# Process

* Quality assurance process
  + Unlike manufacturing, which has a repetitive phase, in software development quality of the result is controlled by the process used to develop the software
  + QA is process oriented
* Quality control
  + Product oriented
  + Testing for quality is assuring quality it's controlling it
* Techniques
  + Change-control procedures: One big obstacle to achieving software quality is uncontrolled changes
  + Measurement of results:  Unless you measure the results of your QA plan, you won't know if it's working
* Setting Objectives
  + One powerful technique for improving software quality is to set explicit quality objectives from the internal and external characteristics
* When to do QA
  + A single architectural error can affect several classes and many routines while a single construction error is unlikely to affect more than one routine or class
* Costs:  The longer a defect remains in the system, the more expensive it becomes to remove

# What is quality?

* One of the main goals of software development is to produce high-quality software
* Quality - meeting the specification and fit to use
* Characteristics
  + Reliability is the probability of not failing in a specified length of time. Software reliability is a measure of how often the software encounters a data input or other condition that it does not process correctly to produce the right answer.  Software reliability is not concerned with software wearing out
  + Integrity- degree to which a system prevents unauthorized or improper access to its programs and data
  + Accuracy is different from correctness; it is a determination of how well a system does the job it's built for rather than whether it was built correctly
  + Robustness- how well a system functions in the presence of invalid inputs or stressful environmental condition
* General quality principle
  + Debugging and associated refactoring and other rework consume about 50 percent of the time on a traditional software development cycle

# Validation vs. Verification

* Testing is only part of a broader process of software verification and validation.
* The distinction between the two terms is largely to do with the role of specifications.
* **validation** is the process of checking whether the specification captures the customer's needs
* **verification** is the process of checking that the software meets the specification
* Definitions from Barry Boehm, a software engineering pioneer
  + Verification:  checking that software meets its stated functional and non-functional requirements
    - Meeting requirements
      * There is almost always conflicting requirement from different stakeholder groups that result in a compromise that is not pleasing to both groups
      * It is impossible to measure certain quality characteristics such as maintainability
  + Validation:  more general – aims to ensure that software does what the customer expects it to do

# Faults and failures

* Error: a mistake made by a programmer or software engineer which caused a fault, which in turn may cause a failure
* Fault : condition that may cause a failure in the system
  + defect or bug
* Failure: inability of system to perform a function according to its spec due to some fault

# Error detection techniques

* Testing is usually the most important activity in quality control
  + Software testing is concerned with exercising and observing product behavior
  + dynamic verification
  + Developers tend to rely on testing as the primary method of both quality assessment and improvement.
    - Includes Unit test, Component test, Integration test, Regression test, System test, Beta test
* Inspections and reviews
  + Informal technical reviews:  Desk checking the design or the code or walking through code with a few peers
  + Inspections and software reviews
    - Goal is to find errors
    - Software inspections are concerned with analysis of   
      the static system representation to discover problems
  + Static verification
* Formal methods are mathematical techniques used to "prove" that a program is correct
* Static analysis involves analyzing the static structure of the program
* Use more than 1 technique
  + Major organizations use a wider variety of techniques and achieve high defect-removal efficiencies
  + Defect detection methods work better in combination than they do singly
  + Human processes are better than computer-based testing to find certain types of errors and the opposite is true for other types of errors

# Measuring Quality

* QA metrics:
  + Example:  size of product in lines of codes, number of reported faults in a delivered software product, number of person-days required to develop a system
  + Dynamic metrics:  Usually a clear relation between dynamic metrics and software quality
  + Example of static metrics:   fan-in/fan-out, length of code, length of identifiers, depth of conditional nesting
  + Static metrics usually have an indirect relationship with quality
* Measurement ambiguity
  + Number of change requests could be because the product is buggy or it could be because the product is high-quality and users want more features

# Testing

* Dynamic verification means while the program is running
* Testing cannot prove that a product completely works
  + We can demonstrate that parts of the software works

