Lecture 12: Software Quality

CS190: Software System Design

March 20, 2002
Steven P. Reiss

I. Today's Class

A. What is software engineering

1. Developing techniques to produce high quality software economically

2. We know what economically means -- measurable

3. What does high quality mean

B. What I'd like to look at today is the meaning of software quality

II. Background

A. What do we mean by quality software?

1. Might vary with perspective

a) Tester: meets requirements

b) User: easy to use, efficient

c) Programmer: few bugs

d) Maintainer: easy to modify

2. Books on the subject

a) Listing various traits

3. Models

a) ISO 9001 :: software quality model

B. Standards imply measurements

1. This is the area called software metrics

2. What do you measure

a) What can you measure

b) What is meaningful to measure

III. Software Metrics

A. Most try to measure complexity

1. How do you determine software complexity
2. Lines of code
3. Number of entries
4. Size of functions
5. Numbers of levels of nested loops/conditions
6. Dependencies between classes
7. Cohesion within a class

B. Others try to measure the process

1. Lines of code written per day
2. Number of bugs in a region

C. We covered many of these in discussing Tracking

D. Effectiveness

1. Too many variables in coding to have these be predictive overall
2. Lines of code tends to correlate best
3. Bugs correlate with future bugs
4. Mixed results on other metrics

IV. Software Quality

A. Correctness

1. Program satisfies specifications
2. Program fulfills requirements (user's objectives)

B. Reliability

1. The extent to which a program can be expected to perform its intended function
2. Number of bugs
3. Degree of precision
4. Failure rate

C. Efficiency

1. Amount of resources needed
2. Overall or for a particular command or function

D. Integrity

1. Controlling access to software or data by unauthorized persons
2. Ensuring that data remains consistent in all cases

E. Usability

1. How usable is the program
2. Learnability
3. Operability
4. Ease of preparing input
5. Ease of intepreting the output
6. User usage errors
7. Quality of the user interface

F. Maintainability

1. Effort required to locate and fix bugs
2. Effort required to understand the software by new programmers

G. Flexability

1. Effort required to modify the system
2. Effort required to adapt the system to changing requirements

H. Testability

1. Effort required to test a program
2. Completeness of test suite
3. Client-server examples
4. User interface examples

I. Portability

1. Effort required to adapt to new hardware
2. Effort required to adapt to new software environment

J. Reusability

1. How much of the program can be reused in other applications
2. How much of the program came from other applications

K. Interoperability

1. **Effort required to couple one system to another**

V. Discussion

A. Phases of software engineering

1. **Most of these are aimed toward maintenance**
2. **More programmer oriented than user oriented**
3. **Doesn't measure quality of code or design directly**
   a) These are measured indirectly (how)

B. How does this interact with XP

1. **XP puts the stress on SIMPLICITY**
   a) Also on testing
2. **Simplicity implies a lot of these constraints**
3. **How does XP methodology affect quality?**

C. What can be measured accurately

1. **Are there meaningful measures for most of these**
2. **Are the measures in any way predictive**