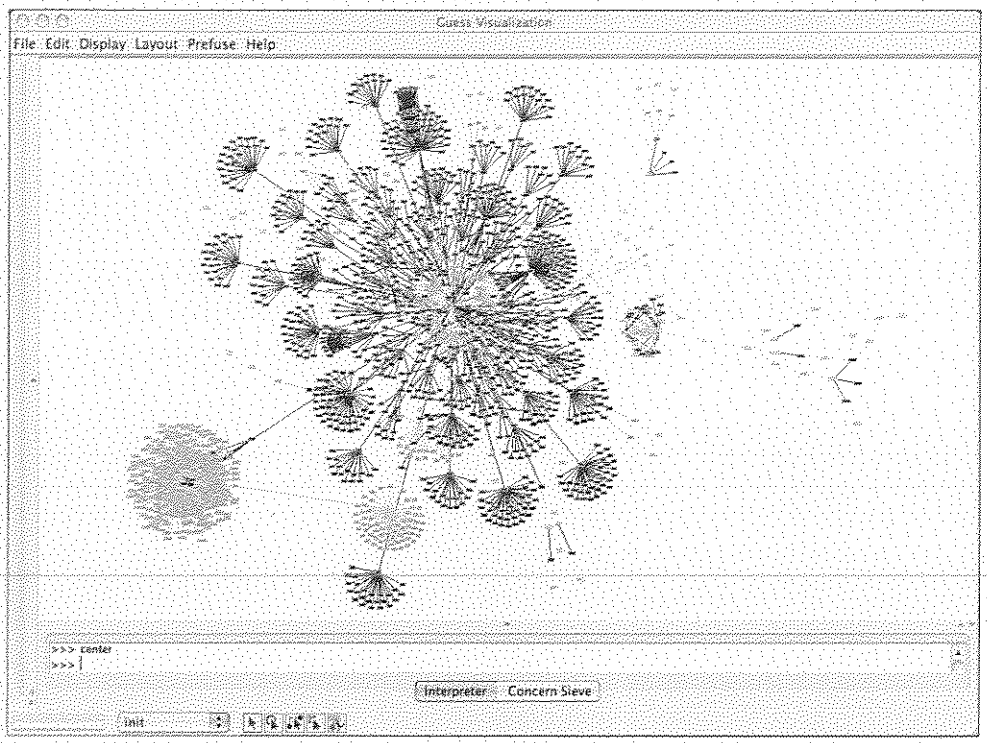


- abstracting away source files



GH-SEI

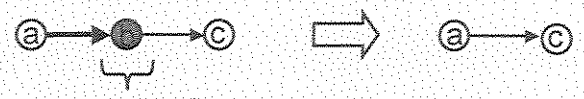
16



6. General Rules (5)

Raising level of abstraction:

