Introduction I

Overview

- We will discuss the course content, structure and marking scheme
- We will learn about software engineering, software quality and software quality assurance
- We will also consider what methods we can use to achieve quality software

CSCI 3060U

Software Quality Assurance

- **Instructor:** Michael Miljanovic
  Office hours: Monday 3:30pm-4:30pm (UA2029), otherwise by appointment
- **Teaching Assistants:** Jude Arokiam, Hrim Mehta, Baskare Askari
- **Lectures:**

  Tues. 2:10pm-3:30pm (J102),
  Fri. 2:10pm-3:30pm (J102)

CSCI 3060U

Software Quality Assurance

- **Laboratories:**

  Mon. 12:40pm – 2:00pm (UA2230),
  Mon. 9:40am – 11:00am (ERC1092),
  Tues. 12:40pm – 2:00pm (UA2230),
  Weds. 2:10pm-3:30pm (ERC1092)

CSCI 3060U

Software Quality Assurance

- Textbook:
  - No required textbook
  - We will be using online resources

CSCI 3060U

Software Quality Assurance

- Aims of the Course:
  - The primary purpose of this course is to build on what has been learned about software engineering in previous courses. The course will provide details on topics aimed at the development of high quality software systems and allow students to practice what they have learned through a group project.

CSCI 3060U

Software Quality Assurance

- Topics:
  - ***Introduction*** *(0.5 weeks)*
    - Introduction to Software Engineering
    - Software Quality - what is it, how is it measured, how is it achieved
  - ***Software Process*** *(2 weeks)*
    - Software Process Models - plans for achieving and improving software quality
    - eXtreme Programming - a controversial modern software process

CSCI 3060U

Software Quality Assurance

- Topics:
  - ***Software Testing*** *(5 weeks)*
    - Systematic Testing - what is it, levels of testing, designing for test
    - Black Box Testing - functional, input, output, partitioning and gray box testing
    - White Box Testing - coverage, path, decision and mutation testing
    - Continuous Testing - regression, defect testing
    - Test Automation - test maintenance and analysis, harnesses, tracking, tools

CSCI 3060U

Software Quality Assurance

- Topics:
  - ***Software Inspection*** *(1 week)*
    - Systematic Inspection - what is it, levels of inspection, inspection process, formal reviews
    - Inspection in the Software Process - requirements, design, process and code inspections
    - Code Inspection - techniques, practices, continuous inspection, refactoring

CSCI 3060U

Software Quality Assurance

- Topics:
  - ***Software Metrics*** *(0.5 weeks)*
    - Software Metrics - measurement basics, assessment and prediction
    - Product Quality Metrics, Process Metrics, etc.
  - ***Alternative Verification and Validation Techniques*** *(0.5 weeks)*
    - Dynamic analysis, static analysis, formal methods.

CSCI 3060U

Software Quality Assurance

- Quality Assurance Tools and Case Studies
  - Throughout the course we will introduce new quality assurance tools and case studies of real software bugs

CSCI 3060U

Software Quality Assurance

- Marking:

Course Project	**40%**
Term Tests (3)	**24%**
Final Exam	**36%**

Notes:

- *Peer evaluations of team members will be conducted and can contribute to the final project mark.*
- *In order to pass the course you **must** pass the individual work (i.e., the 60% of the mark consisting of the tests and the final exam).*

More Information?

- Course website: http://www.sqrlab.ca/csci3060u/

More Information?

- Course website: http://www.sqrlab.ca/csci3060u/

Contacting Your Professor/TA

- Slack: https://csci3060u-w20.slack.com/

Contacting Your Professor/TA

- Slack: https://csci3060u-w20.slack.com/

Used for discussions, questions & answers and course announcements. If you need to contact your professor or TA – try here first!

CSCI 3060U

Software Quality Assurance

- Agile Methods – eXtreme Programming (XP)
  - Agile development methods are relatively new and exciting ways for producing quality software. Since the traditional software quality methods can be dry, boring and dated, we will use one agile development approach (XP) as a theme of the course.
- Course Project
  - As much as possible, the course project will be carried out using agile principles and methods.

What is Software Engineering?

