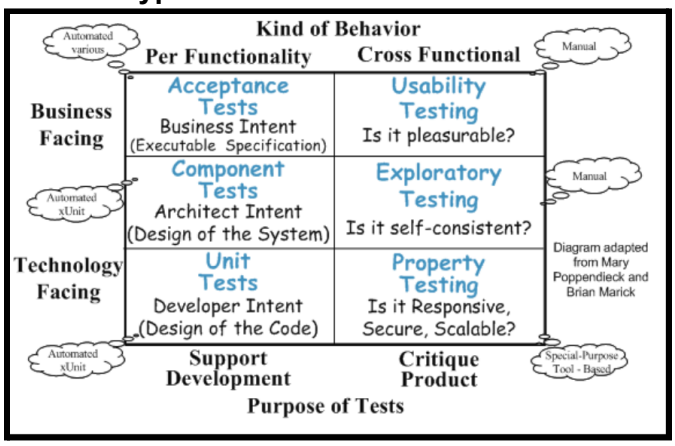
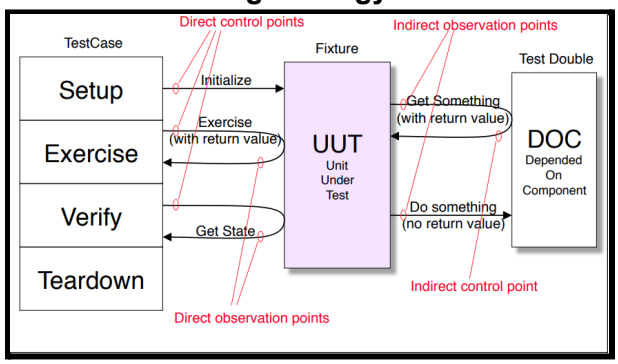
Automated Testing:

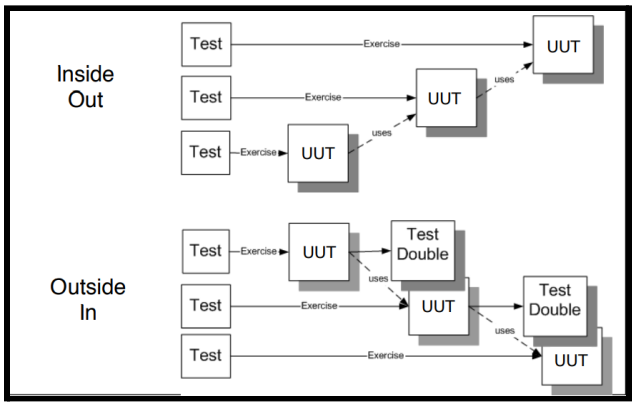
- **Different types of tests:**



- Where possible, automate your testing. By doing regression testing, tests can be repeated whenever the code is modified. This takes the tedium out of extensive testing and makes more extensive testing possible.
- In order to do automated testing, you'll need:
  - **Test drivers** which will automate the process of running a test set. It sets up the environment, makes a series of calls to the Unit-Under-Test (UUT), saves the results and checks if they were right and generates a summary for the developer.
  - **Test stubs** which will simulate part of the program called by the UUT. It checks whether the UUT set up the environment correctly, checks whether the UUT passed sensible input parameters to the stub and passes back some return values to the UUT.
- **Automated Testing Strategy:**

