Software Engineering

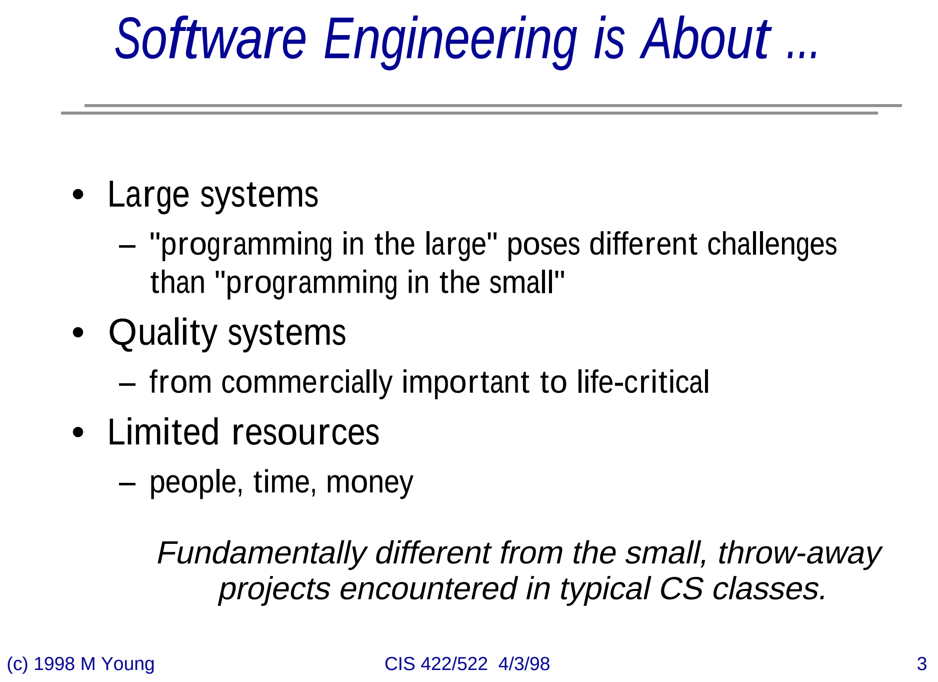
Lecture 1: Overview

History: The "Software Crisis"

Term coined circa 1968

- Cheaper, more powerful machines => more demands on software
- Methods for developing small systems did not scale up
- Many large systems were failing, or late
- Software costs beginning to dominate
  - 1960: 80/20 division of hardware/software costs
  - 1970: 50/50 division
  - 1983: 20/80 division of cost; software dominates

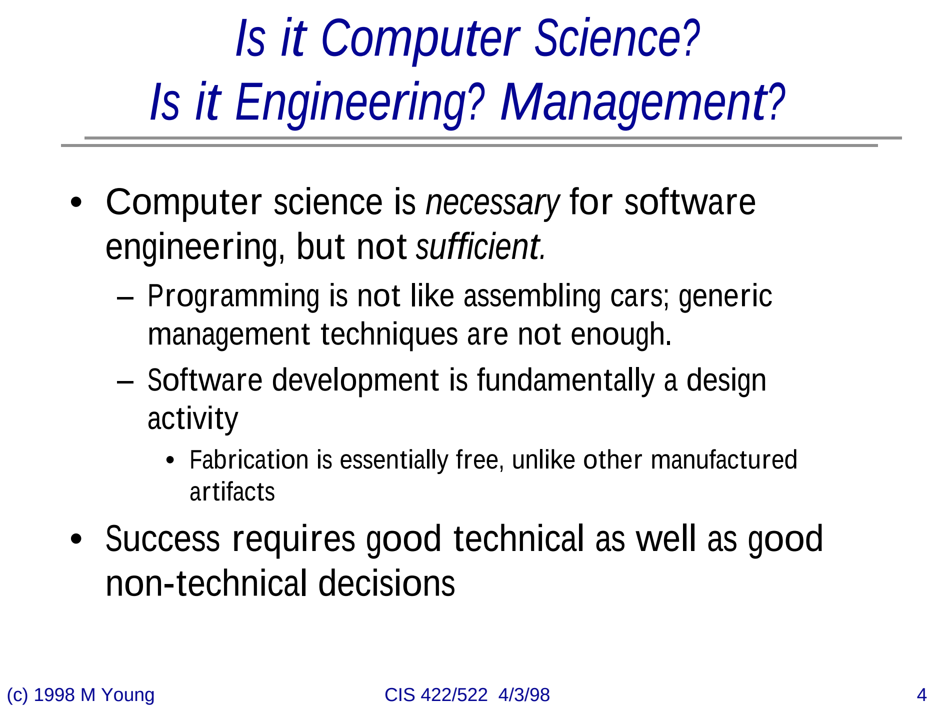
© 1996-1998 Michal Young; portions © 1998 Mauro Pezzè
For reprint permissions contact michal@cs.uoregon.edu



Software Engineering is About ...

- Large systems
  - "programming in the large" poses different challenges than "programming in the small"
- Quality systems
  - from commercially important to life-critical
- Limited resources
  - people, time, money

Fundamentally different from the small, throw-away projects encountered in typical CS classes.



Is it Computer Science?
Is it Engineering? Management?