- sandwich rule



no regular file

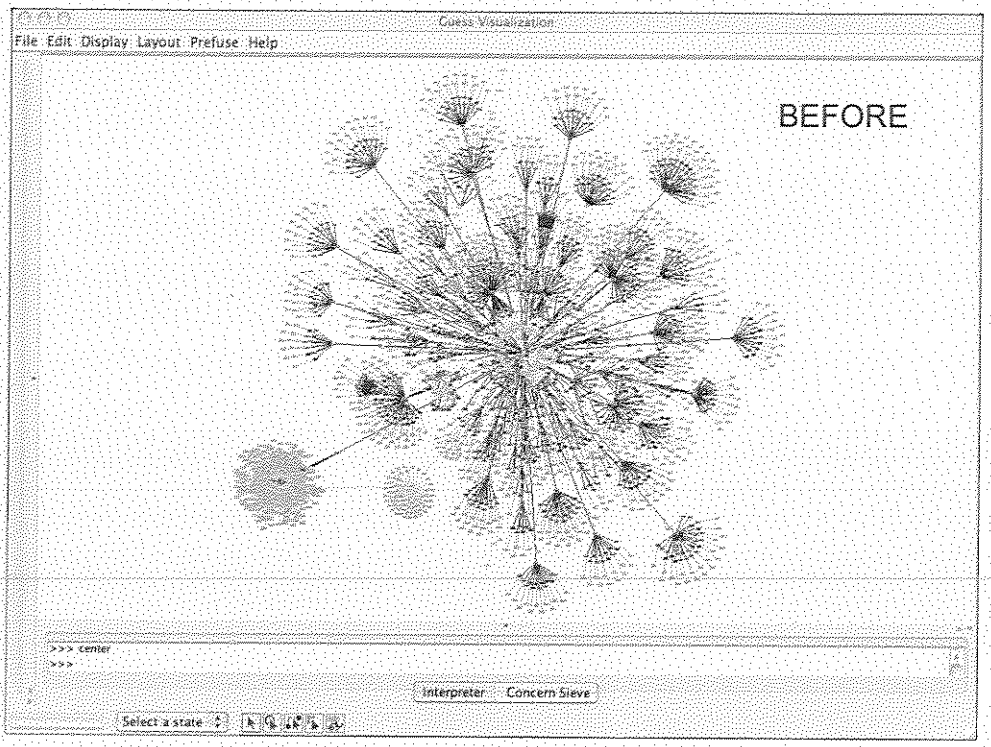
14 nodes
14 edges

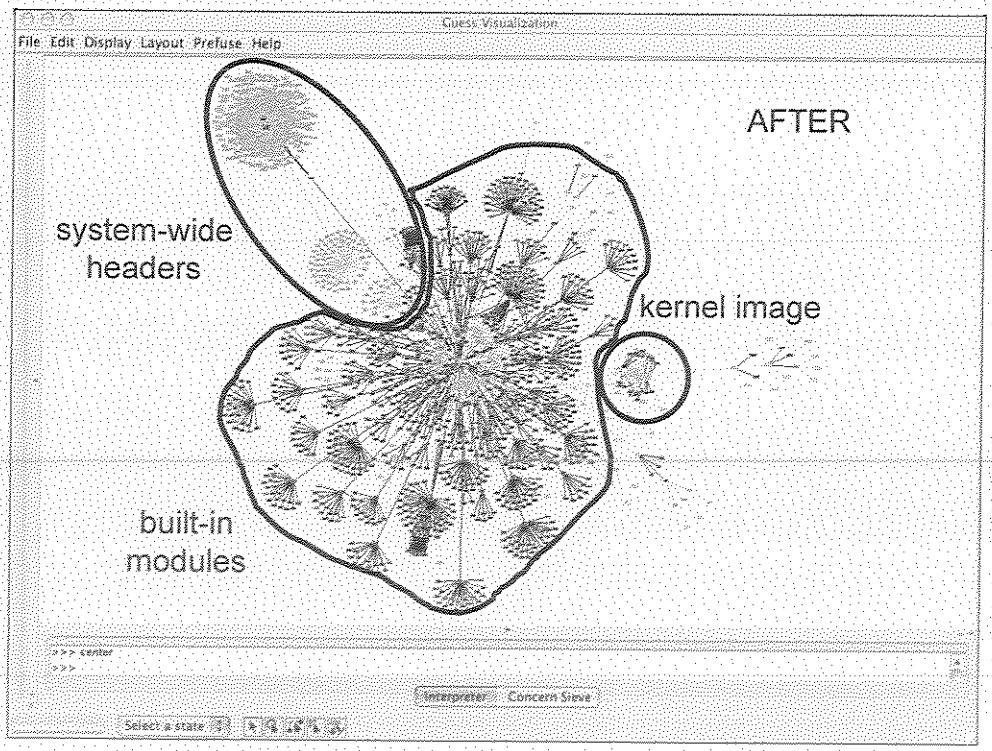
☺ abstraction