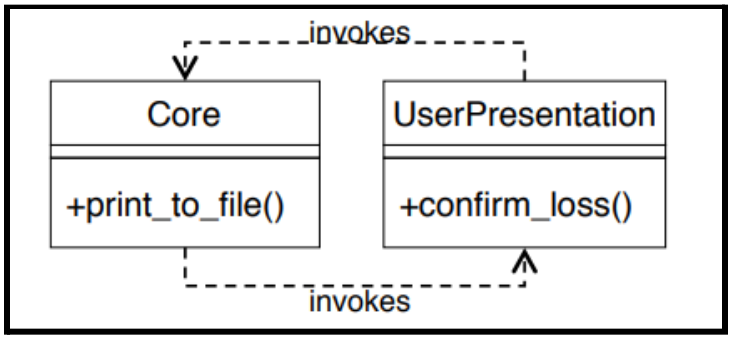


- Test Order

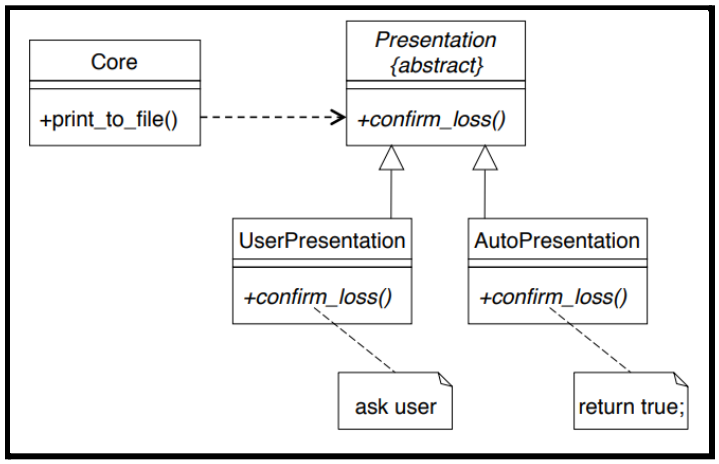


- **JUnit:**
- JUnit is a unit testing framework for Java.
- Assertion methods in JUnit:
    - **Single-Outcome Assertions:**
      E.g. fail;
    - **Stated Outcome Assertions:**
      E.g. assertNotNull(anObjectReference);
      E.g. assertTrue(booleanExpression);
    - **Expected Exception Assertions:**
      E.g. assert_raises(expectedError) {codeToExecute };
    - **Equality Assertions:**
      E.g. assertEqual(expected, actual);
    - **Fuzzy Equality Assertions:**
      E.g. assertEqual(expected, actual, tolerance);
- **Principles of Automated Testing:**
    - Write the test cases first.
    - Design for testability.
    - Use the front door first. This means test using public interfaces and avoid creating backdoor manipulations.
    - Communicate intent. Treat tests as documentation and make it clear what each test does.
    - Don't modify the UUT. Avoid test doubles and test-specific subclasses unless absolutely necessary.
    - Keep tests independent.
    - Isolate the UUT.
    - Minimize rest overlap.
    - Check one condition per test.
    - Test different concerns separately.
    - Minimize untestable code.
    - Keep test logic out of production code.
- **Challenges for automated testing:**
    - **Synchronization** - How do we know a window popped open that we can click in?
    - **Abstraction** - How do we know it's the right window?
    - **Portability** - What happens on a display with different resolution/size?
- **Techniques for testing the presentation layer:**
    - **Script the mouse and keyboard events:**
        - We can write a script that sends mouse and keyboard events. (E.g. "send_xevents @400,100")
        - However, this is not good practice/design because the script is write-only and fragile.
    - **Script at the application function level:**
        - E.g. Applescript: tell application "UMLet" to activate.
        - This is robust against size and position changes but fragile against widget renamings, and layout changes. Hence, this is still not good practice/design.
    - **Write an API for your application:**
        - We can use these APIs for testing.
        - E.g. Allow an automated test to create a window and interact with widgets.

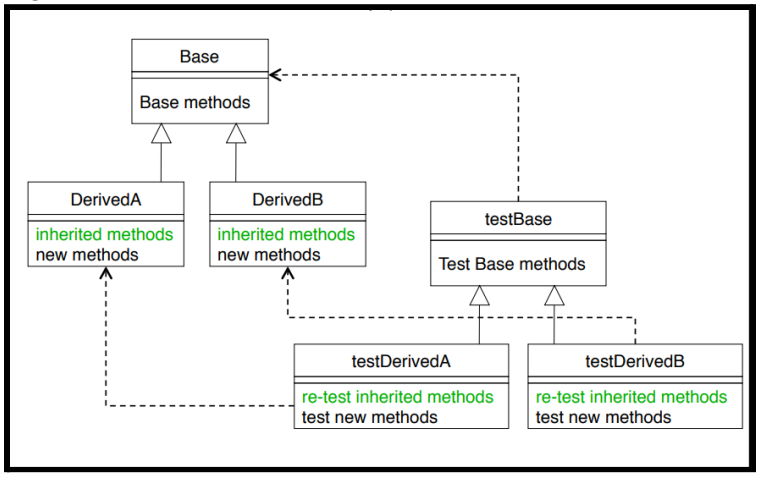
- **Circular Dependencies:**
- If you have circular dependencies in your code, you should refactor your code to remove them.
- E.g.
  Here, we have a circular dependency.



Because we have a circular dependency, we need to refactor the code. Here is the new code.



- **Testing Object Oriented Code:**
- Object oriented code can be hard to test. The best/most efficient way to test object oriented code is to have a parent test class for the parent class and to extend the parent test class for the subclasses.
  E.g.



E.g.
The test class for the parent class.

```
class FooTest {
        @Test
        public void testSomeMethodBar() {
         ...
        }

        @Test public void void someOtherMethodBaz(Baz baz) {
        ...
        }
}
```

The test class for the subclass class. Notice that this test class inherits from FooTest.

```
class EnhancedFooTest extends FooTest {
        @Test
        public void testSomeMethodBar() {
         ...
        }
}
```

- **When to stop testing:**
  - **Motorola's Zero-failure** testing model predicts how much more testing is needed to establish a given reliability goal.
  - **Failures = $ae^{-b(t)}$** where "a" and "b" are constants and "t" is the testing time.
  - The **reliability estimation process** gives the number of further failure free hours of testing needed to establish the desired failure density.
    Note: If a failure is detected during this time, you stop the clock and recalculate.
    Note: This model ignores operational profiles.
    Inputs needed:
    - **fd** = target failure density (e.g. 0.03 failures per 1000 LOC)
    - **tf** = total test failures observed so far
    - **th** = total testing hours up to the last failure

