Scalable Software Engineering
What is it? Why and How?

S. C. Kothari
Electrical & Computer Engineering Department
Iowa State University
Contact: kothari@iastate.edu, 515-441-4412

Scalable Software Engineering

- SSE = paradigms, techniques, abstractions, and practices that scale to large software.
- End goal: Improve productivity and quality.
- What can we do differently to improve
  - Research ?
  - Education ?
  - Industry Practices ?
- This presentation focuses on education.

What have educators done?

- So far, educators have advocated:
  - A new programming language every few years.
  - Composition techniques: component-based, refactoring, reuse etc.
  - Development practices: extreme, agile, and others.

Score Card

- Business Week – Industry Outlook 2004: software productivity finished dead last, declining from 1998 to 2003.
- Software systems continue to have ever larger error counts.
- Financial risks of taking on a software project are growing.

What may be wrong?

Toy Problems

- Class room software engineering tends to be about toy problems.
- What may appear unnecessary in solving a toy problem may be critical in the evolution of a large system in production.

Misuse of Computing Power

- Software engineering uses little computer power to support design and validation.
- Instead, software engineering uses computer power to compensate for design and implementation flaws.

Dogmas and Fashions

- Case of object-oriented programming:
  - Business week cover story (1991) - Software Made Simple: Unlike AI, OO would have an immediate and practical payoff.
  - John Warnock, CEO of Adobe: The whole object thing is a red herring.
- Problem solving and engineering take the back seats.

Disconnect: Academia & Industry

- Academia is working frantically to produce research papers and to change curriculum every few years.
- Software progress in industry is by dint of nonconformist intellect and massive amounts of money.

Generic Programming

- Class room software engineering tends to be about generic programming skills.
- Real-world problems are about design, validation, maintenance, evolution, and performance – generic programming not the most valuable skill.
- Software is application-driven and the engineering needs vary – no one solution for all.

Serious Problem

- In software engineering, we are reaching the human limits of managing complexity ad hoc.
- One million lines of code = about 8 ft stack of paper.

Teaching scalable software engineering – what is different?