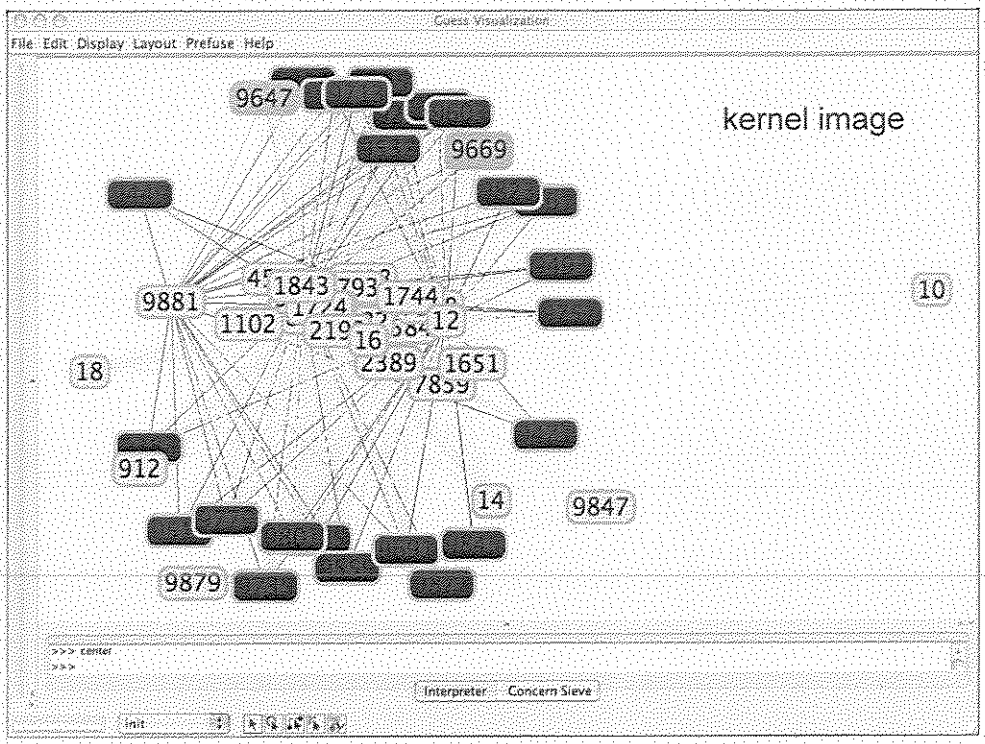
- rules influenced by style of build scripts

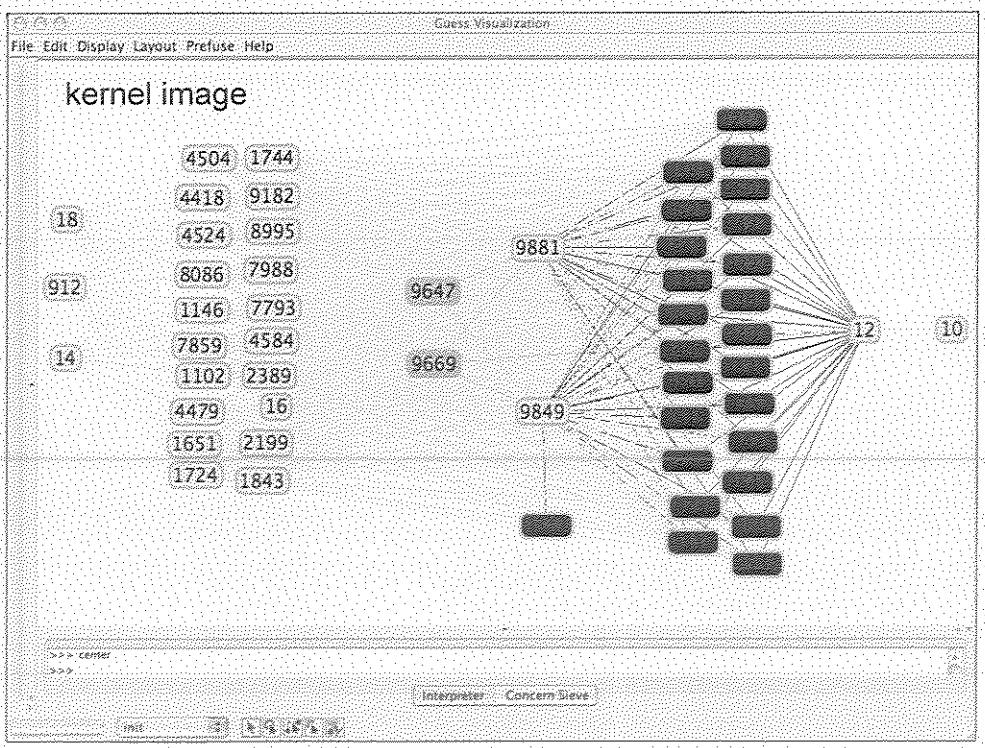
⇒ some build systems have more/less architectural info