    Formula:

$$\frac{ln(fd/(0.5+fd)) * xh}{ln((0.5+fd)/(tf+fd))}$$

- **Fault Seeding:**
- **Fault seeding** is a technique for evaluating the effectiveness of a testing process. One or more faults are deliberately introduced into a code base, without informing the testers. The discovery of seeded faults during testing can be used to calibrate the effectiveness of the test process.
- The idea is that

$$\frac{\text{Detected seeded faults}}{\text{Total seeded faults}} = \frac{\text{Detected nonseeded faults}}{\text{Total nonseeded faults}}$$

and we can use this data to estimate test efficiency and to estimate the number of remaining faults.

Acceptance Testing:

- **Introduction to Acceptance Testing:**
- **Acceptance testing** is testing conducted to determine whether a system satisfies its acceptance criteria.
- There are two categories of acceptance testing:
    1. **User Acceptance Testing (UAT):** It is conducted by the customer to ensure that the system satisfies the contractual acceptance criteria before being signed-off as meeting user needs. This is the final test performed. The main purpose of this testing is to validate the software against the business requirements. This validation is carried out by the end-users who are familiar with the business requirements.
    2. **Business Acceptance Testing (BAT):** It is undertaken within the development organization of the supplier to ensure that the system will eventually pass the user acceptance testing. This is to assess whether the product meets the business goals and purposes or not.
- **The 3 main goals of accepting testing are:**
    1. Confirm that the system meets the agreed upon criteria.
    2. Identify and resolve discrepancies, if there are any.
    3. Determine the readiness of the system for cut-over to live operations.
- **The acceptance criteria are defined on the basis of the following attributes:**
    1. Functional Correctness and Completeness
    2. Accuracy
    3. Data Integrity
    4. Data Conversion
    5. Backup and Recovery
    6. Competitive Edge
    7. Usability
    8. Performance
    9. Start-up Time
    10. Stress
    11. Reliability and Availability
    12. Maintainability and Serviceability
    13. Robustness
    14. Timeliness
    15. Confidentiality and Availability
    16. Compliance
    17. Installability and Upgradability
    18. Scalability
    19. Documentation
- **Selection of Acceptance Criteria:**
- The acceptance criteria discussed are usually too general, so the customer needs to select a subset of the quality attributes.
- The quality attributes are then prioritized to the specific situation.
- Ultimately, the acceptance criteria must be related to the business goals of the customer's organization.

- Example of an acceptance test plan

1. Introduction
2. Acceptance test category For each category of acceptance criteria (a) Operation environment (b) Test case specification (i) Test case Id# (ii) Test title (iii)Test objective (iv) Test procedure
3. Schedule
4. Human resources

- **Acceptance Test Execution:**
- The acceptance test cases are divided into two subgroups:
  - The first subgroup consists of basic test cases.
  - The second subgroup consists of test cases that are more complex to execute.
- The acceptance tests are executed in two phases:
  - In the first phase, the test cases from the basic test group are executed.
  - If the test results are satisfactory then the second phase, in which the complex test cases are executed, is taken up.
  - In addition to the basic test cases, a subset of the system-level test cases are executed by the acceptance test engineers to independently confirm the test results.
- Acceptance test execution activity includes the following detailed actions:
  - The developers train the customer on the usage of the system.
  - The developers and the customer coordinate the fixing of any problem discovered during acceptance testing.
  - The developers and the customer resolve the issues arising out of any acceptance criteria discrepancy.
- The acceptance test engineer may create an Acceptance Criteria Change (ACC) document to communicate the deficiency in the acceptance criteria to the supplier.
- An ACC report is generally given to the supplier's marketing department through the on-site system test engineers.
- E.g. of an ACC document

1. ACC Number:	A unique number
2. Acceptance Criteria Affected:	The existing acceptance criteria
3. Problem/Issue Description:	Brief description of the issue
4. Description of Change Required:	Description of the changes needed to be done to the original acceptance criterion
5. Secondary Technical Impacts:	Description of the impact it will have on the system
6. Customer Impacts:	What impact it will have on the end user
7. Change Recommended by:	Name of the acceptance test engineer(s)
8. Change Approved by:	Name of the approver(s) from both the parties

- **Acceptance Test Report:**
- The acceptance test activities are designed to reach at a conclusion:
  - Accept the system as delivered.
  - Accept the system after the requested modifications have been made.
  - Do not accept the system.
- Usually some useful intermediate decisions are made before making the final decision.
- A decision is made about the continuation of acceptance testing if the results of the first phase of acceptance testing is not promising. If the test results are unsatisfactory, changes are made to the system before acceptance testing can proceed to the next phase.
- During the execution of acceptance tests, the acceptance team prepares a test report on a daily basis.
  I.e. During the execution of acceptance tests, a daily acceptance test report is made.
- E.g. of a daily acceptance test report