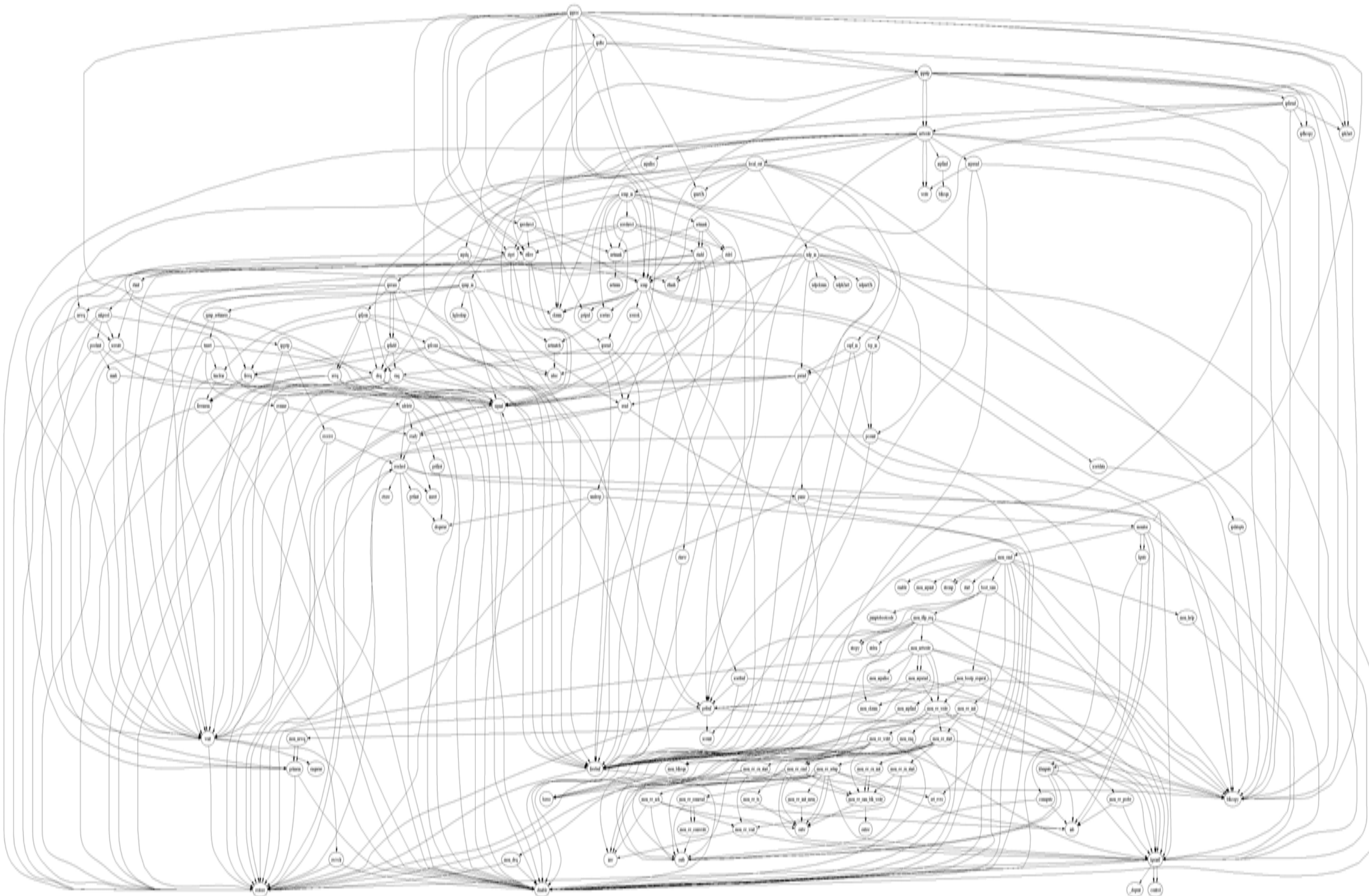
Personal Experiences

- Course projects – Adding new features to the operating system(1988-99): An average student spent 30 hours understanding the code, and 10 hours modifying and debugging the code.
- Research project with150K lines of legacy software (1996-99): Of three years, we spent a couple of month on modifying the code – rest in understanding, debugging, and validating.

