Today’s Lecture

- Quality assurance
- Back to the future: testing
What Do These Have in Common?

- First launch of space shuttle
- Airbus 320 [http://catless.ncl.ac.uk/Risks/10.02.html#subj1.1](http://catless.ncl.ac.uk/Risks/10.02.html#subj1.1)
- Audi 5000
- Mariner 1 launch: [http://catless.ncl.ac.uk/Risks/5.73.html#subj2.1](http://catless.ncl.ac.uk/Risks/5.73.html#subj2.1)
- AT&T telephone network
- Ariane 5 [http://catless.ncl.ac.uk/Risks/18.24.html#subj2.1](http://catless.ncl.ac.uk/Risks/18.24.html#subj2.1)
- Y2K
Impact of Failures

- Not just “out there”
  - Space shuttle
  - Mariner 1
  - Ariane 5

- But also “at home”
  - Your car
  - Your call to your mom
  - Your homework
  - Your hospital visit

Peter Neumann’s Risks Forum: http://catless.ncl.ac.uk/Risks
Verification and Validation

- Verification
  - Ensure software meets specifications
  - Internal consistency
  - “Are we building the product right?”

- Validation
  - Ensure software meets customer’s intent
  - External consistency
  - “Are we building the right product?”
Software Qualities

- Correctness
- Reliability
- Robustness
- Performance
- User friendliness
- Verifiability
- Maintainability
- Repairability
- Safety

- Evolvability
- Reusability
- Portability
- Understandability
- Interoperability
- Productivity
- Size
- Timeliness
- Visibility
Quality Assurance

- Assure that each of the software qualities is met
  - Goals set in requirements specification
  - Goals realized in implementation
- Sometimes easy, sometimes difficult
  - Portability versus safety
- Sometimes immediate, sometimes delayed
  - Understandability versus evolvability
- Sometimes provable, sometimes doubtful
  - Size versus correctness
An Idealized View of QA

Complete **formal specification** of problem to be solved

**Design**, in formal notation

**Code**, in verifiable language

Executable **machine code**

Execution on **verified hardware**
A Realistic View of QA

Mixture of **formal** and **informal** specifications

Design, in mixed notation

Code, in C++, Java, Ada, ...

Pentium **machine code**

Compilation by commercial compiler

Execution on **commercial hardware**
No matter how sophisticated the QA process, the problem of creating the initial specification remains.
Second Complication

- Complex data communications
  - Electronic fund transfer
- Distributed processing
  - Web search engine
- Stringent performance objectives
  - Air traffic control system
- Complex processing
  - Medical diagnosis system

Sometimes, the software system is extremely complicated making it tremendously difficult to perform QA
It is difficult to divide the particular responsibilities involved when performing quality assurance
Fourth Complication

- Quality assurance lays out the rules
  - You will check in your code every day
  - You will comment your code
  - You will…
- Quality assurance also uncovers the faults
  - Taps developers on their fingers
  - Creates image of “competition”
- Quality assurance is viewed as cumbersome
  - “Just let me code”

Quality assurance has a negative connotation
Available Techniques

- Formal program verification
- Static analysis of program properties
  - Concurrent programs: deadlock, starvation, fairness
  - Performance: min/max response time
- Code reviews and inspections
- Testing

Most techniques are geared towards verifying correctness
V-Model of Development and Testing

- Develop Requirements
  - Requirements Review
- Design
  - Design Review
- Develop Acceptance Tests
  - Acceptance Test Review
- Develop Integration Tests
  - Integration Tests Review
- Code
  - Code Review
- Develop Unit Tests
  - Unit Tests Review
- Execute System Tests
- Execute Integration Tests
- Execute Unit Tests
Implementation/Testing Interaction

Implementation (previous lectures)

Testing (this lecture)
Testing

- Exercise a module, collection of modules, or system
  - Use predetermined inputs ("test case")
  - Capture actual outputs
  - Compare actual outputs to expected outputs
- Actual outputs equal to expected outputs
  \[\rightarrow\]
  test case \textit{succeeds}
- Actual outputs unequal to expected outputs
  \[\rightarrow\]
  test case \textit{fails}
Testing Terminology

- **Failure**
  - Incorrect or unexpected output
  - Symptom of a fault

- **Fault**
  - Invalid execution state
  - Symptom of an error
  - May or may not produce a failure

- **Error**
  - Defect or anomaly in source code
  - Commonly referred to as a “bug”
  - May or may not produce a fault
Testing Goals

- Reveal failures/faults/errors
- Locate failures/faults/errors
- Show system correctness
  - Within the limits of optimistic inaccuracy
- Improve confidence that the system performs as specified (verification)
- Improve confidence that the system performs as desired (validation)

Program testing can be used to show the presence of bugs, but never to show their absence [Dijkstra]
Levels of Testing

- **Unit testing**
  - Testing of a single code unit
  - Requires use of test drivers

- **Integration testing**
  - Testing of interfaces among integrated units
    - Incremental
    - “Big bang”
  - Often requires test drivers and test stubs

- **Acceptance testing**
  - Testing of complete system for satisfaction of requirements
Test Tasks

- Devise test cases
  - Target specific areas of the system
  - Create specific inputs
  - Create expected outputs

- Choose test cases
  - Not all need to be run all the time
    » Regression testing

- Run test cases
  - Can be labor intensive

All in a systematic, repeatable, and accurate manner
Two Approaches

◆ White box testing
  – **Structural testing**
  – Test cases designed, selected, and ran based on structure of the code
  – Scale: tests the nitty-gritty
  – Drawbacks: need access to source

◆ Black box testing
  – **Specification-based testing**
  – Test cases designed, selected, and ran based on specifications
  – Scale: tests the overall system behavior
  – Drawback: (less systematic) less thorough
Test Oracles

- Provide a mechanism for deciding whether a test case execution succeeds or fails
- Critical to testing
  - Used in white box testing
  - Used in black box testing
- Difficult to automate
  - Typically relies on humans
  - Typically relies on human intuition
  - Formal specifications may help
Example

- Your test shows \( \cos(0.5) = 0.8775825619 \)
- You have to decide whether this answer is correct?
- You need an oracle
  - Draw a triangle and measure the sides
  - Look up cosine of 0.5 in a book
  - Compute the value using Taylor series expansion
  - Check the answer with your desk calculator
Use the Principles — Even in Testing

- Rigor and formality
- Separation of concerns
  - Modularity
  - Abstraction
- Anticipation of change
- Generality
- Incrementality