# Stages of testing

* Development testing involves system designers and programmers
  + Discover bugs and defects.
* Release testing: A separate testing team test a complete version of the system before it is released to users.
  + Goal of release testing is to make sure that the system meets the requirements of the system stakeholders
* User testing: Users or potential users of a system test the msystem in their own environment.
  + Acceptance testing is type of user testing where the customer formally tests a system to decide if it should be accepted from the system supplier or if requires further work

# Development testing

* Unit testing: Individual program units or object classes are tested.
* Component testing: Several individual units are integrated to create composite components.
* System testing: Some or all of the components in a system are integrated and the system is tested as a whole.

# Limitations of developer testing

* Clean tests test for the way code works
* Dirty tests test for the way code breaks

# Unit testing

* Testing is expensive and time consuming, so it is important that you choose effective unit test cases.

# Automated Testing

* Whenever possible, unit testing should be automated so that tests are run and checked without manual intervention.
* In automated unit testing, you make use of a test automation framework (such as JUnit) to write and run your program tests.
* Unit testing frameworks provide generic test classes that you extend to create specific test cases.

# Automated test components

* An automated test has 3 parts

# Unit test effectiveness

* Unit testing should
  + demonstrate that when used as expected, the component that you are testing does what it is supposed to do.
  + reveal defects in components, if there are any
* Testing is expensive and time consuming, so it is important that you choose effective test cases
* This leads to 2 types of unit test case:
  + Those that reflect normal operation of a program and show that the component works as expected.
  + Those based on testing experience of where common problems arise.

# Black box testing

* Examples:
  + Partition testing
  + Guideline-based testing
    - Use testing guidelines that reflect previous experience of the kinds of errors that programmers often make when developing components to choose test cases.

# Equivalence partitions

* Identify groups of inputs that have common characteristics and should be processed in the same way.
* Each of these classes is an equivalence partition or domain where the program behaves in an equivalent way for each class member.
* Test cases should be chosen from each partition.
* Example: Example: pick larger of two integers
  + First > Second: 10,7
  + Second > First: 8,12
  + First = second: 36, 36
* Equivalence classes determined by requirements and some intuition

# Partition Testing Example

* specification states that the program accepts 4-8 inputs which are 5-digit integers > 10,000

# Boundary values

* Past experiences show that boundaries are error-prone
* Only works with ordinal values
* The basic boundary value testing for a value would include:
  + at the minimum boundary
  + immediately above minimum
  + between minimum and maximum (nominal)
  + immediately below maximum
  + at the maximum boundary

# Path testing

* The objective of path testing is to ensure that the set of test cases is such that each path through the program is executed at least once.
* The starting point for path testing is a program flow graph that shows nodes representing program decisions and arcs representing the flow of control.
* Testing
  + Analyze number of paths in program
  + Decide which ones to test
  + Decreasing coverage:
    - Logical paths
    - Independent paths
    - Branch coverage
    - Statement coverage

# Component Testing

* Software components are made up of several interacting objects.
  + You access the functionality of these objects through the interfaces.
* You focus on interfaces because you can assume that unit tests on the individual objects within the component have been completed.

# Interface testing

* Interface types
  + Parameter interface: data passed from one method or procedure to another
  + Shared memory interface: block of memory is shared between procedures or functions.
  + Procedural interface: sub-system encapsulates a set of procedures to be called by other sub-systems.
  + Message passing interface: sub-systems request services from other sub-systems  
    Interface Errors
* These are one of the most common forms of error in complex systems
* Example of interface misuse:  parameters in the wrong order.
* Example of interface misunderstanding:  passing an unsorted array to a binary search method

# Interface Testing Guidelines

* Inspections can sometimes be more cost effective than testing for and discovering interface errors.
* Strong-typed languages (like Java) can detect a wide range of interface errors

# System Testing

* System testing involves integrating components to create a version of the system and then testing the integrated system.
* Focus is on testing the interactions between components.
* System testing checks that components are compatible and interact correctly.
* System testing tests the emergent behavior of a system.
* System testing checks that components are compatible, interact correctly and transfer the right data at the right time across their interfaces.