"the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software"

-IEEE

What Causes Software Errors?

1. Faulty definition of requirements
   - Erroneous requirement definitions
   - Absence of important requirements
   - Incomplete requirements
   - Unnecessary requirements included
2. Client-developer communication failures
   - Misunderstanding of client requirements presented in writing, orally, etc.
   - Misunderstanding of client responses to design problems

What Causes Software Errors?

3. Deliberate deviations from software requirements
   - Reuse of existing software components from previous projects without complete analysis
   - Functionality omitted due to budge or time constraints
   - “Improvements” to software that are not in requirements
4. Logical design errors
   - Errors in interpreting the requirements into a design (e.g. errors in definitions of boundary conditions, algorithms, reactions to illegal operations,...)
5. Coding errors
   - Errors in interpreting the design document, errors related to incorrect use of programming language constructs, etc.

What Causes Software Errors?

6. Non-compliance with documentation and coding instructions
   - Errors resulting from other team members coordinating with non- complying member's code
   - Errors resulting from individuals trying to understand/maintain/test non-complying member's code
7. Shortcomings of the testing process
   - Incomplete test plan
   - Failure to report all errors/faults resulting from testing
   - Incorrect reporting of errors/faults
   - Incomplete correction of detected errors

What Causes Software Errors?

8. Procedural errors
   - Incorrect procedures related to user activities that occur in the software
9. Documentation errors
   - Errors in the design documents or code comments
   - Errors in user manuals for software

Quality – What is it?

Quality Not a single idea - many aspects

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 - e.g., a Cadillac may be spoken of as a "quality" car, in spite of the fact that its reliability and repair record is no better than a Chevrolet

Quality – What is it?

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 to have confidence, that they are using "best practices"
- 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, 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?

So What is Software Quality?

Software Quality

- The degree to which a system, component, or process meets specified requirements

or

- The degree to which a system, component or process meets customer or user needs or expectations.

?

So What is Software Quality?

Software Quality is: [IEEE Definition]

- The degree to which a system, component, or process meets specified requirements
  [based on Philip B. Crosby's definition, 1979]

- The degree to which a system, component or process meets customer or user needs or expectations.
  [based on Joseph M. Juran, 1988]

So What is Software Quality?

Software Quality

- Software quality is normally spoken of in terms of several different dimensions often called quality parameters
- These can be split (roughly) into two groups

  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 nontechnical solutions

Software Quality

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
  - measured in terms of failure rate (#failures per hour)
- Capability - does all that is required
  - measured in terms of requirements coverage (% of required operations implemented)
- Maintainability - is easy to change and adapt to new requirements
  - measured in terms of change logs (time and effort required to add a new feature) and impact analysis (#lines affected by a new feature)
- Performance - is fast and small enough
  - measured in terms of speed and space usage (seconds of CPU time, Mb of memory, etc.)

Software Quality

User Quality Parameters

- Usability - is sufficiently convenient for the intended users
  - measured in terms of user satisfaction (% of users happy with interface and ease of use)
- Installability - is convenient and fast to install
  - measured in terms of user satisfaction (#install problems reported per installation)
- Documentation - is well documented
  - measured in terms of user satisfaction (% of users happy with documentation)
- Availability - is easy to access and available when needed
  - measured in terms of user satisfaction (% of users reporting access problems)

Classification of Software Quality Factors

McCall's Factor Model

- Classifies all software requirements into 11 factors.
- The factors are grouped in 3 categories:

1. Product operation factors

   Factors that deal with requirements that directly affect the daily operation of the software.

