What is Software Quality?

Readable?
Commented?
Documented?
Tested?
Correct?
Reliable?
Secure?
Maintainable?
Portable?
An economic view of software quality

Development takes time.
Time "is" money.

High-quality software has:
- Low cost of development and maintenance.
- High resulting value.

So, let's consider value-cost as a reasonable measure of software quality.
What are value factors?
Wednesday, September 05, 2012
4:12 PM

Sales?
Performance?
Usability?
Fitness for purpose?
What are cost factors?

Time to program?
Time to test/debug?
Time to make changes?
Required resources?
Deployment time?
Maintenance?
"Software engineering is planning for change."

- New functionality
- Changes in environment (port to Android)
- Fix bugs.
- Changes in staffing.
Some caveats

I am not a "quality fascist."
I do not believe in software quality for its own sake.
Some quality is justifiable -- but not all.
Traditional principles of software quality

1. Plan for changes in:
   a. Staffing.
   b. Requirements.
   c. Customer base.
   d. Platform/architecture.

2. Leave behind complete documentation of all phases of development.

3. Spend more now, less later.
In Brooks' "the tar pit", a distinction is made between

○ A **program**
○ A **programming product**: documentation, support, maintenance.
○ A **systems program**: a program that couples closely with its environment.
○ A **systems programming product**: documentation, support, maintenance.
Realities of software engineering

○ No one is an island: real software development requires large teams of programmers.
○ The hardest part of creating a program is breaking it up into pieces that can be crafted separately.
○ Almost all of software engineering is about enabling teams to work and interact effectively.
The fundamental principle of software engineering

The fundamental principle: break up projects into pieces to:

- **Maximize cohesion**: the property of a piece that it is about or describes one thing or kind of thing. A cohesive project is easier for a human to understand.

- **Minimize coupling**: the property of a piece that it depends upon other pieces. An uncoupled piece is easier to develop independently from other pieces.
Why is coupling bad?

○ Coupling between things translates to communication between the humans creating the things.

○ Communication is a dominant cost of software development.

○ It can be lessened by effective documentation, but the more the coupling, the more extensive the documentation must be.

○ Example: Operating system manual!
Application of the cohesion/coupling principle: the **waterfall model** of software development.

- Break software development into several **cohesive phases**.
- Each phase exhibits **minimal coupling** with the next phase.
Phases of the waterfall model:
  ○ **Systems engineering**: deciding upon the environment in which the software will run.
  ○ **Requirements analysis**: deciding what the software will do, but not how it will do it.
  ○ **Design**: deciding how requirements will be met.
  ○ **Implementation**: creating software according to the design.
  ○ **Testing**: verifying function of the implemented product.
  ○ **Maintenance**: making changes over time as requirements change.
"Traditional" software engineering

○ Complete each step of the waterfall model in sequence.
○ Each phase must be complete before the next phase begins.
○ Each phase's deliverables are subjected to formal review before the next phase is started.
Requirements: entities and relationships

Requirements analysis: determine what software should do.
Key part of requirements analysis: model software as one part of a complex transaction between user and computer.

In this model:

**Entities** are parts of the system.
**Relationships** are how they interact.
Cohesion and coupling

Wednesday, September 05, 2012
2:12 PM

Design: **cohesion** and **coupling**

All software design techniques maximize cohesion and minimize coupling.

Primary goal of design: break project into pieces that can be completed by one programmer.

Secondary goal: document what should be done so that others can take over if a programmer leaves.
Testing: Verification and Validation

**Verification:** Are we building the product right? Is it conformant with what we documented?

**Validation:** Are we building the right product? Is it really what the customer needs?
Agile methods

Problem: often precise requirements are not known in advance.
Solution: determine requirements as part of coding, via a feedback loop with customers.

Talk with customer about initial requirements.
"Code sprint": create a mockup of those requirements.
"Is this what you wanted?"
In-class exercise
Wednesday, September 09, 2009
2:07 PM

The first three phases of the waterfall model:

- **Systems Engineering**: describing the environment in which software will operate
- **Requirements**: what software will do.
- **Design**: how requirements will be met.

Can you tell the difference?

Break into groups of four people and see!