1. Date:	Acceptance report date
2. Test case execution status:	Number of test cases executed today Number of test cases passing Number of test cases failing
3. Defect identifier:	Submitted defect number Brief description of the issue
4. ACC number(s):	Acceptance criteria change document number(s), if any
5. Cumulative test execution status:	Total number of test cases executed Total number of test cases passing Total number of test cases failing Total number of test cases not executed yet

- At the end of the first and the second phases of acceptance testing an acceptance test report is generated.
- E.g. of a finalized acceptance test report

1. Report identifier
2. Summary
3. Variances
4. Summary of results
5. Evaluation
6. Recommendations
7. Summary of activities
8. Approval

- **Acceptance Testing in Extreme Programming:**
- In the XP framework, the user stories are used as the acceptance criteria.
- The user stories are written by the customer as things that the system needs to do for them.
- Several acceptance tests are created to verify the user story has been correctly implemented.
- The customer is responsible for verifying the correctness of the acceptance tests and reviewing the test results.
- A story is incomplete until it passes its associated acceptance tests.
- Ideally, acceptance tests should be automated, either using the unit testing framework, before coding.
- The acceptance tests take on the role of regression tests.

Static Analysis:

- Static analysis is a method of computer program debugging that is done by examining the code without executing the program.
- This process provides an understanding of the code structure and can help ensure that the code adheres to industry standards.
- Automated tools can assist programmers and developers in carrying out static analysis. The software will scan all code in a project to check for vulnerabilities while validating the code.
- Static analysis is generally good at finding coding issues such as:
    - Programming errors
    - Coding standard violations
    - Undefined values
    - Syntax violations
    - Security vulnerabilities
- Once the code is written, a static code analyzer should be run to look over the code. It will check against defined coding rules from standards or custom predefined rules. Once the code is run through the static code analyzer, the analyzer will have identified whether or not the code complies with the set rules. It is sometimes possible for the software to flag false positives, so it is important for someone to go through and dismiss any. Once false positives are waived, developers can begin to fix any apparent mistakes, generally starting from the most critical ones. Once the code issues are resolved, the code can move on to testing through execution.
- Example of static analysis tools include:
    - FindBugs
    - JLint
    - JSHint
- Different tools find different bugs.
- Benefits of using static analysis include:
    - It can evaluate all the code in an application, increasing code quality.
    - Automated tools are less prone to human error and are faster.
    - It will increase the likelihood of finding vulnerabilities in the code, increasing web or application security.
    - It can be done in an offline development environment.
- Drawbacks of using static analysis include:
    - False positives can be detected.
    - It will detect harmless bugs that may not be worth fixing.
    - A tool might not indicate what the defect is if there is a defect in the code.
    - Static analysis can't detect how a function will execute.

Quality:

- **Introduction to Quality:**
- Quality is value to some person.
- Quality is fitness to purpose.
- Quality is exceeding the customer's expectations.
- **Quality in Use:** The user's view of the quality of a system.
  I.e. What's the end-user's experience?
- **External Quality Attributes:** External quality determines the fulfillment of stakeholder requirements. It is about the functionality of the system.
  I.e. Does it pass all the tests?
- **Internal Quality Attributes:** Internal quality has to do with the way that the system has been constructed.
  I.e. Is it well-designed?
- **Process Quality:** Process quality focuses on the steps of manufacturing the product.
  I.e. Is it assembled correctly?
- **Quality Assurance (QA):**
- Verification and validation (V&V) focuses on the quality of the product.
- QA focuses on the quality of the processes. It focuses on improving the software development process and making it more efficient and more effective.
- It looks at:
  - How well are the processes documented?
  - How well do people follow these processes?
  - Does the organisation measure key quality indicators?
  - Does the organisation learn from its mistakes?
- Examples of QA standards and practices are:
  - ISO9001
  - TickIt
  - Capability Maturity Model (CMM)
  - Total Quality Management (TQM)
- **A History of Managing Quality for Industrial Engineering:**
- **Product Inspection** (1920s): Examine intermediate and final products and discard defective items.
- **Process Control** (1960s): Monitor defect rates to identify defective process elements & control the process.
- **Design Improvement** (1980s): Engineering the process and the product to minimize the potential for defects.
- **Total Quality Management (TQM):**
- **Total quality management** is the continual process of detecting and reducing or eliminating errors in manufacturing, streamlining supply chain management, improving the customer experience, and ensuring that employees are up to speed with training.
- Total quality management aims to hold all parties involved in the production process accountable for the overall quality of the final product or service.
- TQM uses statistical methods to analyze industrial production processes.
- The basic principle of TQM is counter-intuitive: In the event of a defect, don't adjust the controller or you'll make things worse. Instead, analyze the process and improve it.
- It was developed by William Deming.
- While TQM shares much in common with the Six Sigma improvement process, it is not the same as Six Sigma. TQM focuses on ensuring that internal guidelines and process standards reduce errors, while Six Sigma looks to reduce defects.

