Software Engineering

- name coined at the NATO Science Committee Conference, October 1968
- Engineering-- established, scientifically sound practices that well-trained practitioners follow
- Software Engineering-- the application of scientific knowledge to the the development and maintenance of software systems
- Software-- ALL associated artifacts to assist with the development, operation, validation, and maintenance of programs/software systems
  - e.g., code, documentation, designs, requirements, user manuals, installation manuals, test cases, test results, trouble reports, revision history, installation scripts,...

Another definition

- Ghezzi: A field of computer science that deals with the building of software systems that:
  - are so large & complex to require teams of developers
  - exist in multiple versions
  - used for many years
  - undergo changes/evolution

Why engineer software?

- Impact on Society
- Economics
- Quality Concerns

Why engineer software? Impact on Society

- If you fly, your life depends on software
  - Airbus
- Your bank account depends on software
  - New York bank reconciliation failure
- Medical devices are controlled by software
  - Therac-25
- … and so on

Why engineer software? Economics

- An important industry
  - Software is an important industry
  - Worldwide competition
  - Global development models
- Significant installed software base
  - Well-designed software easier to maintain
  - Poorly designed legacy s/w may be a hinderance

Why engineer software? Quality

- Analogy to the automobile
  - U.S. automobile industry use to be very complacent about quality
    - Lost a significant amount of market share
    - Will complacency about s/w quality lead to the same result?
  - There are many recalls for automobiles
    - Some fixed for free
  - There are many defects in software
    - Some "fixed" for free
    - Some fixed in the "the next" release
      - With the customer paying for the upgrade

Quality Issues

- Software is now an integral part of every facet of our societal infrastructure
  - Transportation
  - Communication
  - Financial
  - Poor quality software menaces the maintenance of that infrastructure
- Software is the "Grand Enabler" holding the key to scientific and engineering challenges
  - Human genome project
  - Space exploration
  - Weather prediction