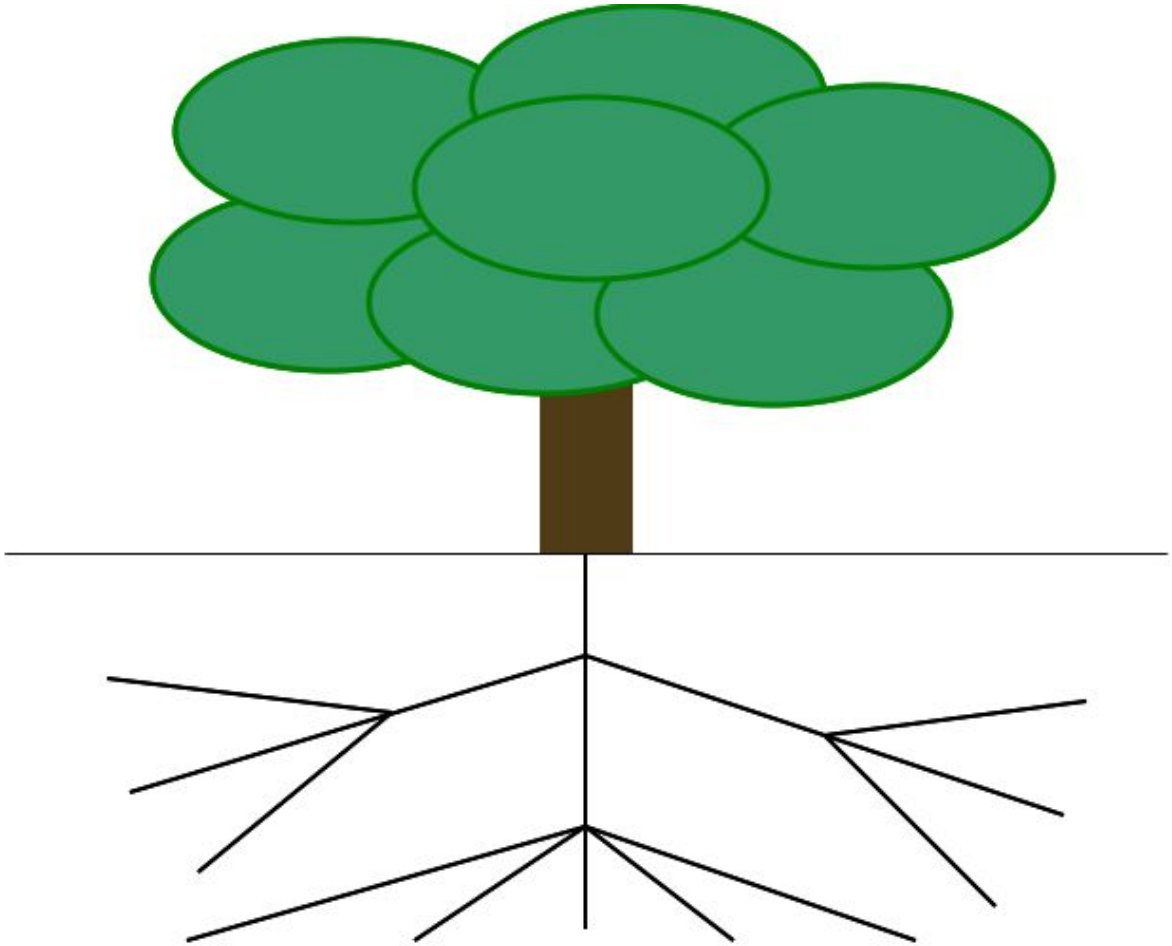
2. Product revision factors

   Factors that deal with requirements that affect software maintenance.

