Software Reliability Engineering:

An Introduction

Objectives

- Introduce some concepts of software reliability engineering
  - Focus on failures, not defects
  - Operational profiles
  - Measuring reliability
- These topics will be covered in more detail in next class session

Defects vs. Reliability

- Defects are a developer view of quality
  - "All defects are not created equal"
  - Defects in more frequently used or more critical sections of the code matter a lot more

- Reliability and failures are the user view of quality
  - How frequently does the software fail in typical usage?
  - A "failure" is when the user cannot get their work done using the software
  - Note that this is against actual user needs, not the specification

Error –> Defect –> Fault –> Failure

- A human or tool error results in a defect
  - The defect is the cause of the problem
  - Occurs at development time
- A defect in the software may (or may not) lead to a “fault” at execution time
  - Depends on whether the erroneous code is encountered
  - Depends on whether the erroneous code produces wrong results, given the specifics of the computation / situation
- A fault may or may not lead to a “failure” – behavior of software that does not meet customer needs
  - There may be incorrect behavior that does not matter to the user
  - The software may be fault-tolerant, so that the fault does not cause a failure, for example, dropped packet gets retransmitted

Measuring Reliability

- Create operational profiles
  - Identify the set of operations and their relative frequency
- Create automated system tests
  - Test all the operations, and build an automated verifier that checks whether the operation produced the right result
- Run system tests repeatedly, in random order, with relative frequencies matching the operational profile
  - Mimicking actual use of the software
- Track the frequency of failures and plot graphs
  - Measures reliability, results highly valid
    - Subject to accuracy of the operational profile
    - Much better measure of likely user experience than the alternatives

Operational Profiles

- To measure reliability, we need to know how the software is used

- We need an "operational profile":
  - Set of user operations, with relative frequency of each operation
  - Focus quality assurance efforts on the most frequently used and most critical operations

- The set of operations is known from the use cases
  - In requirements engineering, need to gather information about the relative frequency of different operations

Creating an Operational Profile

- Sample application: Word Processor

Operation	**Frequency**	**Approx. Relative Freq.**
Open file	1/session (5 session/day)	0.001
Close file	1/session	0.001
Save file	25/session	0.04
Insert text	1000/session	1.0
Cut-and-paste	6/session	0.006
Check spelling	1000/session	1.0
Repaginate	100/session	0.1
Upgrade software	1/ 6 months	0.000001

Value of Operational Profiles

- Knowing which operations users perform most frequently helps in:
  - Release Planning: Which features to develop first
  - Where to put in more design, inspection and testing effort
  - Testing that focuses on what is most relevant to user
  - Performance Engineering: Knowing usage hotspots
  - Usability: Designing GUIs - menus, hotkeys, toolbars
  - Implementing Workflow: Automate most frequent operations (wizards), or streamline the flow between them

- Obviously, this is critical information to gather during requirements!

Automating Testing

- Create test cases for each operation, based on equivalence classes
    - Randomize the input parameters
        - Randomly pick which equivalence class, value within equivalence class
- Build a verifier, which performs the same operation as the software, but in simpler ways
    - Uses simple internal computational model to keep track of the state of the system and expected results of operations
    - Failure if actual result does not match expected result
- Can run millions of tests instead of hundreds
    - Same tests but in different sequence and with different input values may result in different behaviors (because internal state is different)
    - Those are the kinds of defects that usually make it to the field

Are Automated Verifiers Feasible?

- Verifier takes same sequence of inputs as actual software, performs computations using algorithmic models of expected behavior, and generates "expected result" values
  - Database operations can be modeled with collections
  - Embedded operations such as sending and receiving messages can be modeled with state machines
  - Document manipulation can be modeled with collections
- Often complexity of verifier comparable to actual software
  - But no need for GUIs, file/database I/O, exception handling, sending/receiving messages, compression/decompression
- Note that cost of development is only a fraction of the cost of testing, especially for high-reliability and safety-critical software
  - Automated testing saves a lot, and achieves higher reliability

Tracking Failures

- Plot failure rates vs. time during development
  - Results in "reliability growth curve"
  - Shows how quality of software is changing as development progresses
- Can also be used for reliability certification
  - Can run enough tests to evaluate whether a given reliability target is met (within a statistical confidence interval)
    - Example: "95% confidence that the failure intensity <= 4 failures per 100,000 hours of operation"
  - Very useful as acceptance criteria for customers
  - Also very useful when you depend on external software such as compilers, operating systems, libraries, etc.
- We can generate MTBF ("mean time between failures") numbers for software, just like other engineering fields!

Conclusion

- A focus on failures and reliability complements a focus on defects
  - A customer view versus a developer view
- Operational profiles focus quality assurance (and other) efforts
  - Focus on the operations (features) used most often
- Automated testing helps identify failures and failure rates
  - Track failure intensity
  - Reliability growth curves
  - Statistical mean time between failures – an availability metric

- We will cover these more next class session