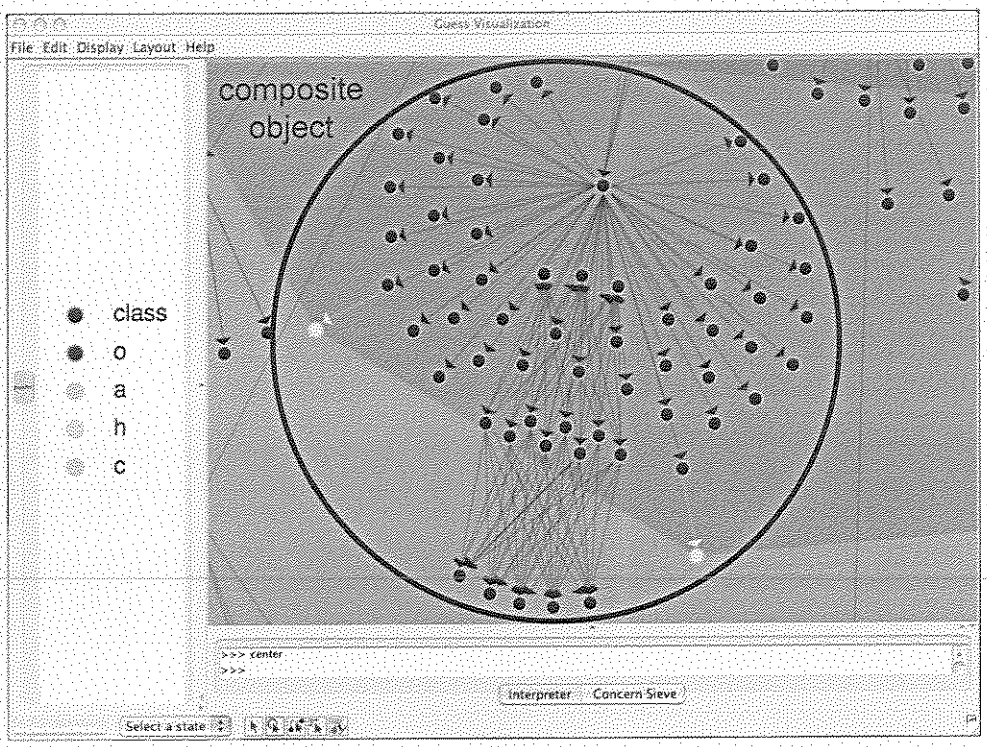
- lose architectural info?


Guess Visualization
File Edit Display Layout Prefuse Help
BEFORE
>>> center
>>>
Interpreter
Concern Sieve
Select a state


Guess Visualization
File Edit Display Layout Prefuse Help
AFTER
system-wide
headers
kernel image
built-in
modules
>>> center
>>>
Interpreter
Concern Sieve
Select a state


Guess Visualization
File Edit Display Layout Prefuse Help
kernel image
9647
9669
9881
1102
1843
1744
12
16
2389
1651
18
912
14
9847
10
9879
>>> center
>>>
Interpreter
Concern Sieve
init

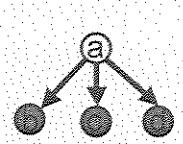

Guess Visualization
File Edit Display Layout Prefuse Help
kernel image
4504
1744
4418
9182
18
4524
8995
8086
7988
912
1146
7793
7859
4584
14
1102
2389
4479
16
1651
2199
1724
1843
9647
9669
9881
9849
12
10
>>> center
>>>
Interpreter
Concern Sieve
init