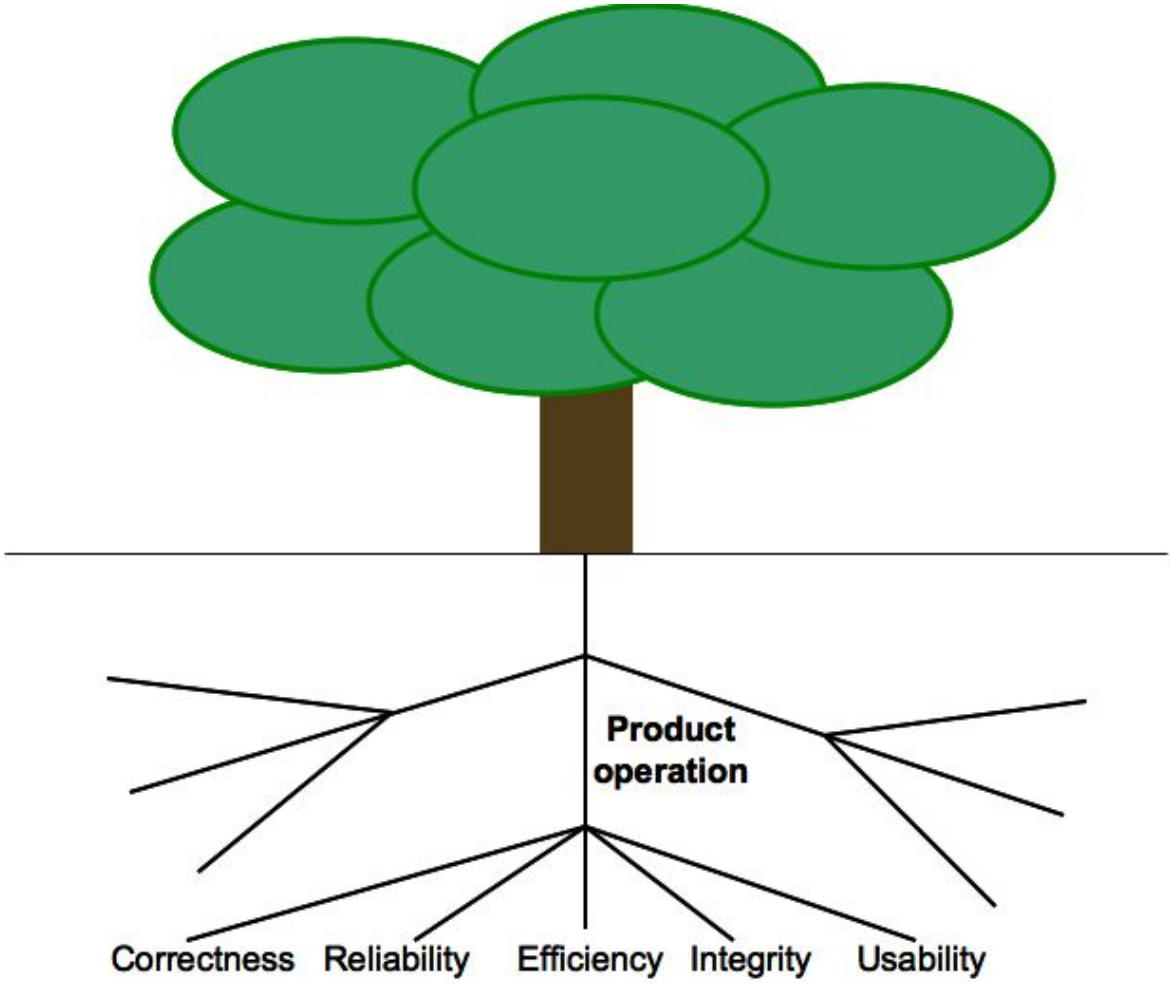
3. Product transition factors

   Factors that deal with requirements that affect the adaptation of software to other platforms, environments, interaction with other software.

