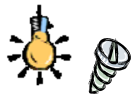
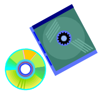
Software engineering

What makes software development hard?



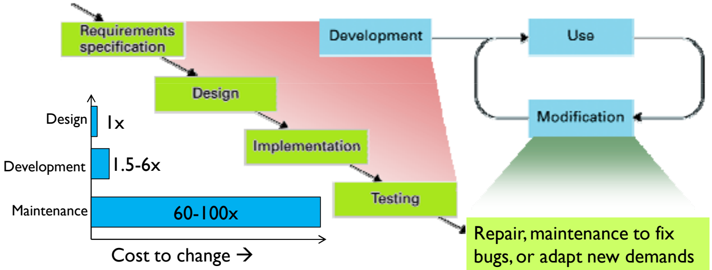


	Traditional Engineering	Software Engineering
"Off the shelf" components available	Often	Rarely
Required performance	Within tolerances	Perfect
Quality metrics	Mean time to failure	Unclear
Scientific basis	Physics	Unclear

Size makes differences

	Small program	Working system
Code size (lines)	Tens to hundreds	10^4~10^7
Complexity	Low	High
Repeated updates	No	Yes
Developer(s)	Usually one person	Usually many people
Reliability requirement	Low	High

The software life cycle



More efforts put on the early development will win tremendous payoff later

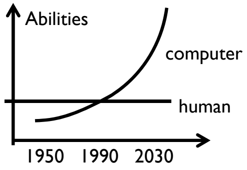
OR, discard old software and redevelop the new one from scratch!?

Waterfall model

- Analysis (requirement specification)
  - Identify the needs of the users, and compiles them to requirements, further to technical specifications
- Design
  - Focus on how to accomplish these specifications
  - Applies modular decomposition to breakdown the entire complexity
- Implementation
  - Actual coding, creating data files & database
- Testing
  - Tightly coupled with implementation, bottom-up from each module

Analysis phase: requirement specification

- Stakeholder: future users
  - From an entity, such as a company or agency
  - From free markets, such as the Internet
- Software requirements specification
  - Wants, needs, costs, and feasibility
  - Hardware, software, data, human factors
  - Economic considerations and technical considerations



Design phase: modularization

- Modules: the division of software into manageable units,
  - Ex: the procedures or objects
- Goal: Minimize coupling & maximize cohesion
- Coupling: the independence between modules
  - Control coupling: a module passes control to another module
    - EX: module A calls module B
  - Data coupling: sharing data between modules
    - Implicit coupling: **global variables** (BAD)
- Cohesion: internal binding within a module
  - Logical cohesion: logical similarity (not very good)
  - Functional cohesion: components are focused around performance of a single activity (better)

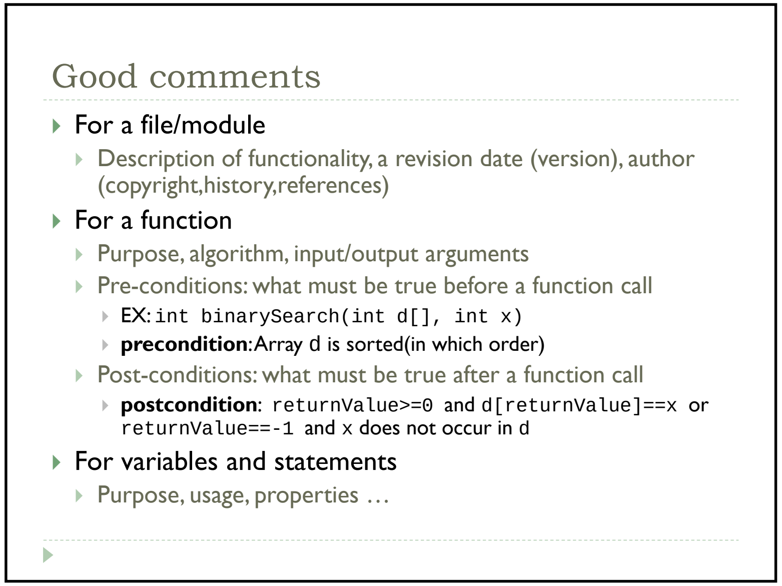
Implementation phase

- Do you know what this code does?

```
int i;main(){for(;i["]<i;++i){--i;}"];re[illegible]llo,
world!\n",'/'/'/'));}read(j,i,p){write([illegible],i/i);}
```

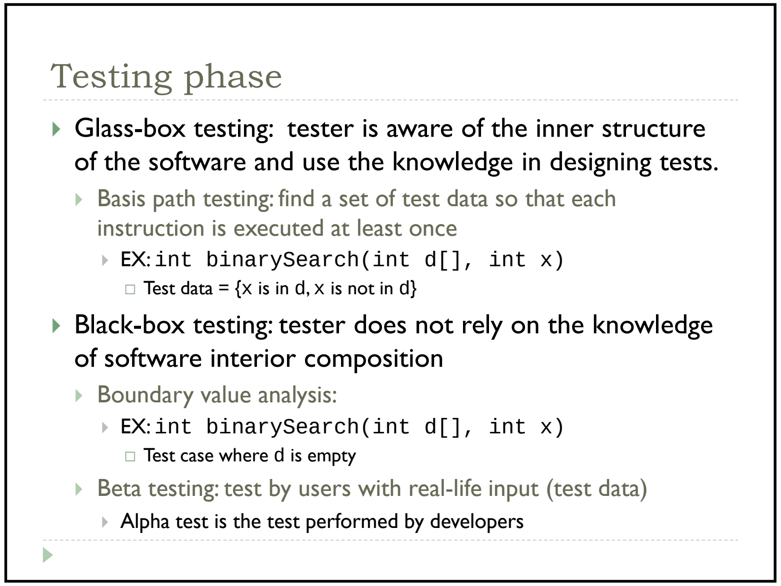
DON'T DO THAT

- It can be compiled and executed, but unless you want to show how bad a programming style can be
- Programming style: rules to help programmers to read and understand source code and to avoid bugs/errors
  - Ex: Clear statements and type definitions
    - EX1: **char* dest, src;** //what's **src**'s type?
    - EX2: ***p++;** // which value is increased, p or *p?
  - Ex: Consistent naming conventions
  - …



Good comments

- For a file/module
  - Description of functionality, a revision date (version), author (copyright,history,references)
- For a function
  - Purpose, algorithm, input/output arguments
  - Pre-conditions: what must be true before a function call
    - EX: `int binarySearch(int d[], int x)`
    - **precondition**: Array `d` is sorted(in which order)
  - Post-conditions: what must be true after a function call
    - **postcondition**: `returnValue>=0` and `d[returnValue]==x` or `returnValue==-1` and `x` does not occur in `d`
- For variables and statements
  - Purpose, usage, properties …



Testing phase

- Glass-box testing: tester is aware of the inner structure of the software and use the knowledge in designing tests.
  - Basis path testing: find a set of test data so that each instruction is executed at least once
    - EX: `int binarySearch(int d[], int x)`
      - □ Test data = {x is in d, x is not in d}
- Black-box testing: tester does not rely on the knowledge of software interior composition
  - Boundary value analysis:
    - EX: `int binarySearch(int d[], int x)`
      - □ Test case where `d` is empty
  - Beta testing: test by users with real-life input (test data)
    - Alpha test is the test performed by developers