7. Application-Specific Rules (1)

- composite object files

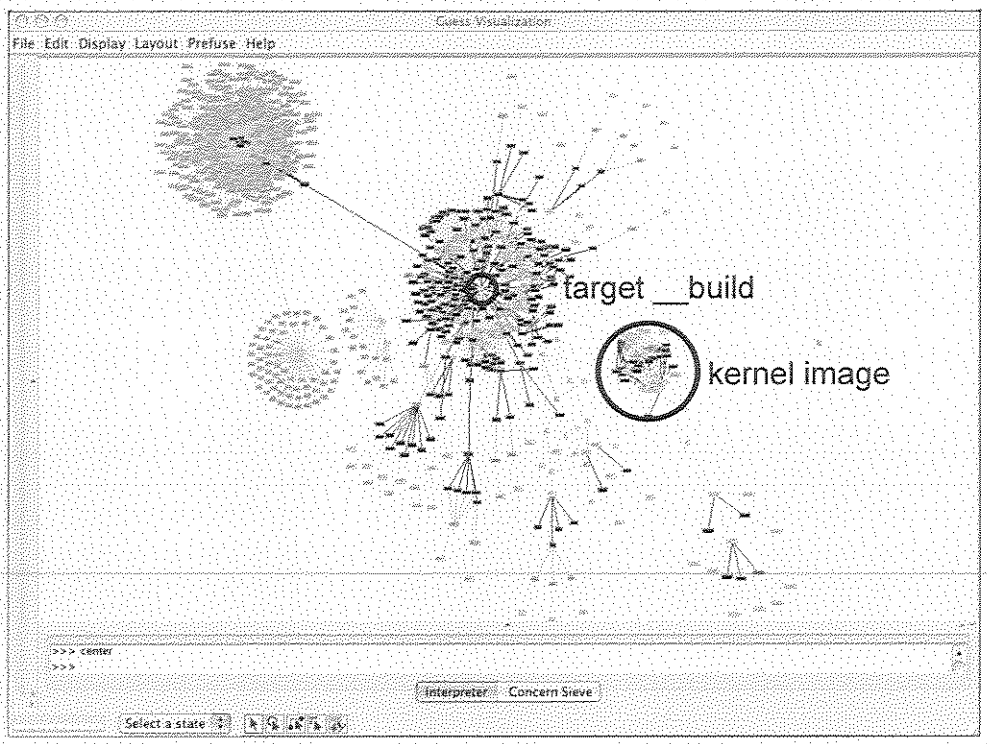
897 nodes
1056 edges

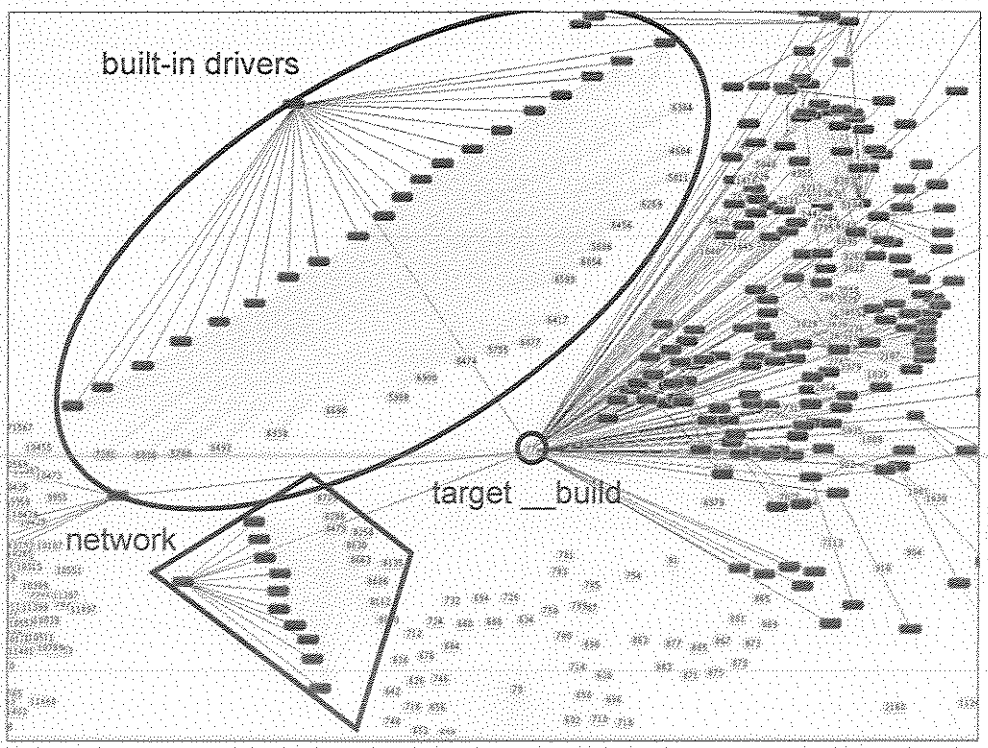


} object

} object

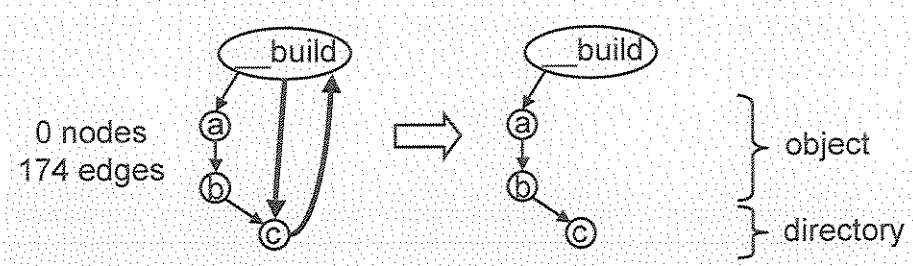
☺ highly effective

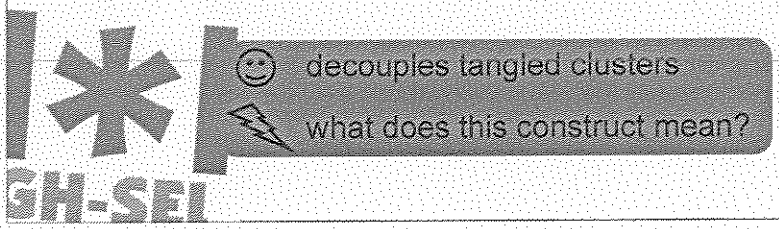