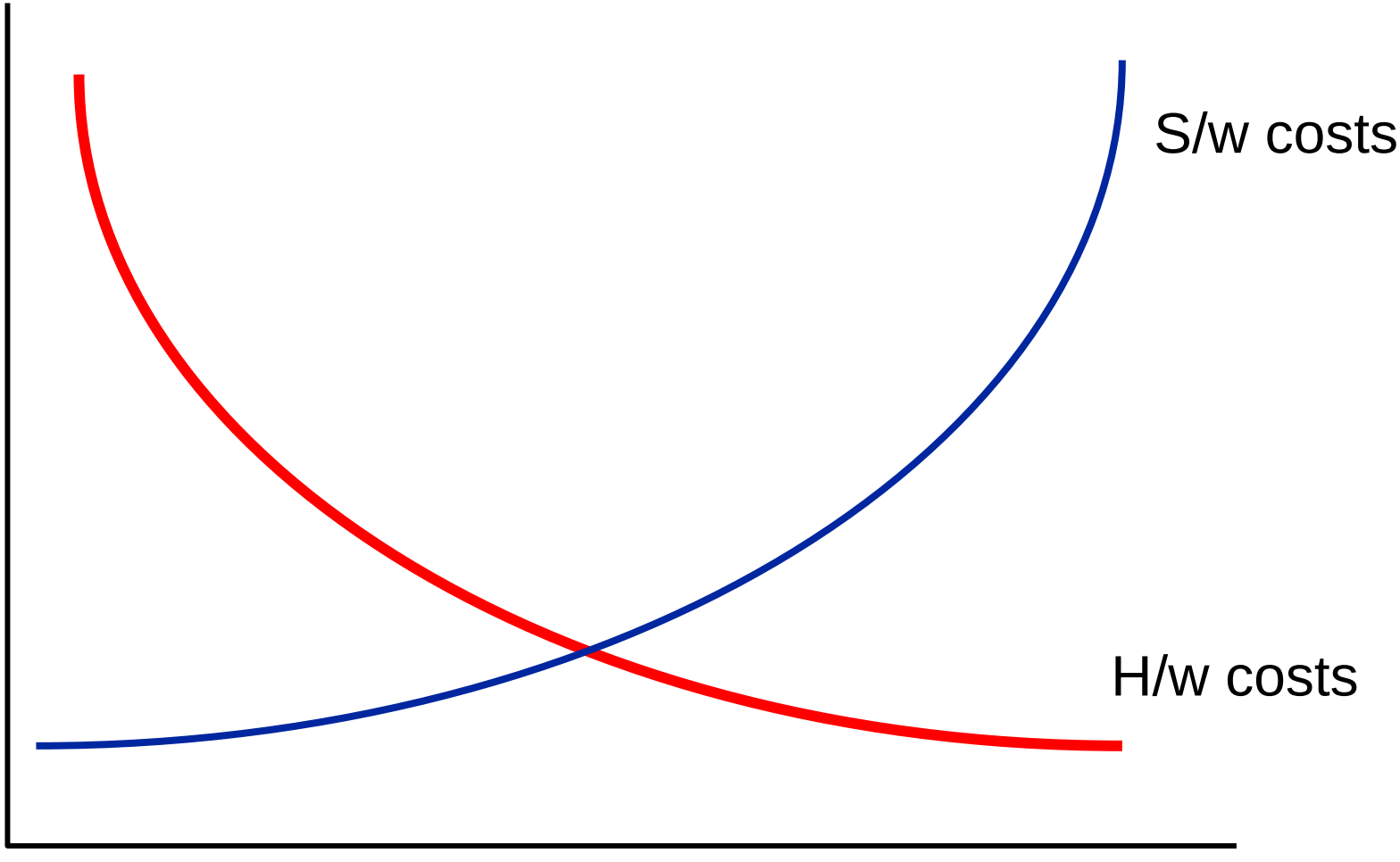
The nature of software

- Software is a complex, intricately interconnected data aggregate
- Software Development is the process of creating such a complex product, while continuously assuring that it remains consistent
- Software Engineering combines some of the approaches of classical engineering with some of the abstract approaches of mathematics