- **Six Sigma:**
- **Six Sigma** is a quality-control methodology developed in 1986 by Motorola.
- The method uses a data-driven review to limit mistakes or defects in a corporate or business process. The key ideas are to use statistics to measure defects and to design the process to reduce defects.
- Six Sigma emphasizes cycle-time improvement while at the same time reducing manufacturing defects to a level of no more than 3.4 occurrences per million units or events.
- Six Sigma points to the fact that, mathematically, it would take a six-standard-deviation event from the mean for an error to happen.
- **Quality Management for Software:**
- All defects are design errors, not manufacturing errors.
- Process improvement principles still apply to the design process.
- **Defect removal:**
- There are two ways to remove defects:
    1. Fix the defects in each product. (I.e patch the product)
    2. Fix the process that leads to defects. (I.e. prevent defects from occurring)
- The latter is cost effective as it affects all subsequent projects.
- **Defect prevention:**
- Programmers must evaluate their own errors.
- Feedback is essential for defect prevention.
- There is no single cure-all for defects. They must be eliminated one by one.
- Process improvement must be an integral part of the process.
- Process improvement takes time to learn.
- **Six Sigma Might not be Suitable for Software:**
- Software processes depend on human behaviour, meaning they are not predictable.
- Software characteristics are not ordinal:
    - We cannot measure the degree of conformance for software.
    - The mapping between software faults and failures is many-to-many.
    - Not all software anomalies are faults.
    - Not all failures result from the software itself.
    - We cannot accurately measure the number of faults in software.
- Typical defect rates:
    - NASA Space shuttle: 0.1 failures/KLOC
    - Best military systems: 5 faults/KLOC
    - Worst military systems: 55 faults/KLOC
    - Six Sigma would demand 0.0034 faults/KLOC
- **Process Modeling & Improvement:**
- **Process Description:** Understand and describe the current practices.
- **Process Definition:** Prescribe a process that reflects the organization's goals.
- **Process Customization:** Adapt the prescribed process model for each individual project.
- **Process Enactment:** Carry out the process. This means developing the software and collecting process data.
- **Process improvement:** Use lessons learned from each project to improve the prescriptive model. I.e. Analyze defects to eliminate causes.
- **Capability Maturity Model (CMM):**
- The **Capability Maturity Model** is a process improvement approach developed specially for software process improvement.
- CMM has 5 levels. An organization is certified at CMM level 1 to 5 based on the maturity of their quality assurance mechanisms.

- CMM Levels

Level	Characteristic	Key Challenges
1 (Initial)	In this stage the quality environment is unstable. Simply, no processes have been followed or documented. Ad hoc/Chaotic. No cost estimation, planning, management.	Project Management Project Planning Configuration Management Change Control Software Quality Assurance
2 (Repeatable)	Some processes are followed which are repeatable. This level ensures processes are followed at the project level. Processes are dependent on individuals.	Establish a process group Identify a process architecture Introduce SE methods and tools
3 (Defined)	A set of processes are defined and documented at the organizational level. Those defined processes are subject to some degree of improvement. A process is defined and institutionalized.	Process measurement Process analysis Quantitative Quality Plans
4 (Managed)	This level uses process metrics and effectively controls the processes that are followed. The process is measured.	Automatic collection of process data Use process data to analyze and modify the process
5 (Optimizing)	This level focuses on the continuous improvements of the processes through learning & innovation. Improvement and feedback are fed back into the process.	Identify process indicators Empower individuals