- Computer science is *necessary* for software engineering, but not *sufficient.*
  - Programming is not like assembling cars; generic management techniques are not enough.
  - Software development is fundamentally a design activity
    - Fabrication is essentially free, unlike other manufactured artifacts
- Success requires good technical as well as good non-technical decisions

© 1996-1998 Michal Young; portions © 1998 Mauro Pezzè
For reprint permissions contact michal@cs.uoregon.edu

Programming in the Large

"Multi-person development of multi-version software" (Parnas 78)

- Team development
  - The most crucial design decisions involve communication among people
- Months or years of development
  - Plan must consider milestones and not just endpoint
- Years of operation and evolution
  - Programs last longer than programmers!

Software Development by Teams

Product Structure

- Software product structure is partly motivated by problems of cooperating in software development
- Product structure: Modularity
  - Divide design, coding, and testing into pieces for individual programmers and subsystem teams
  - Minimize and control communication among team members (module interfaces are *human* interfaces)
  - Provide small granularity for process visibility

© 1996-1998 Michal Young; portions © 1998 Mauro Pezzè
For reprint permissions contact michal@cs.uoregon.edu

Software Development by Teams

Process Structure

- Software development methods and processes are largely motivated by team development
- Process structure:
  - Phases and milestones: For schedule, planning, and management of team development
  - Key documents facilitate shared understanding and agreement, and a record of decisions and rationale
  - Coordination with well-defined roles and responsibilities facilitates effective teamwork

Quality systems Reliable, robust, safe, ...

- Because of flexibility and relative cost, software is replacing hardware in critical applicatiions
  - Clever programming isn't enough to keep a B777 or A320 in the air, to keep the phones working, or to ensure proper X-ray doses
- "Correct" is not enough!
  - Design flaws and poor requirements are more expensive and potentially as dangerous as program "bugs"
  - Quality must be maintained at every stage of development

© 1996-1998 Michal Young; portions © 1998 Mauro Pezzè
For reprint permissions contact michal@cs.uoregon.edu

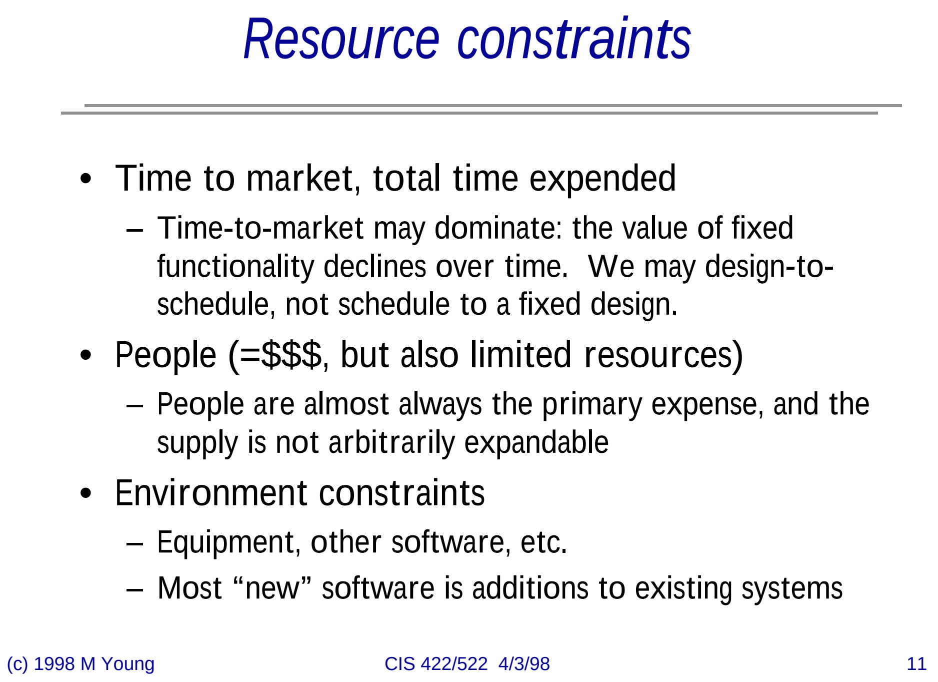
Fast, easy to use, powerful Attractive to clients and buyers

- *Fast: Selective* application of computer science
  - The essential tools are algorithms and data structures, but their skillful application is a matter of engineering
  - Complexity is an additional cost, in development time, reliability, and maintainability, so we must be selective
- *Easy to use:* Human factors and software design
  - The essential background is psychology, but we must draw also on programming and design
- *Powerful:* General, simple, orthogonal, *beautiful*

Maintainable and Reusable

- Fast to market, inexpensive
  - Most software delivery is revisions of existing systems; schedule depends on *change cycle*
  - Expense depends on scope of change
- Borrowing existing software is cheaper than writing new
  - Reusable components are an asset
  - Advantage is schedule and quality, as well as cost
  - Reuse not only of code, but also design, test suites, user manuals, ...

© 1996-1998 Michal Young; portions © 1998 Mauro Pezzè
For reprint permissions contact michal@cs.uoregon.edu



Resource constraints

- Time to market, total time expended
  - Time-to-market may dominate: the value of fixed functionality declines over time. We may design-to-schedule, not schedule to a fixed design.
- People (=$$$, but also limited resources)
  - People are almost always the primary expense, and the supply is not arbitrarily expandable
- Environment constraints
  - Equipment, other software, etc.
  - Most “new” software is additions to existing systems

© 1996-1998 Michal Young; portions © 1998 Mauro Pezzè
For reprint permissions contact michal@cs.uoregon.edu