1 History of software engineering

- Software is everywhere
  - buying bread, driving car, washing clothes
  - synonyms: *programs*, *applications*
- People, who develop the software
  - software engineers, software developers, programmers
  - they possess skills and tools that allow them to develop and evolve software

Difficulties that programmers face

- Accidental difficulties
  - difficulties of current/ past/ future technologies
  - quirks of the operating systems, compilers, languages, processes
  - they come and go
- Essential difficulties of software
  - subset of essential (defining) properties
  - no easy answers to essential difficulties !!!

Essential difficulties

- Invisibility
  - senses cannot be easily used in comprehension
  - visualizations, sonifications, require lots of work
- Complexity
  - programs are among most complex systems ever created
  - our short-term memory accommodates only ~7 things
- Changeability
  - software is constantly changing
  - yesterday's comprehension may be obsolete

Essential difficulties, cont.

- Conformity
  - Large system (hardware, users, domain)
  - The program glues it all together, reflects the large system
  - This adds even more complexity
- Discontinuity
  - People easily understand linear or semi-linear systems: shower
  - Software is discontinuous, small change of input can result in huge change of output: password

Software engineering

- Set of recommendations how to develop software
  - “software” is a result of “software engineering”
- A discipline with a considerable body of knowledge and considerable importance in both academia and industry

Beginning of software

- Software separated from the hardware in 1950's
  - emerged as a distinct technology
  - became independent product
- Original programmers recruited from the ranks of hardware engineers and mathematicians
  - used ad-hoc techniques from their former fields

Paradigm

- Thomas S. Kuhn
  - "The Structure of Scientific Revolutions"
- Paradigm
  - "Coherent tradition of scientific research"
    - includes law, theory, application, instrumentation, terminology, research agenda, textbooks, norms, curricula, culture of the field…
    - not only the ideas, but also investment
  - currently overused
    - "object oriented paradigm", etc.

Anomaly

- “Anomaly is an important fact that directly contradicts the old paradigm”

- Dilemma: disregard anomaly vs. paradigm shift
  - to shift paradigm means to abandon large part of the investment
  - the anomaly must be compelling

Paradigm shift

- Discontinuity in the development of the discipline (revolution)
  - Kuhn collected extensive historical data on paradigm shift
  - phlogiston - > oxygen in 1770's
    - Lavoisier

Resistance to paradigm shift

- Advantages of the new paradigm is in dispute
  - attempts are made to extend old paradigm to accommodate anomalies
  - band-aid approaches try to fix old paradigm
- Knowledge and investment accumulated up to that point may lose its significance
  - some knowledge may be completely lost (knowledge of color of the chemical compounds)
- Final victory of the new paradigm guaranteed by a generation change
- Unsuccessful attempts at paradigm shift

