Software Quality

- Principles
- Testing/metrics/inspections

Quality Assurance

- Achieving Quality
  - Product and software quality does not happen by accident, and is not something that can be added on after the fact
  - To achieve quality, we must plan for it from the beginning, and continuously monitor it day to day
  - This requires discipline
  - Methods and disciplines for achieving quality results are the study of Quality Assurance or QA

Quality Assurance

- Three General Principles of QA
    1. Know what you are doing
    2. Know what you should be doing
    3. Know how to measure the difference

Software Quality Assurance

- QA Principle 1: Know what you are doing
  - In the context of software quality, this means continuously understanding what it is you are building, how you are building it, and what it currently does
  - This requires organization, including having a management structure, reporting policies, regular meetings and reviews, frequent test runs, and so on
  - We normally address this by following a software process with regular milestones, planning, scheduling, reporting, and tracking procedures

Software Quality Assurance

- QA Principle 2: Know what you <u>should be</u> doing
  - In the context of software quality, this means having explicit requirements and specifications
  - These must be continuously updated and tracked as part of the software development and evolution cycle
  - We normally address this by requirements and use-case analysis, explicit acceptance tests with expected results, explicit prototypes, frequent user feedback
  - Particular procedures and methods for this are usually part of our software process

Software Quality Assurance

- QA Principle 3: Know how to measure the difference
  - In the context of software quality, this means having explicit measures comparing what we are doing to what we should be doing
  - Achieved using four complementary methods
    - formal methods
    - testing
    - inspection
    - metrics

Software Quality Assurance

- Formal Methods
  - consists of using mathematical models or methods to verify mathematically specified properties
- Testing
  - consists of creating explicit inputs or environments to exercise the software, and measuring its success
- Inspection
  - consists of regular human reviews of requirements, design, architecture, schedules, and code
- Metrics
  - consists of instrumenting code or execution to measure a known set of simple properties related to quality

Software Quality Assurance

- Formal Methods
  - Formal methods include formal verification (proofs of correctness), abstract interpretation (simulated execution in a different semantic domain, e.g., data kind rather than value), state modelling (simulated execution using a mathematical model to keep track of state transitions), and other mathematical methods
  - In practice, formal methods are used directly in software quality assurance in only a small (but important) fraction of systems

Software Quality Assurance

- Formal Methods
  - Primarily safety-critical systems: onboard flight control systems, nuclear reactor control systems, automobile braking and medical equipment, etc.
  - **2010s:** not directly safety-critical, but important systems: compilers (CompCert), OS kernels (seL4)
  - Use of formal methods requires mathematically sophisticated programmers, and is necessarily a slow and careful process, and very expensive

“High-assurance”

- Traditional definition:
  high-assurance software = safety-critical, ...
  - software whose failure could directly lead to injury or death
- Whether a failure *directly* leads to injury or death isn’t really the point:
  - election hacking (no one is directly hurt, but elections have life-and-death consequences)
  - ransomware
  - exposing a private post where you said mean stuff about your boss ⇒ losing your job ⇒ ...
  - “identity theft”
  - ...
- Perhaps **most** software should be considered high-assurance!

Software Quality Assurance

- **Lightweight** Formal Methods
  - Use of lightweight formal methods requires some mathematical background, and is (generally believed to be) less slow and expensive than traditional formal methods
    - Example: Static type checking
    - Research on types has combined static type checking with dynamic type checking; integrating typing with testing

Software Quality Assurance

- Focus of the Course
  - For these reasons, the vast majority (over 95%) of software quality assurance uses testing, inspection, and metrics instead
  - Example: at the Bank of Nova Scotia, over 80% of the total software development effort is involved in testing!
  - Since you have already been introduced to basic formal verification in CISC 223, and since formal methods are the focus of CISC 422, in this course we will concentrate on testing, inspection, and metrics

Software Quality Assurance

- Testing
  - Testing includes a wide range of methods based on the idea of running the software through a set of example inputs or situations and validating the results
  - Includes methods based on
    requirements (acceptance testing),
    specification and design (functionality and interface testing),
    history (regression testing),
    code structure (path testing),
    and many more

Software Quality Assurance

- Inspection
  - Inspection includes methods based on a human or automated review of the software artifacts
  - Includes methods based on requirements reviews, design reviews, scheduling and planning reviews, code walkthroughs, and so on
  - Helps discover potential problems before they arise in practice

Software Quality Assurance

- Metrics
  - Software metrics includes methods based on using tools to count the use of features or structures in the code or other software artifacts, and compare them to standards
  - Includes methods based on code size (number of source lines), code complexity (number of parameters, decisions, function points, modules, or methods), structural complexity (number or depth of calls or transactions), design complexity, and so on
  - Helps expose anomalous or undesirable properties that may reduce reliability and maintainability

Achieving Software Quality

- Software Process
  - Software quality is achieved by applying these techniques in the framework of a software process
  - There have been many software processes proposed, and there are many software processes in use today
  - It would be difficult to claim that one was better than any other in every situation (so we won't!)

Achieving Software Quality

- Standards and Disciplines
  - Because of the importance of quality in software production and the relatively bad track record of the past, many standards and disciplines have been developed to enforce quality practice
    - Total Quality Management (TQM)
    - Capability Maturity Model (CMM)
    - Personal Software Process (PSP)
    - Microsoft Shared Development Process (SDP)
    - Rational Unified Process (RUP)
    - ISO 9000

Summary

- Software Quality
  - We've seen what quality means, how it applies to software, and what methods we can use to achieve it
  - Next time we will begin by reviewing the software development process, explore a number of software process models, and see how quality fits into the software life cycle

Summary

- Today's References (in course readings book)
  - Kan, Metrics and Models in Software Quality Engineering
    - Chapter 1, What is Software Quality?
  - The Software Quality Assurance Definitions Page
    - http://www.sqa.net