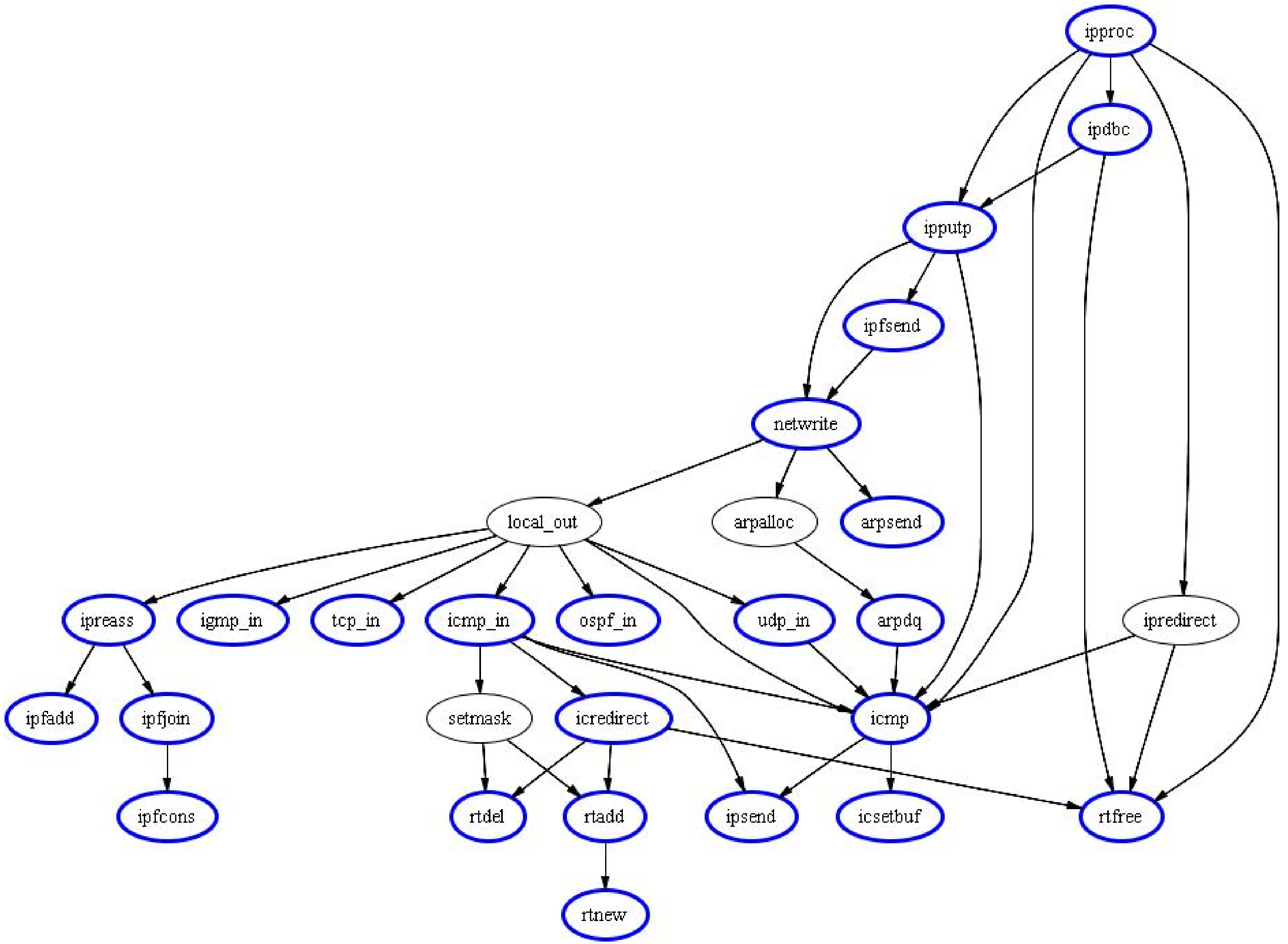
An Example of Complexity

- This example is from an operating system class project.
- The pictures show the dramatic benefit of managing complexity:
  - First picture: the daunting raw complexity
  - Second picture: the dramatic reduction in complexity after analyzing the software
- Manual effort: probably 20 to 30 hours.
- IA Tool: two minutes, assuming the expert applies a good strategy.

RAW COMPLEXITY



DRAMATIC REDUCTION AFTER ANALYSIS



Writing vs. Reading

- Current education practices focus on writing programs.
- Reading (analyzing and understanding) is not taught – we need to change and emphasize program reading.
- Software maintenance costs over the total life of the system dominate software development costs by a factor of 2-10.

Program reading is not easy

- Complex and scattered dependencies between program components.
- Code decay: A study, using a rich data set from the 15-plus year change history for the millions of lines of software in a telephone switching system, has shown statistical evidence of code decay.

Harness the Computing Power

- Powerful tools:
  – Do program mining: on-the-fly extraction of program artifacts and their relationships
  – Analyze complex dependencies in large software
- Tools based on:
  – Powerful query language
  – Graph transformations to manage the complex dependencies

Intelligence Amplifying (IA) Tools

- IA tools combine mechanical processing power of tools with intelligent decision making of human experts.
- Frederick Brooks: “If indeed our objective is to build computer systems that solve very challenging problems, my thesis is that IA > AI, that is, that intelligence amplifying systems can, at any given level of available systems technology, beat AI systems. That is, a machine and a mind can beat a mind-imitating machine working by itself.”

Tool-Assisted Analysis

- It is useful for many software engineering tasks:
  - Efficient debugging
  - Identifying and correcting software defects
  - Efficient testing
  - Architectural analysis and improvements
  - Code audits and safety analysis
  - Impact analysis
  - Automated documentation
  - Generating progress and status reports

Application-specific Software Environments

- Promote application-specific paradigms, graphical environment, and tools for design and analysis of software.
- Examples:
  - Excel for accounting applications
  - Simulink for control applications

Educational Pedagogy

- To summarize what is different:
  - Program reading vs. program writing
  - Real-world software vs. toy problems
  - Engineering design and analysis vs. programming dogmas and fashions
  - Powerful tools vs. labor-intensive manual practices
  - Application-specific paradigms vs. generic programming

Synergistic Activities

- Knowledge-Centric Software Engineering Research at ISU: 9 M.S. and 4 Ph.D. graduates. 26 papers in conferences and journals, 3 patents, 12 software tools, 35 invited talks.
- EnSoft founded in 2002: Engineering services and tools. Customers in USA, Japan, and Europe.
- Close ties with Rockwell: DARPA Project through ISU, tools and services through EnSoft, senior design projects, and distance learning courses.

Scalable Software Engineering Course 2008 Offering: Work

- Reviewing and writing a technical proposal.
- A program reading project with real-world public domain software – tied to query language design and use of graph transformations.
- Read two trend setting SE papers and prepare a presentation.
- Read two techniques or tools papers and prepare presentations.
- Participation in class discussions – writing summary reports and suggesting discussion topics.

THANK YOU

If time permits, we can demonstrate tool-assisted software engineering.

Contact: kothari@iastate.edu