Paradigm shift of ~ 1970

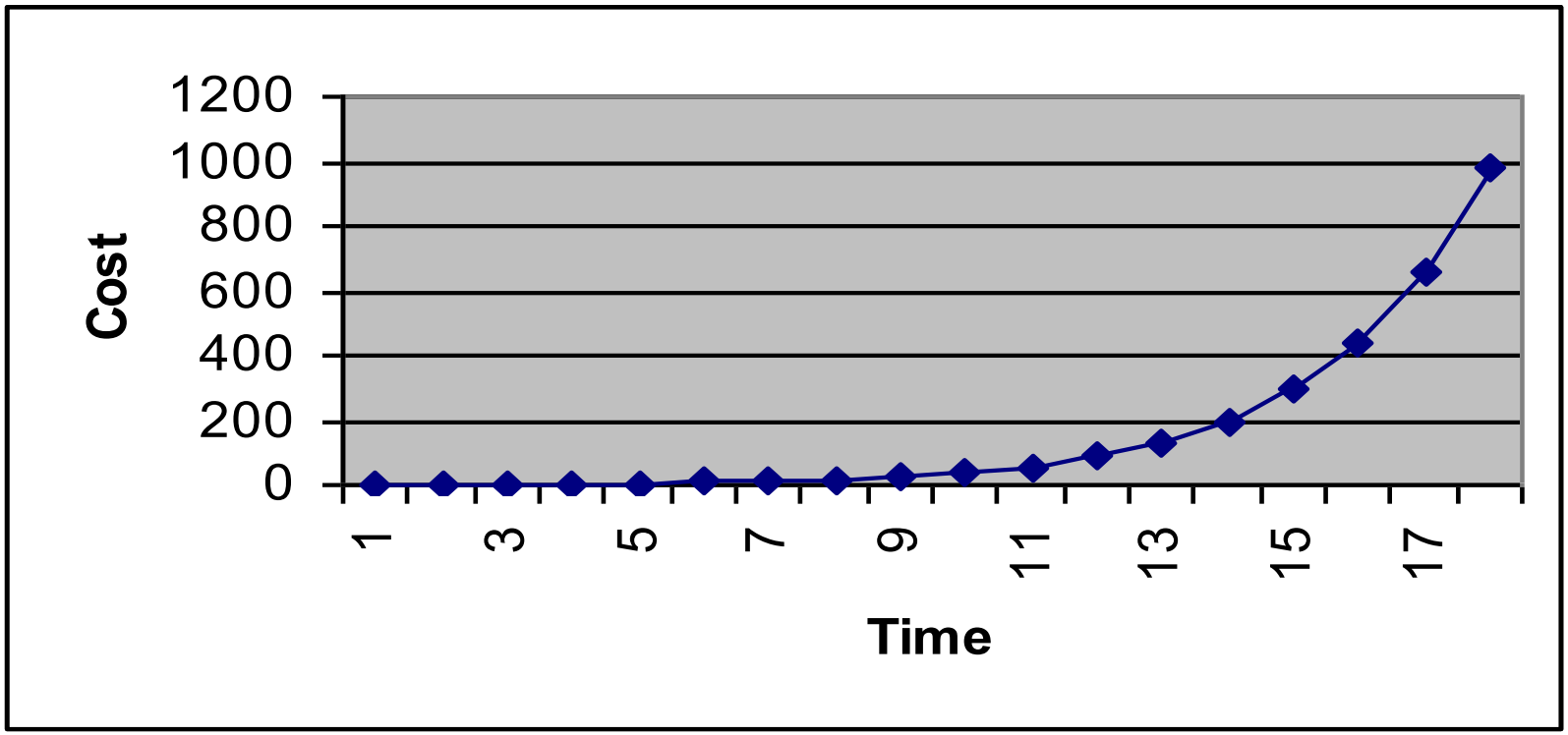
- Anomaly
  - Previous techniques did not scale up
  - Brooks: “Mythical Man-Month”
  - demands of the new operating system OS/360 taxed the limits of the programmers, project managers, and the resources of the IBM corporation
- Paradigm shift established discipline of software engineering
  - dealt with complexity of software
  - software design established as an important consideration
  - introduced the waterfall metaphor

Waterfall metaphor (linear process)