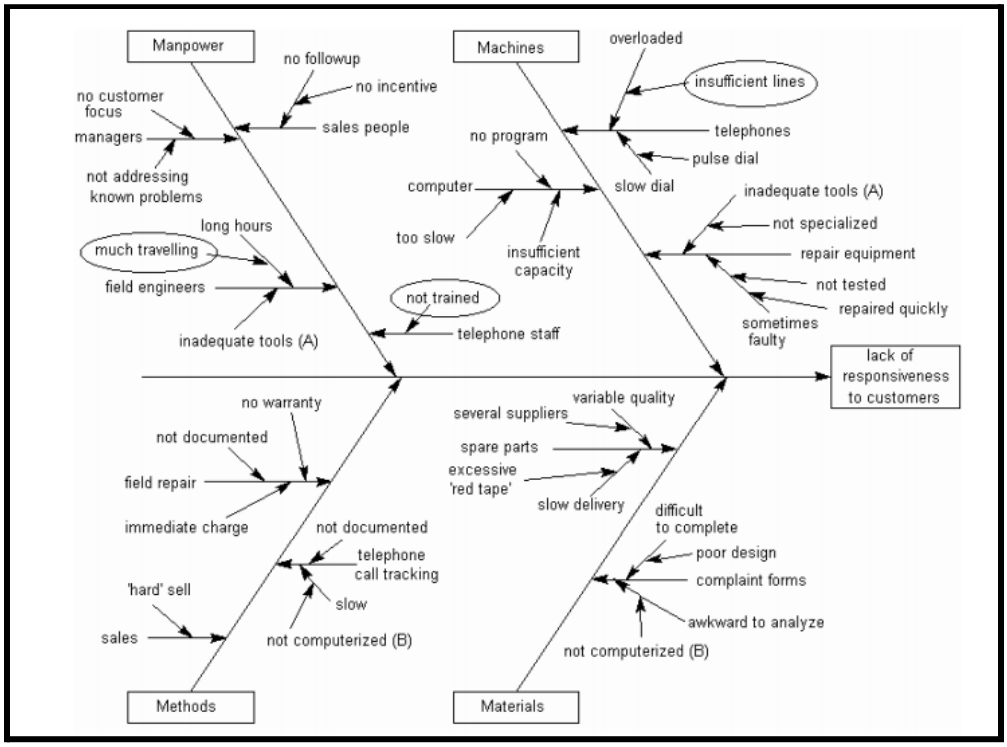
- **Arguments against QA:**

1. The costs may outweigh the benefits:
    - Costs: Increased documentation, more meetings, etc.
    - Benefits: Improved quality of the process outputs.
2. Reduced agility:
    - Documenting the processes makes them less flexible.
3. Reduced thinking:
    - Following the defined process gets in the way of thinking about the best way to do the job.
4. Barrier to Innovation:
    - New ideas have to be incorporated into the Quality Plan and get signed off.
5. Demotivation:
    - Extra bureaucracy makes people frustrated.

- **ISO 9000:**
- The ISO 9000 family of quality management systems is a set of standards that helps organizations ensure they meet customer and other stakeholder needs within statutory and regulatory requirements related to a product or service.
- ISO 9000 deals with the fundamentals of QMS, including the seven quality management principles that underlie the family of standards. It deals with the management systems

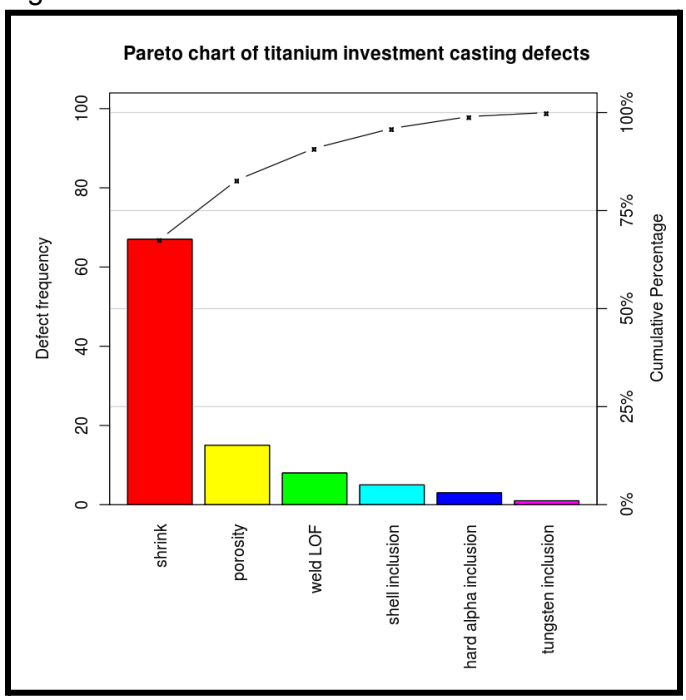
used by organizations to ensure quality in design, production, delivery, and support products.

