THOMSON
COURSE TECHNOLOGY

Ethics in Information Technology, Second Edition

Chapter 7

Software Development & Quality Assurance

Objectives

- Why do companies require high-quality software in business systems, industrial process control systems, and consumer products?
- What ethical issues do software manufacturers face in making tradeoffs between project schedules, project costs, and software quality?

Objectives (continued)

- What are the essential components of a software development methodology, and what are its benefits?
- What is a safety-critical system, and what actions are required during its development?

Strategies to Engineer Quality Software

- High-quality software systems
  - Operate safely and dependably
  - Have a high degree of availability
  - Required to support the fields of
    - Air traffic control
    - Nuclear power
    - Automobile safety
    - Health care
    - Military and defense
    - Space exploration

Strategies to Engineer Quality Software (continued)

- More and more users are demanding high quality software
- Software defect
  - Could cause a system to fail to meet users' needs
  - Impact may be trivial or very serious
  - Patches may contain defects
- Software quality
  - Degree to which software meets the needs of users

Strategies to Engineer Quality Software (continued)

- Quality management
  - How to define, measure, and refine the quality of the development process and products
  - Objective
    - Help developers deliver high-quality systems that meet the needs of users
- Deliverables
  - Products such as:
    - Statements of requirements
    - Flowcharts
    - User documentation

Strategies to Engineer Quality Software (continued)

- Primary cause for poor software quality
  - Developers do not know how to design quality into software
  - Or do not take the time to do so
- Developers must
  - Define and follow a set of rigorous engineering principles
  - Learn from past mistakes
  - Understand the environment in which systems operate
  - Design systems relatively immune to human error

Strategies to Engineer Quality Software (continued)

- Programmers make mistakes in turning design specifications into code
  - About one defect for every 10 lines of code
- Pressure to reduce time-to-market
- First release
  - Organizations avoid buying the first release
  - Or prohibit its use in critical systems
  - Usually has many defects

Legal Overview: Software Product Liability

- Product liability
  - Liability of manufacturers, sellers, lessors, and others for injuries caused by defective products
- Strict liability
  - Software maker held responsible for the injury
  - Regardless of negligence or intent
  - Must prove only that the software product is defective or unreasonably dangerous and that the defect caused the injury

Legal Overview: Software Product Liability (continued)

- Strict liability
  - No requirement to prove that the manufacturer was careless or negligent
    - Or to prove who caused the defect
  - All parties in the chain of distribution are liable
- Legal defenses used against strict liability
  - Doctrine of supervening event; *software was altered*
  - Government contractor defense; *software specifications were provided by the government*
  - Expired statute of limitations; *injury occurred a long time ago*

Legal Overview: Software Product Liability (continued)

- Negligence
  - A supplier is not held responsible for every product defect that causes a customer or third-party loss
  - Responsibility is limited to defects that could have been detected and corrected through “reasonable” software development practices
  - Area of great risk for software manufacturers
  - Defense of negligence may include
    - Legal justification for the alleged misconduct
    - Demonstrate that the user’s own actions contributed to injuries (Contributory negligence)

Legal Overview: Software Product Liability (continued)

- Warranty
  - Assures buyers or lessees that a product meets certain standards of quality
  - Expressly stated
  - Implied by law
- Breach of warranty claim
  - User must have a valid contract that the supplier did not fulfill
  - Can be extremely difficult to prove
    - Because the software supplier writes the warranty
    - Example: Mortenson company vs. Timberline Software

Software Development Process

- Large software project roles
  - System analysts
  - Programmers
  - Architects
  - Database specialists
  - Project managers
  - Documentation specialists
  - Trainers
  - Testers

Software Development Process (continued)

- Software development methodology
  - Work process
  - Controlled and orderly progress
  - Defines activities and individual and group responsibilities
  - Recommends specific techniques for accomplishing various activities
  - Offers guidelines for managing the quality of software during various stages of development

Software Development Process (continued)

- Safer and cheaper to avoid software problems at the beginning than to attempt to fix damages after the fact
  - Identify and remove errors early in the development process
    - Cost-saving measure
    - Most efficient way to improve software quality

Software Development Process (continued)

- Dynamic testing
  - Black-box testing
    - Tester has no knowledge of code
  - White-box testing
    - Testing all possible logic paths through the software unit
    - With thorough knowledge of the logic
    - Make each program statement execute at least once

Software Development Process (continued)

- Static testing
  - Static analyzers are run against the new code
  - Looks for suspicious patterns in programs that might indicate a defect
- Integration testing
  - After successful unit testing
  - Software units are combined into an integrated subsystem
  - Ensures that all linkages among various subsystems work successfully

Software Development Process (continued)

- System testing
  - After successful integration testing
  - Various subsystems are combined
  - Tests the entire system as a complete entity
- User acceptance testing
  - Independent testing
  - Performed by trained end users
  - Ensures that the system operates as they expect

Development of Safety-Critical Systems

- Safety-critical system
  - Failure may cause injury or death
  - Examples
    - Automobile's antilock brakes
    - Nuclear power plant reactors
    - Airplane navigation
    - Roller coasters
    - Elevators
    - Medical devices

Development of Safety-Critical Systems (continued)

- Key assumption
  - Safety will *not* automatically result from following the organization's standard development methodology
- Must go through a more rigorous and time-consuming development process than other kinds of software
- All tasks require
  - Additional steps
  - More thorough documentation
  - More checking and rechecking

Development of Safety-Critical Systems (continued)

- Redundancy
  - Provision of multiple interchangeable components to perform a single function
  - In order to cope with failures and errors
- N-version programming
  - Form of redundancy
  - Involves the execution of a series of program instructions simultaneously by two different systems
  - Uses different algorithms to execute instructions that accomplish the same result

Development of Safety-Critical Systems (continued)

- N-version programming
  - Results from the two systems are compared
  - If a difference is found, another algorithm is executed to determine which system yielded the correct result
  - Instructions for the two systems are often:
    - Written by programmers from two different companies
    - Run on different hardware devices
  - Both systems are highly unlikely to fail at the same time under the same conditions