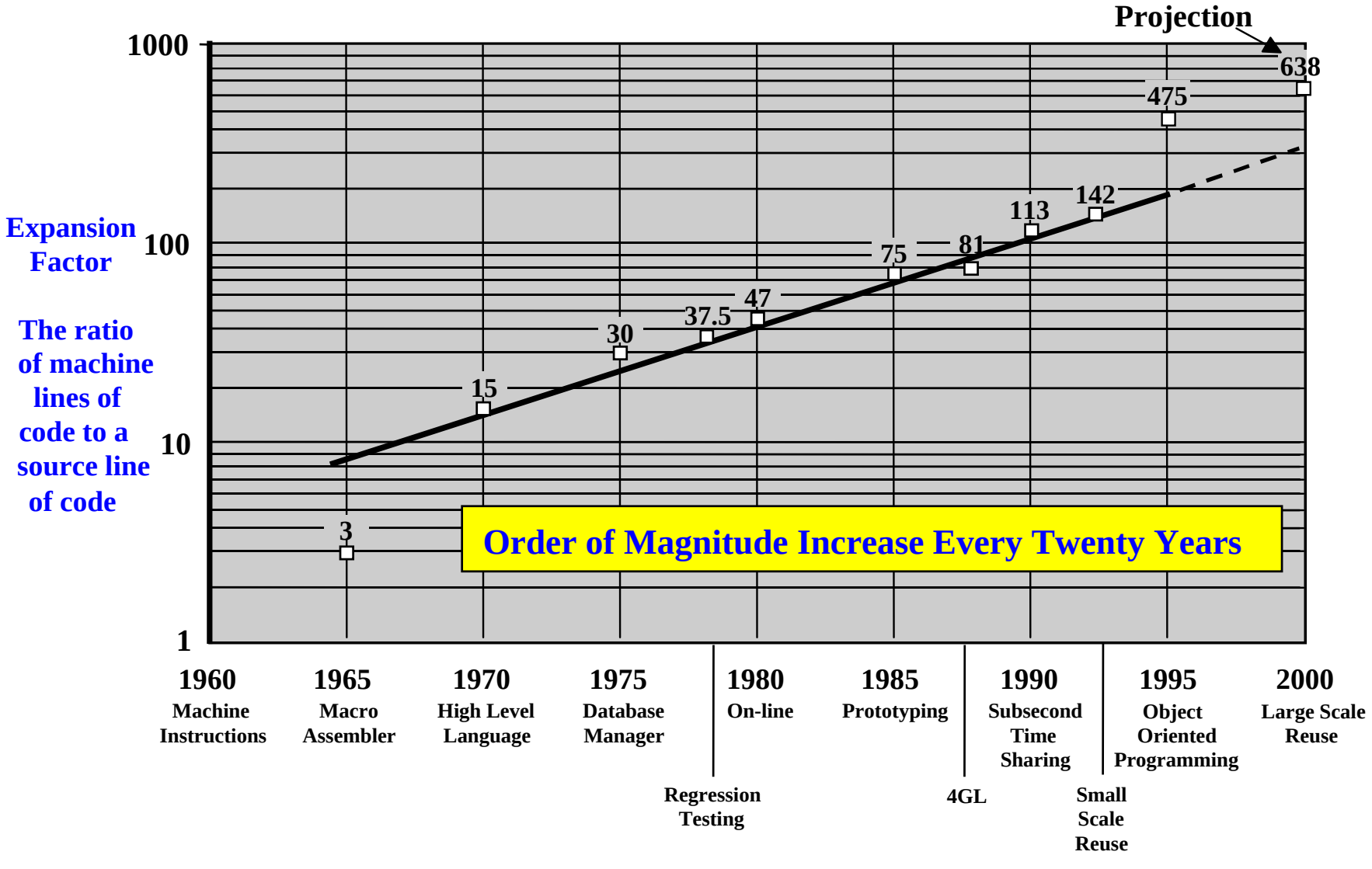
Hardware versus Software

- Percentage wise, hardware costs are decreasing and software costs are increasing
- Is hardware development done better than software development?
  - Yes, but...
    - s/w systems tend to be more complex
    - tend to do new applications in s/w and well-understood applications in h/w
    - despite the use of more rigorous and systematic processes, hardware systems fail too

Hardware / Software Cost trends for projects



Trends in Software Expansion (Bernstein, 1997)



What is novel about software?

- product is unprecedentedly complex
- application horizons expand very fast--with human demands/imagination
- construction is human-intensive
- solutions require unusual rigor
- extremely malleable--can modify the product all too easily

How to improve Software Quality

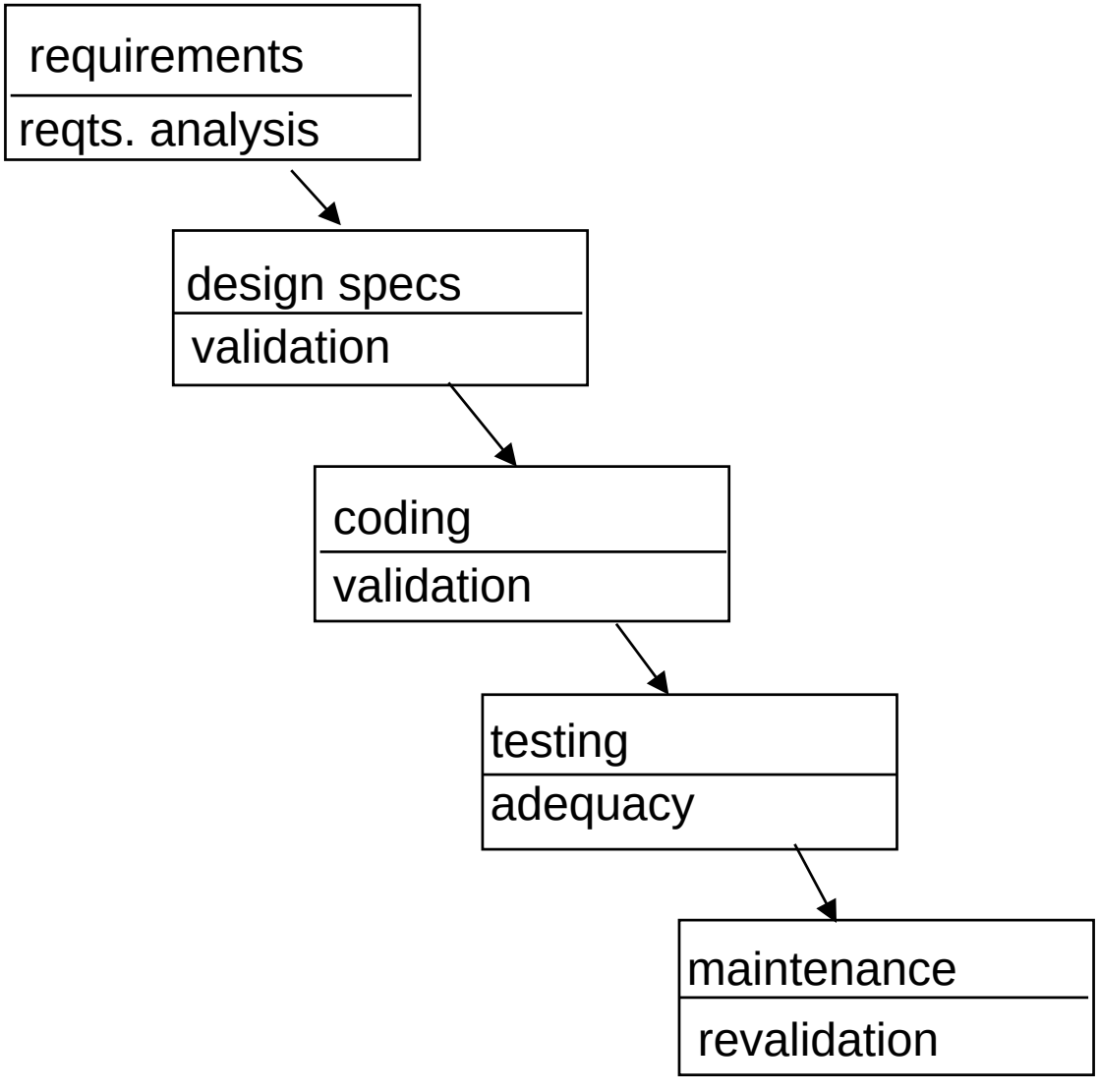
- Treat software as a PRODUCT produced in a systematic way according to a well-defined PROCESS designed to achieve explicit quality objectives
  - Build quality in
  - Define software product
  - Reason about the product
  - Incorporate validation as integral steps in the process