# Integration Testing Strategies

* For the following integration testing strategies, we use this call hierarchy of an example design consisting of 3 layers and 7 subsystems.

# Big bang approach

* This unit tests each of the subsystems, and then does one gigantic integration test, in which all the subsystems are immediately tested together.
* Don‘t try this!! Why: The interfaces of each of the subsystems have not been tested yet.

# Sandwich testing

* How do you select the target layer if there are more than 3 layers?
* Heuristic: Try to minimize the number of stubs and drivers

# Use case testing

* The use-cases can be used as a basis for system testing.
* Each use case usually involves several system components so testing the use case forces these interactions to occur.
* The sequence diagrams associated with the use case documents the components and interactions that are being tested.

# Release testing

* Release testing has to show that the system delivers its specified functionality, performance and dependability, and that it does not fail during normal use.
* Release testing is usually a black-box testing process where tests are only derived from the system specification.

# Release testing vs. system testing

* A separate team that has not been involved in the system development, should be responsible for release testing.
* System testing by the development team should focus on discovering bugs in the system (defect testing).
* The objective of release testing is to check that the system meets its requirements and is good enough for external use (validation testing).

# Performance testing

* Part of release testing may involve testing the emergent properties of a system, such as performance and reliability.
* Tests should reflect the profile of use of the system.
* Performance tests usually involve planning a series of tests where the load is steadily increased until the system performance becomes unacceptable.
* Stress testing is a form of performance testing where the system is deliberately overloaded to test its failure behavior.

# User Testing

* User or customer testing is a stage in the testing process in which users or customers provide input and advice on system testing.
* User testing is important, even when comprehensive system and release testing have been carried out because influences from the user’s working environment have a major effect on the reliability, performance, usability and robustness of a system. These cannot be replicated in a testing environment.
* Types of user testing
  + Alpha testing: Users of the software work with the development team to test the software at the developer's site.
  + Beta testing: A release of the software is made available to users to allow them to experiment and to raise problems that they discover with the system developers.
  + Acceptance testing: Customers test a system to decide whether or not it is ready to be accepted from the system developers and deployed in the customer environment. Primarily for custom systems.

# When to stop testing

* Simple answer, stop when
  + All planned test cases are executed
  + All those problems that are found are fixed
* Other techniques:
  + Stop when you are not finding any more errors
  + Defect seeding -- test until all (or % of )the seeded bugs found
* NOT -- when you ran out of time -- poor planning!

# Inspections and Testing

* Mostly focus on source code but any readable representation can be inspected(requirements, design, configuration data, test data, etc.)

# Formal Inspections

* Involve examining the software, its documentation and records of the process to discover errors and omissions and to see if quality standards have been followed.
* Characteristics of formal inspections
  + Peers:  at least three people are involved
* Effectiveness:  Defect detection rate of about 25% for unit testing versus 60% for inspections.
* Rarely use formal inspections or peer review processes in agile development.
  + Pair programming is seen as an effective substitute

# Inspection steps

* During reviews, a group of people examine the software and its associated documentation, looking for potential problems and non-conformance with standards.
* The review team makes informed judgments about the quality of the system or project deliverable
* Follow-up:  If it satisfies the exit criteria, the inspection is completed.  Otherwise the producer can rework the product or a re-inspection can be scheduled

# Advantages of inspections

* During testing, errors can mask (hide) other errors
  + Because inspection is a static process, you don’t have to be concerned with interactions between errors.
* Incomplete versions of a system can be inspected without additional costs.
  + If a program is incomplete, then you need to develop specialized test harnesses to test the parts that are available.
* An inspection can also consider broader quality attributes of a program, such as compliance with standards, portability and maintainability.

# Inspections and testing

* Inspection is a static process versus testing which is dynamic
* Inspections cannot check non-functional characteristics such as performance, usability, etc.