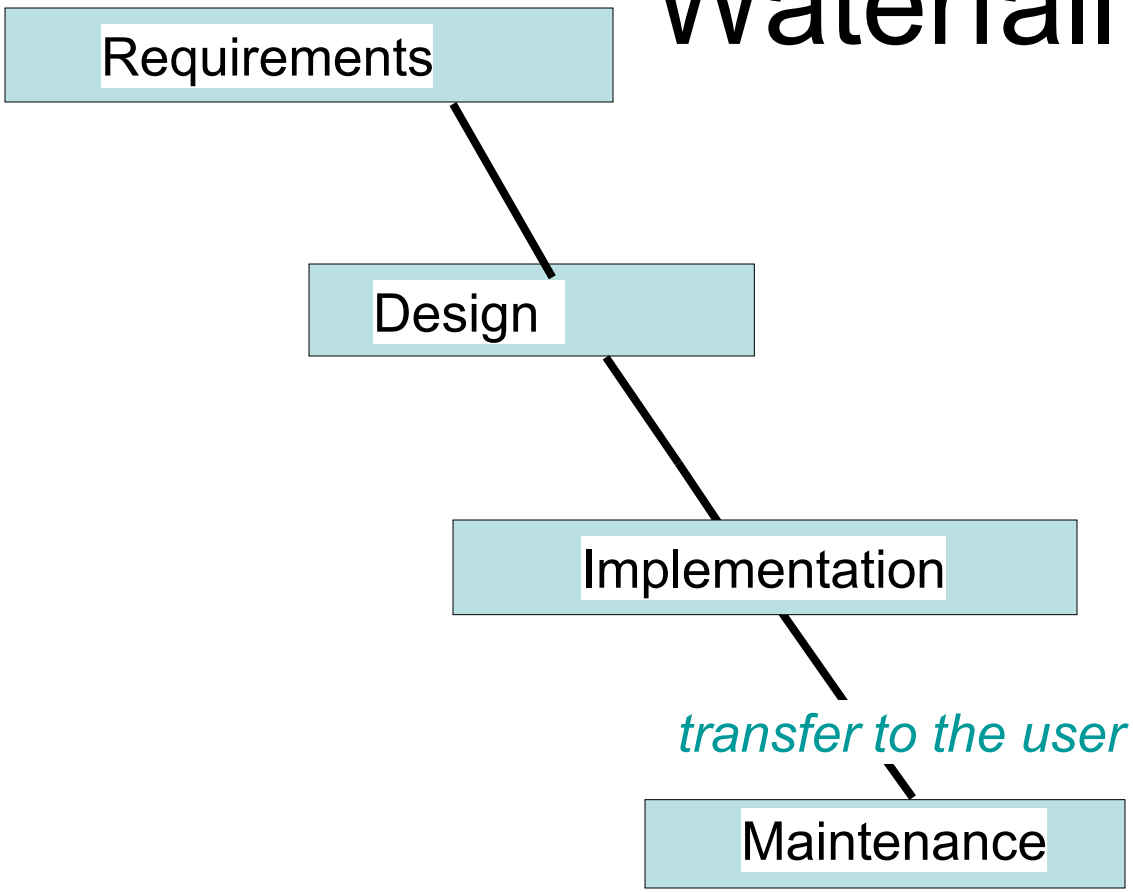
- Used in construction and manufacturing
  - collect the requirements
  - create a design
  - follow the design during the entire construction
  - transfer finished product to the user
  - solve residual problems through maintenance
- Intuitively appealing metaphor
  - good design avoids the expensive late rework
  - waterfall became the dominant paradigm

Exponential cost of change

- major contributor to software cost



Waterfall model



Waterfall paradigm

- Elaborate up-front activities
  - BDUF (Big Design Up Front)
- Textbooks
  - Still largely based on the waterfall

Anomaly of Requirements volatility

- 30% or more requirements may change during development (Cusumano and Selby)
  - this is the direct result of the team's learning process and software interoperability
- Caper-Jones: Requirements for IT change at a rate 2 – 3% per month

Standish group anomaly

- In 1995
  - 31% of all software projects were cancelled
  - 53% were “challenged” (completed only with great difficulty, with large cost or time overruns, or substantially reduced functionality)
  - only 16% could be called successful
- Obviously, the waterfall metaphor did not solve the problems of software development

Band-Aid: Anticipation of changes

- If changes can be anticipated at design time, they can be controlled by a parameterization, encapsulations, etc.
  - waterfall model still can be used
- Experience confirms:
  - many changes are not anticipated by the original designers
  - inability to change software quickly and reliably means that business opportunities are lost
  - only a band-aid solution

Band-Aid: Prototyping

- Create a prototype to capture requirements
- Problem: volatility continues after prototype has been completed
- Another band-aid

Paradigm shift of ~ 2000

- New life span models emphasize software evolution
  - staged model of software lifespan
  - based on data from long-lived systems
- Iterative development
  - Rational Unified Process
- Agile development
  - SCRUM
  - Extreme programming

Summary of paradigms

- The waterfall paradigm tried to freeze requirements for the duration of software development
  - led to too many project failures
- The new paradigm emphasizes software evolution
  - interoperability and complexity cause volatility
    - volatility is a consequence of essential properties

All three paradigms currently coexist

- Ad hoc paradigm still used by some single programmers
  - programming as an art rather than engineering
  - example: small games
- Waterfall works if there is no volatility
  - small or short-lived projects
  - unusually stable requirements and environments
  - some managers still insist on it
- Evolutionary paradigm is the mainstream