But what is that process?

- What methods should be used?
- What tools support those methods?
- How do we know that these methods and tools will lead to a better product?

Based on experimentation, build up a sense of what are the best practices

Software Lifecycle



Waterfall Model

- requirements-- a complete,consistent specification of what is needed
  - provides visibility for customers, developers, and managers
  - benchmark for testing and acceptance
  - reduces misunderstandings
- requirements analysis
  - evaluate completeness and consistency
  - evaluate needs and constraints
  - evaluate feasibility and costs
    - development and maintenance costs
    - probability of success

Waterfall Model (continued)

- design specifications--a description of how the requirements are to be realized
  - high-level architectural design
  - low-level detailed design
- design validation
  - traceability between requirements and design decisions
  - internal consistency

Waterfall Model (continued)

- code--realization of the design in executable instructions
- code validation
  - assure coding and documentation standards have been maintained
  - internal consistency
    - e.g., syntactic analysis, semantic analysis, type checking, interface consistency
  - consistency between design/requirements and code

Waterfall Model (continued)

- testing--reveal problems, demonstrate behavior, assess reliability, evaluate non-functional requirements (e.g., performance, ease of use)
  - unit testing
  - integration testing
  - system testing
  - acceptance testing
  - regression testing
- testing validation
  - adequacy of the testcases

Waterfall Model (continued)

- maintenance--the process of modifying existing software while leaving its primary functionality intact
  - corrective maintenance-- fix problems (20%)
  - adaptive maintenance-- add new functionality/enhance existing features (30%)
  - perfective maintenance-- improve product (50%)
    - e.g., performance, maintainability
- 3 primary steps
  - understand existing software
  - change existing software
  - revalidate existing software
- maintenance involves all the previous phases of the lifecycle

Is the waterfall model an appropriate process model?

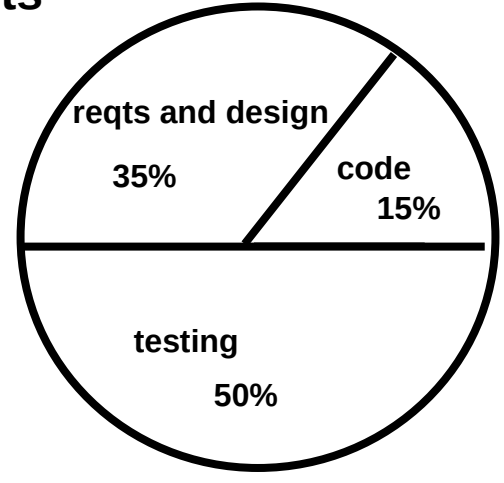
- recognizes distinct activities
- clearly oversimplifies the process
  - wait, wait , wait, surprise model
- actual processes are more complex
  - numerous iterations among phases
  - not purely top down
  - decomposition into subsystems
- many variations of the waterfall model
  - prototyping
  - re-engineering
  - risk reduction
  - ...