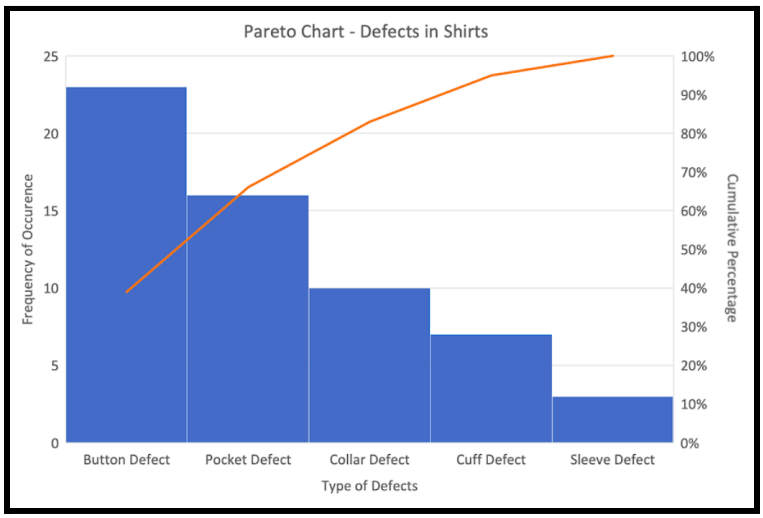
- ISO 9001 deals with the requirements that organizations wishing to meet the standard must fulfil. It includes several important changes for quality management systems, including modifications in terminology, the introduction of new context-based clauses, emphasis on management's role in quality, and a focus on risk-based approach.
- **Ishikawa (Fishbone) Diagram:**
- The fishbone diagram identifies many possible causes for an effect or problem.
- It can be used to structure a brainstorming session.
- It immediately sorts ideas into useful categories.
- We should use a fishbone diagram:
    - When identifying possible causes for a problem.
    - When a team's thinking tends to fall into ruts.
- E.g.



- **Pareto Chart:**
- A **Pareto chart** is a bar graph and a line graph. The lengths of the bars represent frequency or cost (time or money), and are arranged with longest bars on the left and the shortest to the right. In this way the chart visually depicts which situations are more significant. The line represents the cumulative percentage of defects.
- The left vertical axis is the frequency of occurrence. The right vertical axis is the cumulative percentage of the total number of occurrences.
- Pareto charts are useful to find the defects to prioritize in order to observe the greatest overall improvement.
- The Pareto chart is one of the seven basic tools of quality control.

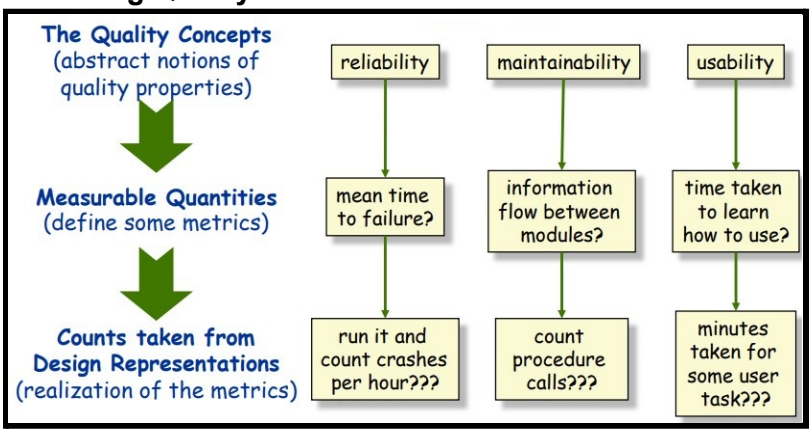
- E.g.



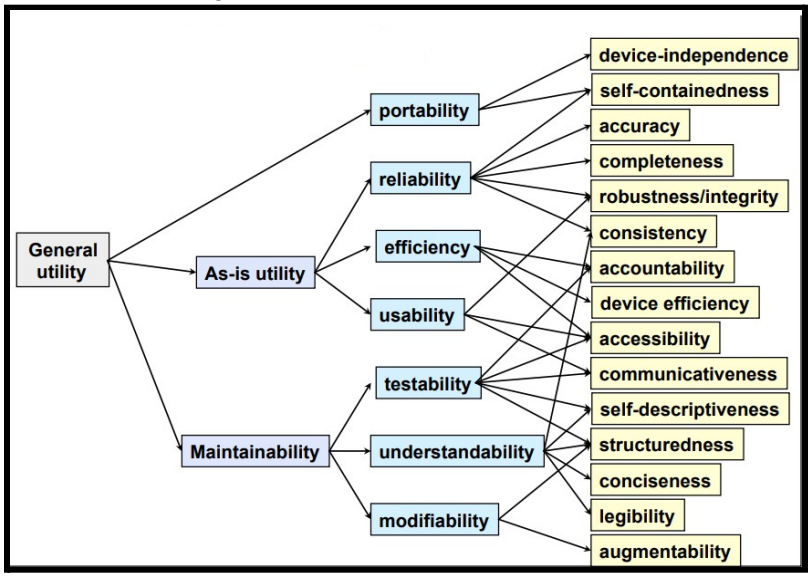


- **How to assess software quality:**
- Reliability:
- The designer must be able to predict how the system will behave:
    - Completeness - Does it do everything it is supposed to do?
    - Consistency - Does it always behave as expected?
    - Robustness - Does it behave well under abnormal conditions?
- Efficiency:
    - How efficient is the use of resources such as processor time, memory, network bandwidth?
    - This is less important than reliability in most cases.
- Maintainability:
    - How easy will it be to modify in the future?
- Usability:
    - How easy is it to use?

