Software Quality Assurance

Five Views of Quality

- Value-based (Engineer to price)
- User-based (Fitness for purpose)
- Process-based (Conformance to requirements)
- Product-based (You get what you pay for)
- Transcendent (Excellence)

Software Quality Definition

- Conformance to explicitly stated functional and performance requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.

The Definition Emphasize

- Software requirements are the foundation from which quality is measured. **Lack of conformance to requirements is lack of quality**.
- Specified standards define a set of development criteria that guide the manner in which software is engineered. **If the criteria are not followed, lack of quality will almost surely result.**
- There is a set of implicit requirements that often goes unmentioned. **If software conforms to its explicit requirements, but fails to meet implicit requirements, software quality is suspect.**

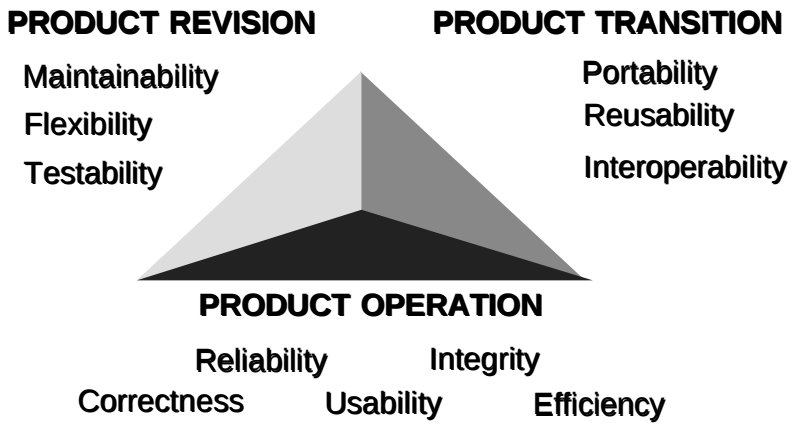
How Important Software Quality?

- It is important to have the understanding that the software quality work begins before the testing phase and continues after the software is delivered.

Software Quality Factors

- It can be categorized in two groups:
  - **Factors that can be Directly Measured** (e.g. errors; KLOC; unit-time)
  - **Factors that can be Measured only Indirectly** (e.g. usability or maintainability)
- Or, it can be categorized into:
  - **Internal** – Attributes which can be measured or observed in isolation.
  - **External** – Attributes which can only be observed in relation to external environment.

McCall's Triangle of Quality



McCall's Triangle of Quality

- McCall's quality factors were proposed in the early 1970s.
- They are as valid today as they were in that time.
- It's likely that software built to conform to these factors will exhibit high quality well into the 21st century, even if there are dramatic changes in technology.

Product Operations – Operation Characteristics

Correctness	The extent to which a program satisfies its specification and fulfills the customer's mission objectives.
Reliability	The extent to which a program can be expected to perform its intended function with required precision.
Efficiency	The amount of computing resources and code required by a program to perform its function.
Integrity	The extent to which access to software or data by unauthorized persons can be controlled.
Usability	The effort required to learn, operate, prepare input, and interpret output of a program.

Product Revision – Ability to Undergo Change

Maintainability	The effort required to locate and fix an error in a program.
Flexibility	The effort required to modify an operational program.
Testability	The effort required to test a program to ensure that it performs its intended function.

Product Transition – Adaptability to New Environments

Portability	The effort required to transfer the program from one hardware and / or software system environment to another.
Reusability	The extent to which a program (or parts of a program) can be reused in other applications - related to the packaging and scope of the functions that the program performs.
Interoperability	The effort required to couple one system to another.

ISO 9126 Quality Factors

- These factors provide a basis for indirect measures and an excellent checklist for assessing the quality of the system.
  - Functionality
  - Reliability
  - Usability
  - Efficiency
  - Maintainability
  - Portability

ISO 9126 Quality Factors – Functionality

- The degree to which the software satisfies stated needs as indicated by the following sub-attributes:
  - Suitability
  - Accuracy
  - Interoperability
  - Compliance
  - Security

ISO 9126 Quality Factors – Reliability

- The amount of time that the software is available for use as indicated by the following sub-attributes:
  - Maturity
  - Fault tolerance
  - Recoverability

ISO 9126 Quality Factors – Usability

- The degree to which the software is easy to use as indicated by the following sub-attributes:
  - Understandability
  - Learnability
  - Operability

ISO 9126 Quality Factors – Efficiency

- The degree to which the software makes optimal use of system resources as indicated by the following sub-attributes:
  - Time behaviour
  - Resource behaviour

ISO 9126 Quality Factors – Maintainability

- The ease with which repair may be made to the software as indicated by the following sub-attributes:
  - Analysability
  - Changeability
  - Stability
  - Testability

ISO 9126 Quality Factors – Portability

- The ease with which the software can be transposed from one environment to another as indicated by the following sub-attributes:
  - Adaptability
  - Installability
  - Conformance
  - Replaceability