Guess Visualization
File Edit Display Layout Prefuse Help
target __build
kernel image
>>> center
>>>
Interpreter
Concern Sieve
Select a state


built-in drivers
target __build
network

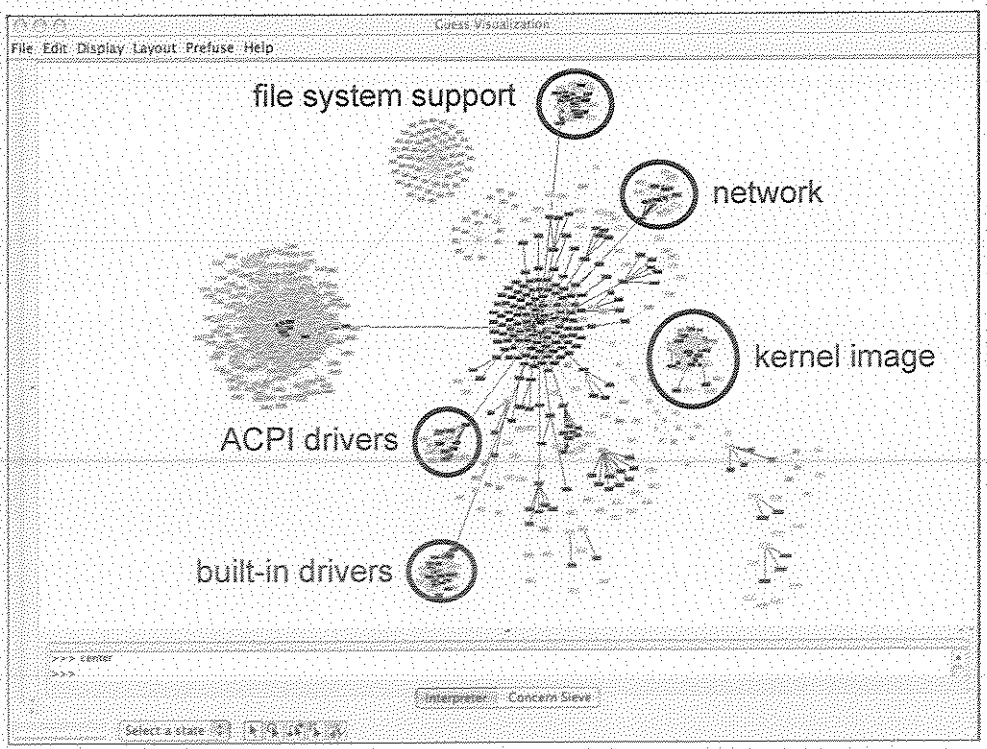
7. Application-Specific Rules (2)

