CISC 327
Software Quality Assurance

Lecture 1

Course Summary

Software Quality

CISC 327 - © 2003–2018 J.R. Cordy, S. Grant, J.S. Bradbury, J. Dunfield

Course Summary

- Instructor: Dr. Joshua Dunfield
  - http://www.cs.queensu.ca/~joshuad/
  - joshuad@cs.queensu.ca
- Lectures
  - Mon. 15:30–16:30 WLH 205
  - Thu. 16:30–17:30 WLH 205
  - Fri. 10:30-11:30 **Dupuis Auditorium**
- Office Hours (Goodwin 534)
  - TBA + by appointment + when door is wide open

Course Summary

- TAs / Advisors / Markers:
  - _, @
  - _, @
  - _, @
  - _, @
- Assignment Advising
  - TBA

Course Summary

- Textbook
  - Lectures and web references
    - CISC 327 Course Readings **2018** (**or 2014 / 2015 / 2016 / 2017**)
    - Readings form part of the course material, even if not covered in class
- Webpage
  - http://www.cs.queensu.ca/~joshuad/327
  - Easier to remember: http://dunfieldlab.ca/327
  - Frequent updates as course progresses
- onQ will be used **minimally**

Course Summary

- Goal of the course
  - Concepts, theory, and practice of software quality assurance through testing, inspection, and measurement of software systems
- Course project
  - An opportunity to put what you learn into practice. You will form small software "companies" and design and build a small high quality software product
- Quality stories
  - Includes reviews of real applications of QA

Course Summary

- Topics

Introduction (1 week)	Software quality definitions
Process (2 weeks)	Software process models
Testing (3 weeks)	Testing methods and roles
Inspection (3 weeks)	Inspections and reviews
Measurement (2 weeks)	Software metrics
Safety and Security (1 week)	Software security

Course Summary

- eXtreme Programming (XP)
  - A controversial but influential member of the agile software development family based on iterative and incremental development
  - Traditional software quality methods can be boring and dated, so we will use XP as a theme of the course, coupled with real-world examples
  - As much as possible, the course project will be carried out using the principles and methods of XP

Course Summary

- Marking

4	In-Class Mini-Exams @ ~12.5% each	50%
~6	Project Assignments @ ~8% each	50%

  - The Mini-Exams are equivalent to a final exam
  - Your mark in the course is bounded by your personal combined Mini-Exams mark
  - **You must pass the combined Mini-Exams to pass the course**
  - Some project marks will be based on peer assessment

Who is this person?

- Joshua Dunfield
  - Programming languages researcher
    - PhD, Carnegie Mellon (2007)
    - Postdoc at McGill (2007–2010)
    - Postdoc at Max Planck Institute for Software Systems (2010–2014)
    - Research associate* at UBC (2014–2017)
    - Assistant professor, Queen's (2017–)

** don't worry, UBC didn't know what that was either*

Who designed most of this course?

- James Cordy
  - 8 years as professional software developer, QA officer
    - Chief programmer on 5-year, $5M Euclid project
    - Co-designer and chief programmer of the Turing programming language & compiler
  - 6 years as VP and CTO of Legasys Corporation
    - Y2K analysis & solution, licensed to IBM Global Services
    - VP Research & Development, solution designer, QA officer
  - 2 years at U. Toronto (Eng. Sci.) and 30+ years at Queen's as Prof. of Software Engineering
    - Distinguished scientist, ACM; Senior member IEEE; IBM faculty fellow; Grand professor, T.U. Dresden, P.Eng. (SWE)
  - Professor Emeritus since 2018

Why am I teaching a SE course?

- *Operational reason:* QSC needs me to
- *Principled reason:*
  Software engineering is *adjacent*
  to programming languages
- Consequences:
  - I'll only be a few steps ahead of you
  - I won't think stuff is obvious because I've known it for years (except when I veer off into programming languages)
- *this slide is from 2017, but still mostly true*

What am I doing differently?

- I can't share stories from industry
  - I'll get Jim to guest-lecture
- Some new material around the intersection of programming languages and software engineering:
  - Static analysis
  - Formal verification
  - **Security**

Software Quality

- Software Engineering
  - "The application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software."
    - IEEE
- Quality
  - Not a single idea - many aspects

Software Quality

- Popular View
  - In everyday life, usually thought of as intangible, can be felt or judged, but not weighed or measured
  - "I know it when I see it" - implies that it cannot be controlled, managed, or quantified
  - Often influenced by perception rather than fact
    - For example, a Cadillac may be spoken of as a "quality" car, in spite of the fact that its reliability and repair record is much the same as a Chevrolet

Software Quality

- Professional View
  - In a profession such as software development, there is an ethical imperative to quality
  - Quality is not just a marketing and perception issue, it is a moral and legal requirement - we have a professional responsibility associated with the software we create
  - Professionals must be able to demonstrate, and have confidence, that they are using "best practices"

Software Quality

- Professional View
  - In practical terms, therefore, product quality must be measurable in some way
  - Product quality is spoken of in terms of
    - conformance to requirements, including timeliness and cost
    - fitness for use, does it actually do the job?
    - freedom from errors and failures, is it reliable and robust?
    - customer satisfaction, are users happy with it?

Software Quality

- Why does it matter?
  - McConnell's "Code Complete" suggests an industry average of 15–50 errors per 1,000 lines of delivered code
  - Thousands of bugs in commercial software
  - If we still use the software, what harm is there in a few errors?

Software Quality

- Therac-25
  - Radiation therapy machine
  - Patients were given massive overdoses of radiation
    - Approximately 100 times the intended dose
  - Primarily blamed on bad software design and development practices
    - Software designed so that it was realistically impossible to automate testing

Software Quality

- Software Quality
  - Software quality is normally spoken of in terms of several different dimensions called quality parameters, roughly split into two groups
    - Technical Quality Parameters
    - User Quality Parameters

Software Quality

- Technical Quality Parameters
  - Correctness, reliability, capability, performance, maintainability
  - These are open to objective measures and technical solutions (focus of this course)
- User Quality Parameters
  - Usability, installability, documentation, availability
  - These often require subjective analysis and non-technical solutions

Technical Quality Parameters

- Correctness: lack of bugs and defects
  - measured in terms of defect rate
    (# bugs per line of code)
- Reliability: does not fail or crash often
  - in terms of failure rate (# failures per hour)
- Capability: does all that is required
  - in terms of requirements coverage
    (% of required operations implemented)

Technical Quality Parameters

- Maintainability: easy to change and adapt to new requirements
  - in terms of change logs (time and effort to add a new feature) and impact analysis (# lines affected by a new feature)
- Performance: fast and small enough
  - in terms of speed and space usage (seconds of CPU time, MB of memory, etc.)

User Quality Parameters

- Usability: sufficiently convenient for the intended users
  - in terms of user satisfaction (% of users happy with interface and ease of use)
- Installability: convenient and fast to install
  - in terms of user satisfaction (# install problems reported per installation)

User Quality Parameters

- Documentation:
  well documented
  - in terms of user satisfaction
    (% of users happy with documentation)
- Availability:
  easy to access and available when needed
  - in terms of user satisfaction
    (% of users reporting access problems)

Customers vs. users

- Pre-internet era: *customers* = *users*
- Internet era:
  If someone pays, they are the customer
- Consequence:
  - Facebook's users are **not** Facebook's customers
  - "Customer satisfaction"?

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 should be 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