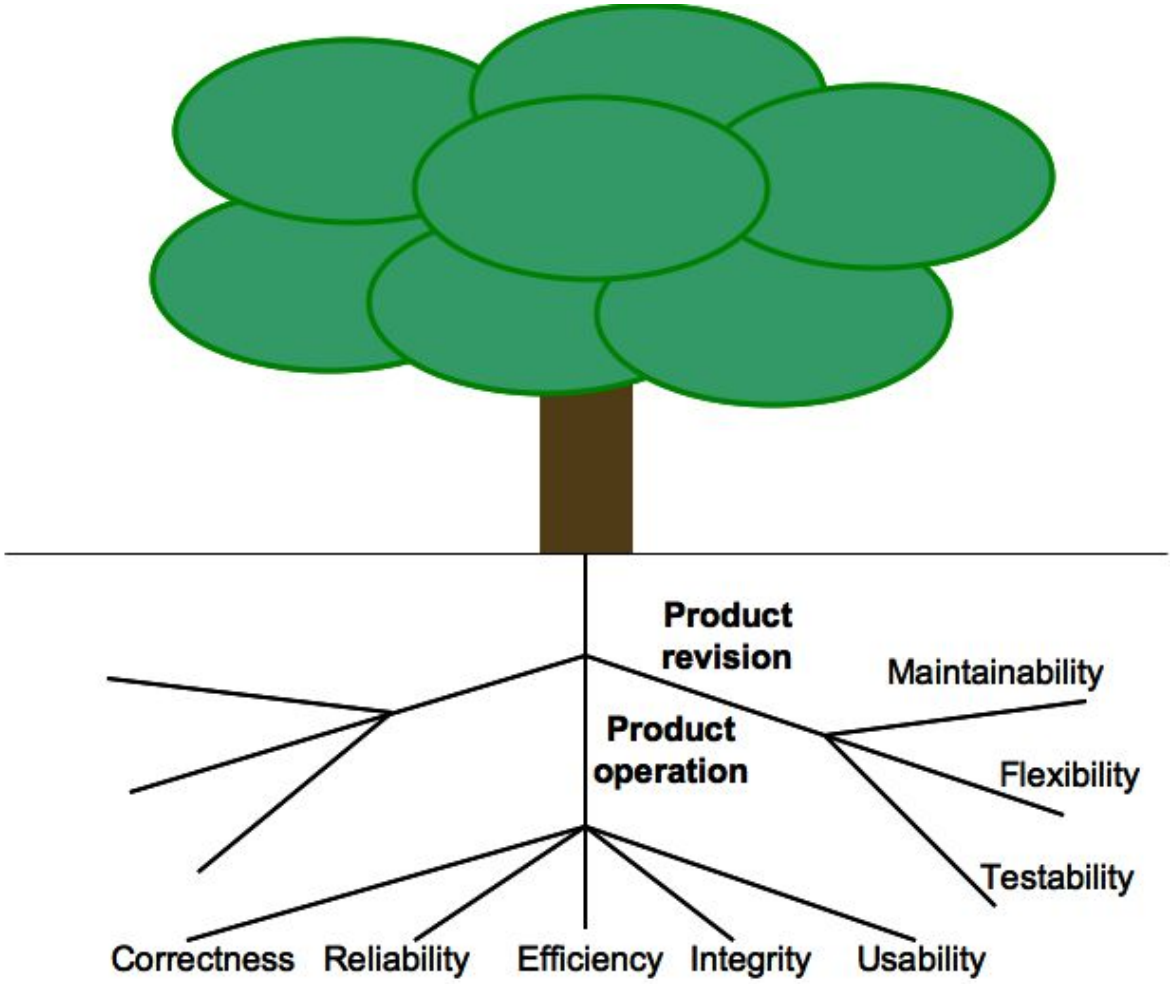
McCall's Factor Model Tree



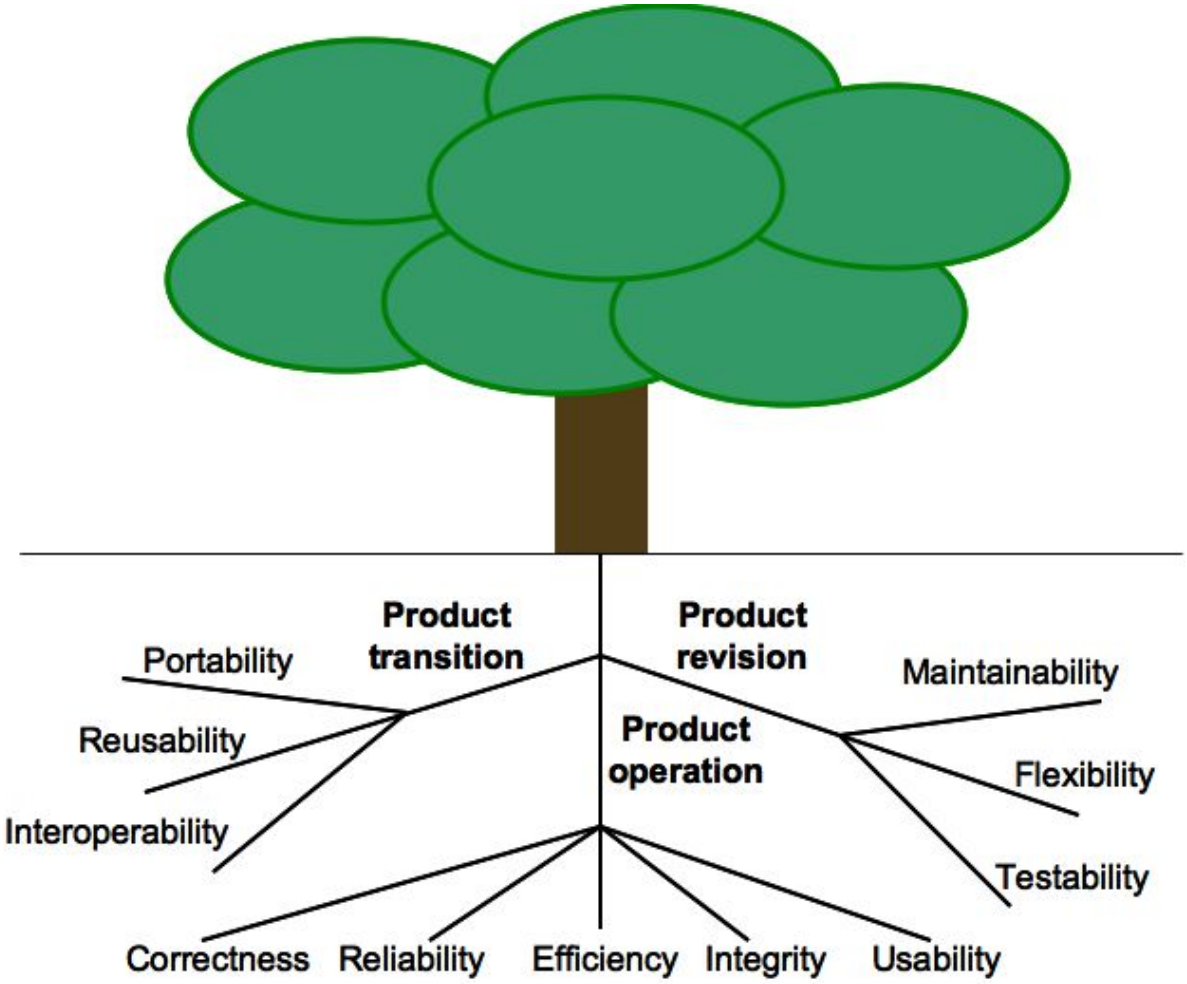
McCall's Factor Model Tree



McCall's Factor Model Tree



McCall's Factor Model Tree



Quality Assurance – What is it?

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 – What is it?

Three General Principles of QA

- Know what you are doing
- Know what you should be doing
- 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 - 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

Formal Methods

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
- Traditionally, use of formal methods requires mathematically sophisticated programmers, and is necessarily a slow and careful process, and very expensive
- In the past, formal methods have been used directly in software quality assurance in a small (but important) fraction of systems
  - Primarily safety critical systems such as onboard flight control systems, nuclear reactor control systems, embedded systems such as automobile braking systems and medical equipment, and so on…however, today formal methods is becoming more widely used (as we will see later).

Testing

Focus of the Course

- The vast majority (over 99%) of software quality assurance uses testing, inspection and metrics instead of formal methods
- Example: at the Bank of Nova Scotia, over 80% of the total software development effort is involved in testing!

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

Inspection and Metrics

Inspection

- Inspection includes methods based on a human 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

Metrics

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 are many software processes proposed, of which eXtreme Programming is one of the more recent

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
- Examples are Total Quality Management (TQM),
  the Capability Maturity Model (CMM),
  the Personal Software Process (PSP),
  the Microsoft Shared Development Process (SDP),
  the Rational Unified Process (RUP),
  and ISO 9000

Introduction I

Summary

- We've seen what software engineering means
- We've seen what quality means, how it applies to software, and what methods we can use to achieve it

References

- Kan, *Metrics and Models in Software Quality Engineering, ch.1*
- *Galin, Software Quality Assurance: From Theory to Implementation, ch 2 & 3*

Next time

- We will begin by covering the software development process, explore a number of software process models, and see how the software life cycle can affect software quality