Software costs

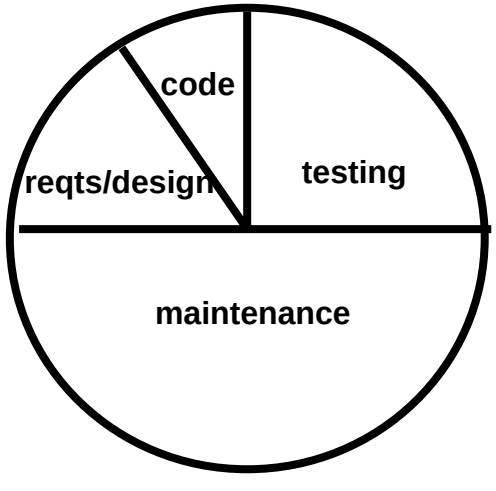
- Development costs
  - generally measured in hundreds to thousands of dollars per delivered LOC
    - many artifacts associated with a line of code
  - testing and analysis is usually 50% of this cost
- Maintenance costs
  - 2-3 times as much as development

Software Costs

Development costs



Full lifecycle costs



Decisions made throughout a project affect the cost of maintenance

- planning for maintenance increases front end cost
- Industry is often unwilling to pay these costs up-front
  - Time to market
  - Job turn-over

Some interesting numbers

- About 25% of s/w projects fail
  - Failure rate increases as the size of the project increases
- Costs about $100/LOC
  - Ranges between $10-$600
- Typical programmer produces about 30 LOCs a day
  - Ranges between 10-100 LOCs
- Ranges between 3-10 faults/KLOC

Barriers to engineering software

- industry's short term focus
- shortage of skilled personnel
- inadequate investment in R&D
  - PITAC (Kennedy-Joy) Report
  - US SW GNP is ~$228B but less than 1% spent on R&D
- Poor technology transfer models
  - "toss over the fence"
- Lack of "good" standards
  - Lack of experimental basis for standards

Standards

- IEEE Standards Group
  - P 730.2, Guide for Software Quality Assurance Programming (6/89)
  - R 982.1, Std Dict. of Measures to Produce Reliable Software (3/88)
  - R 982.2, Guide: Use of Std Meas. to Produce Reliable Software (3/88)
  - R 1002, Software Engineering Taxonomy (9/92)
  - ...
  - R 1012, Software Verification Plans (3/92)
  - P 1420.2, Software Reuse - Data Model for Reuse Library Interop.: Basic Data Model (12/94)
  - P 1430, Software Reuse - Concept of Operations for Interoperating Reuse Libraries (6/95)
  - * P 1498, Software Life Cycle Processes: Acquirer-Supplier Agreement
- NIST FIPS
  - FIPSPUB99 GUIDELINE:A FRAMEWORK FOR THE EVALUATION AND COMPARISON OF SOFTWARE DEVELOPMENT TOOLS, 1983 March 31.
  - FIPSPUB106 GUIDELINE ON SOFTWARE MAINTENANCE, 1984 June 15.
  - FIPSPUB101 GUIDELINE FOR LIFECYCLE VALIDATION, VERIFICATION, AND TESTING OF COMPUTER SOFTWARE, 1983 June 6.
  - FIPSPUB132 GUIDELINE FOR SOFTWARE VERIFICATION AND VALIDATION PLANS, 1987 November 19.
- DoD
  - MIL-STD-498 -harmonizes the predecessor standards DOD-STD-2167A and DOD-STD-7935A
  - IEEE-STD-1498 and J-STD-016-1995 a commercial version of MIL-STD-498 [1] published in January 1996

President's Information Technology Advisory Committee (PITAC) report, February 1999

- 4 priority areas
- Software;"The demand for software has grown far faster than our ability to produce it. Furthermore, the Nation needs software that is far more usable, reliable, and powerful than what is being produced today. We have become dangerously dependent on large software systems whose behavior is not well understood and which fail in unpredicted ways."
- Other areas: scaleable infrastructure, high-end computing, socioeconomic impacts

High-level Goals of Software Engineering

- improve productivity
  - reduce resources
    e.g., time, cost, personnel
- improve predictability
- improve maintainability
- improve quality

What do we need?

- Scientific basis for exploration and evaluation
- Organized discipline
- Trained professionals
- Technology transfer strategies
- Quality control

- Model for s/w engineering
  - Based on accumulated experimental evaluations, recommended best practices

Some Contributions of SE

60's — recognized software problem
- waterfall model

70's — programming-in-the-small
- structured programming
- step-wise refinement
- modularization

80's — programming-in-the-large
- data abstraction
- object-oriented development

90's — programming-in-the-many
- heterogeneity
- distributed computing models
- open architectures