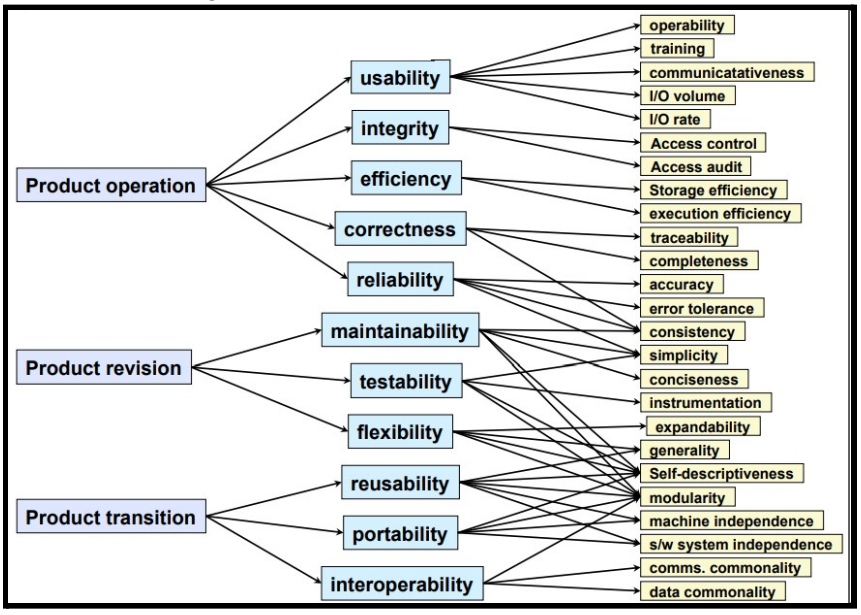
- **Measuring Quality:**



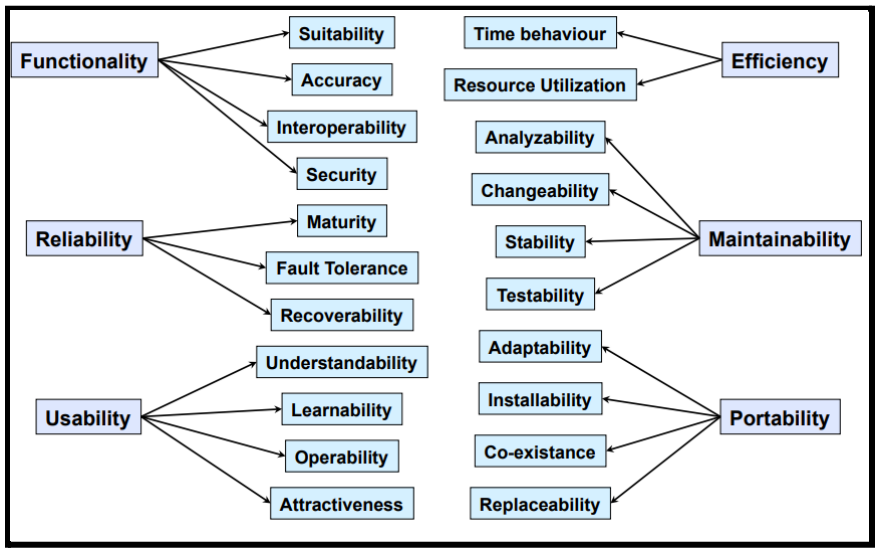
- **Boehm's Quality Map:**



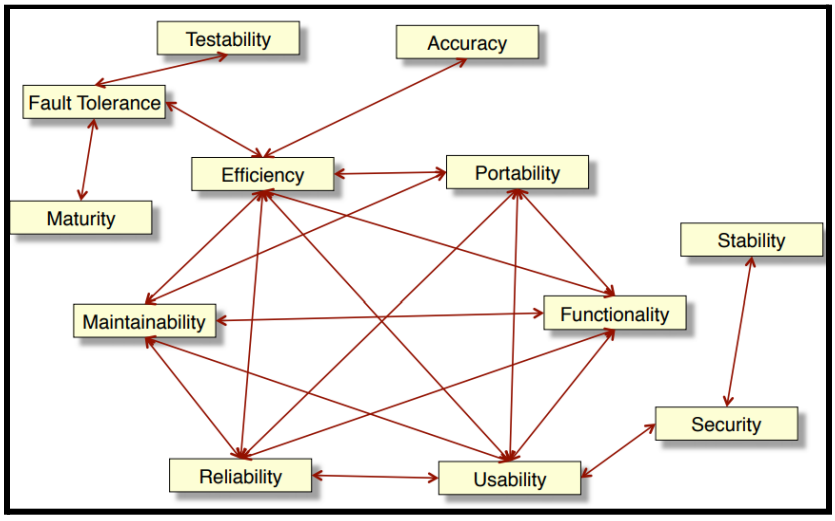
- **McCall's Quality Map:**



- **ISO/IEC 9126:**



- **Conflicts between Quality factors:**



We can summarize the above picture into the below one: