Course - Administrative Issues

NO lecture next week (October 21)
  • prepare for presentations

Next lecture (October 28)
  • first set of student presentations

Presentation format
  • 6 students per session (10 minutes + 3 minutes Q+A each)
  • we will use instantaneous online survey mechanism for peer review

Schedule
  • full schedule has been announced on course website
  • report time conflicts by next Tuesday (October 21)
  • suggest alternative dates
  • “too early” is not a good justification for conflict
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?
Impact of Quality Software

Software errors can have minor or major consequences

- software in dryer may cause clothes not being dried enough
- software in X-ray scanner may overexpose patient to powerful X-rays

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
Ethical decisions involve:

• tradeoff between quality and other factors, such as ease of use, time to market, and development costs.
• some managers may have a short-term profit-oriented view
• others may prefer the more ethical view of delivering high-quality software
• it is a reputation’s game
• need to also review legal implications of software errors
Software product liability

- accidents due to software errors may result in lawsuits and punitive damages
- liability is commonly referred to as product liability
- there is no federal liability law, software liability falls under common law
- strict liability means manufacturer is responsible for regardless of negligence or intent → but there are lines of defense against this
- responsibility may be limited to harmful defects that could have been detected through ‘reasonable’ software practices
- there is also the concept of ‘contributory’ negligence (e.g., accidentally cut finger using nail clippers)
- warranty also protects consumer, but may be hard to read
Inexperienced or quality-ignorant software coding

- quality software evolves right from the start
- but few have the conscience to do it

Human error

- programmers inject one defect for every 10 lines of code
- e.g., Windows XT, 400 M lines of code, even if 99.9% was clean there still would be 1 bug per 10,000 lines of code → large software still contains thousands of bugs

Time pressure

- competition requires fast roll-out with more features
- one can always patch..
- ‘testing’ done by customers..
- some avoid buying the first version (Windows OS → NT was mature)
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
• even patches may contain (new) defects

Software quality

• degree to which software meets the needs of users
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
  - 100 times less cost when bug is detected early before product roll-out
  - bug effect (and its fix) may ripple through large pieces of the software
Dynamic testing

• Black-box testing
  - want code to demonstrate expected output behavior for all input data in test suite
  - tester has no knowledge of code
• White-box testing ( tester has knowledge of code)
  - testing all possible logic paths through the software unit
  - with thorough knowledge of the code’s logic paths
  - make each program statement execute at least once
  - for example, for program to calculate employee gross pay, want test case for less than 40 hours and test case for more than 40 hours. Why?
Dynamic testing

- **Black-box testing**
  - want code to demonstrate expected output behavior for all input data in test suite
  - tester has no knowledge of code

- **White-box testing** (tester has knowledge of code)
  - testing all possible logic paths through the software unit
  - with thorough knowledge of the code’s logic paths
  - make each program statement execute at least once
  - for example, for program to calculate employee gross pay, want test case for less than 40 hours and test case for more than 40 hours. Why?

  … to check calculations for overtime pay
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
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
Safety-Critical Systems

Consequences of software defects in certain systems can be deadly

- companies must take special precautions

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
- example: bug in Therac-25 radiation therapy machine 1985-87
  - wrong sequence of menu selections caused large radiation dose to be delivered to the patient
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)

Project safety engineer

- explicit responsibility for the system’s safety
- uses a logging and monitoring system to track hazards from the project’s start to finish

Hazard log

- used at each stage of the software development process
- assesses how it has accounted for detected hazards
Safety reviews
  • held throughout the development process

Robust configuration management system
  • tracks all safety-related documentation

Formal documentation required
  • including verification reviews and signatures

Key issue
  • deciding when QA staff has performed enough testing
Risk

- probability of an undesirable event occurring times the magnitude of the event’s consequences if it does happen.
- consequences include
  - damage to property
  - loss of money
  - injury to people
  - death
ISO 9000 standard

- guide to quality products, services, and management
- organization must submit to an examination by an external assessor
- requirements:
  - written procedures for everything it does
  - follow those procedures
  - prove to the auditor the organization fulfilled the first two requirements
Failure mode and effects analysis (FMEA)

- important technique to develop an ISO 9000 compliant system
- used to evaluate reliability
- determine the effect of system and equipment failures
- goal: identify potential design and process failures early in a project
Failure mode and effects analysis (FMEA)

- Failure mode:
  - describes how a product or process could fail
- Effect
  - adverse consequence that a customer might experience
- seldom is a one-to-one relationship between cause and effect
DO-178B/EUROCCAE ED-128

- evaluation standard for the international aviation community
- developed by Radio Technical Commission for Aeronautics (RTCA)
### Manager’s Checklist for Improving Software Quality

**TABLE 7-2  Manager’s checklist for improving software quality**

<table>
<thead>
<tr>
<th>Questions</th>
<th>Yes</th>
<th>No</th>
</tr>
</thead>
<tbody>
<tr>
<td>Has senior management made a commitment to quality software?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Have you used CMMI to evaluate your organization’s software development process?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Have you adopted a standard software development methodology?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Does the methodology place a heavy emphasis on quality management and address how to define, measure, and refine the quality of the software development process and its products?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Are software project managers and team members trained in the use of this methodology?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Are software project managers and team members held accountable for following this methodology?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Is a strong effort made to identify and remove errors as early as possible in the software development process?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>In the testing of software, are both static and dynamic testing used?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Are white-box testing and black-box testing used?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Has an honest assessment been made to determine if the software being developed is safety-critical?</td>
<td></td>
<td></td>
</tr>
<tr>
<td>If the software is safety-critical, are additional tools and methods employed, and do they include the following: project safety engineer, hazard logs, safety reviews, formal configuration management systems, rigorous documentation, risk analysis processes, and the FMEA technique?</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Summary

More and more users are demanding high quality software.

Software product liability claims are frequently based on:

- strict liability
  → held responsible for injury regardless of negligence or intent
- negligence
- breach of warranty
- misrepresentation (of product quality or hide defect in product)
Software development methodology

• defines activities in the software development process
• defines individual and group responsibilities
• recommends specific techniques
• offers guidelines for managing the quality of products

CMMI: Capability Maturity Model Integration for software

• defines five levels of software development maturity
• identifies the issues most critical to software quality and process improvement
• can serve as benchmark to compare companies in the awarding of software contracts

Safety-critical system

• failure may cause injury or death