- unchaining redundant cycles





27

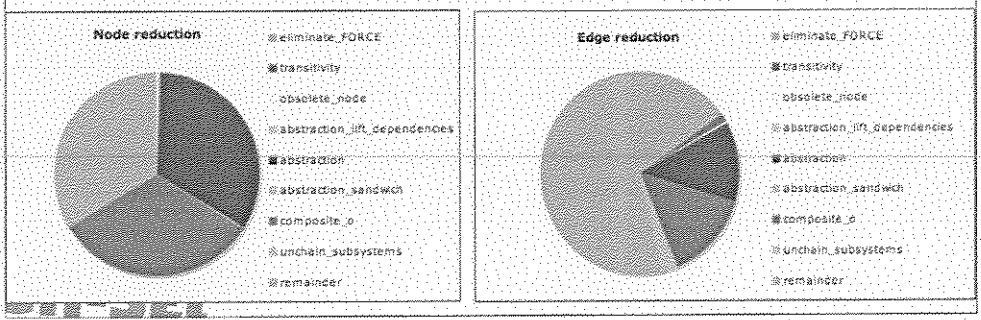


8. Conclusions and Future Work (1)

Conclusions:

- work in progress!
- lots of clean-up and abstraction rules necessary
- build system's knowledge varies per project

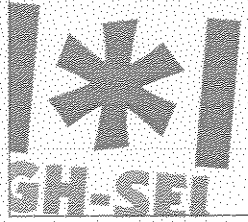
Rules' effectiveness:



8. Conclusions and Future Work (2)

Future work:

- working out dependencies of kernel image
- other cases (GCC, vim, KDE, ...)
- applying clustering techniques
- feed clean-up rules back to build scripts
- come up with new rules
- does order of rules play a role?
- ...



30

References

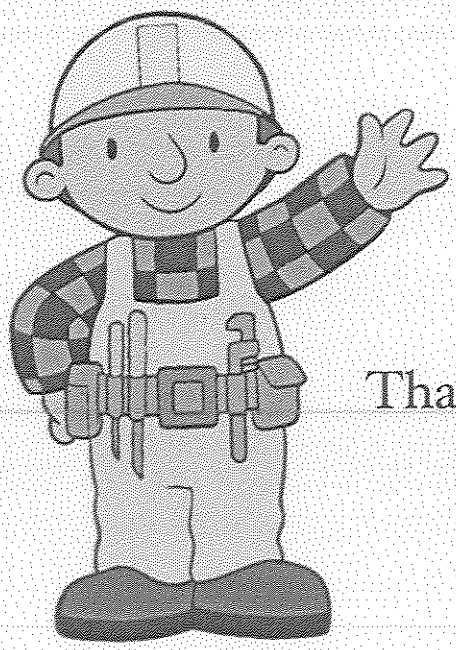
[Biggerstaff89] *Ted J. Biggerstaff. Design Recovery for Maintenance and Reuse. Computer Journal, Vol. 22, No. 7 (p. 36-49), 1989*

[Bowman99] *Ivan T. Bowman, Richard C. Holt and Neil V. Brewster. Linux as a Case Study: Its Extracted Software Architecture. Proc. of ICSE 1999 (p. 555-563)*

[Finnigan97] *P. Finnigan, R. Holt, I. Kalas, S. Kerr, K. Kontogiannis, H. Mueller, J. Mylopoulos, S. Perelgut, M. Stanley, and K. Wong. The Software Bookshelf. IBM Systems Journal, Vol. 36, No. 4 (p. 564-593), November 1997*

[Kazman99] *Rick Kazman and S. Jeromy Carrière: Playing detective. Reconstructing software architecture from available evidence. Proc. of ASE 1999 (p. 107-138)*

[Tu01] *Qiang Tu and Michael W. Godfrey. The Build-Time Software Architecture View. Proc. of ICSM 2001 (p. 398-407)*



Thank you!