Software Quality Metric

Auditability	The ease with which conformance to standards can be checked.
Accuracy	The precision of computations and control.
Communication commonality	The degree to which standard interfaces, protocols, and bandwidths are used.
Completeness	The degree to which full implementation of required function has been achieved.
Conciseness	The compactness of the program in terms of lines of code.
Consistency	The use of uniform design and documentation techniques throughout the software development project.
Data commonality	The use of standard data structures and types throughout the program.
Error tolerance	The damage that occurs when the program encounters an error.
Execution efficiency	The run-time performance of a program.
Expandability	The degree to which architectural, data, or procedural design can be extended.
Generality	The breadth of potential application of program components.
Hardware independence	The degree to which the software is de-coupled from the hardware on which it operates
Instrumentation	The degree to which the program monitors its own operation and identifies errors that do occur.
Modularity	The functional independence of program components.
Operability	The ease of operation of a program.
Security	The availability of mechanisms that control or protect programs and data.
Self-documentation	The degree to which the source code provides meaningful documentation.
Simplicity	The degree to which a program can be understood without difficulty.
Software system independence	The degree to which the program is independent of nonstandard programming language features, operating system characteristics, and other environmental constraints.
Traceablilty	The ability to trace a design representation or actual program component back to requirements.
Training	The degree to which the software assists in enabling new users to apply the system.

The Attributes of Effective Software Metrics

- Simple and Computable
- Empirically and Intuitively Persuasive
- Consistent and Objective
- Consistent in its Use of Units and Dimensions
- Programming Language Independent
- An Effective Mechanism for Quality Feedback

Attributes of Effective Software Metrics – Simple and Computable

- It should be relatively easy to learn how to derive the metric, and its computation should not demand inordinate effort or time.

Attributes of Effective Software Metrics – Empirically and Intuitively Persuasive

- The metric should satisfy the engineer's intuitive notions about the product attribute under consideration.

Attributes of Effective Software Metrics – Consistent and Objective

- The metric should always yield results that are unambiguous.

Attributes of Effective Software Metrics – Consistent in Use of Units and Dimensions

- The mathematical computation of the metric should use measures that do not lead to bizarre combinations of unit.

Attributes of Effective Software Metrics – Programming Language Independent

- Metrics should be based on the analysis model, the design model, or the structure of the program itself.

Attributes of Effective Software Metrics – Effective Mechanism for Quality Feedback

- The metric should provide a software engineer with information that can lead to a higher quality end product.

Software Reliability

- Most hardware-related reliability models are predicated on failure due to wear rather than failure due to design defects.
- In hardware, failures due to physical wear (e.g. the effects of temperature, corrosion, shock) are more likely than a design-related failure.
- The opposite is true for software: in fact, all software failures can be traced to design or implementation problems; wear does not enter into the picture.

Reliability Metric

- Reliability metric is an indicator of how broken a program is.
- Metrics are best weighted by the by severity of errors.
- A minor error every hour is better than a catastrophe every month.
- These metrics are not the same as counting bugs, but indicate different probabilities of happening.

Mean Time Between Failure (MTBF)

- MTBF = MTTF + MTTR
- Measures how long a program is likely to run before it does something bad like crash.
  - MTTF is the mean time to failure.
  - MTTR is the mean time to repair.

Availability

- Software availability is the probability that a program is operating according to requirements at a given point in time
  - Availability = MTTF/ (MTTF + MTTR) * 100%
    - MTTF is the mean time to failure.
    - MTTR is the mean time to repair.

Probability of Failure on Demand

- Common for safety-critical systems
  - e.g. a specification may be that there not exceed 1 chance in 10^{10} that a missile gets through.

Rate of Failure Occurrence

- This tells how many may occur in a given period
  - e.g. 2 errors per 100 minutes. This is considered one of the best metrics

Quality Concepts

- Controlling variation among products is what quality assurance work is all about.
- Software engineers are concerned with controlling the variation in their processes, resource expenditures, and the quality attributes of the end products.
- Also need to be aware that customer satisfaction is very important to modern quality work as is quality of design and quality of conformance.

Cost of Quality

- Prevention Costs
  - Quality planning, formal technical reviews, test equipment, training, etc
- Appraisal Costs
  - In-process and inter-process inspection, equipment calibration and maintenance, testing, etc
- Failure Costs
  - Internal Failure Costs (rework, repair, failure mode analysis)
  - External Failure Costs (complaint resolution, product return and replacement, help line support, warranty work)

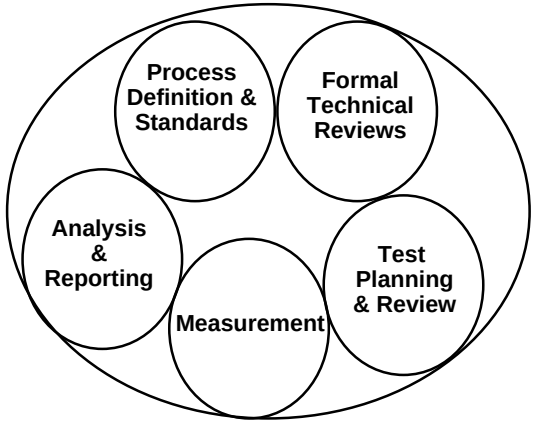
Cost Impact of Software Defects

- The later in the life cycle that an error is detected the more expensive it is to repair.
- Errors remain latent and are not detected until well after the stage at which they are made.
  - 54% of errors detected after coding and unit testing.
  - 45% of these errors were requirements and design errors.
- There are numerous requirements errors.
  - Estimates indicate that 56% of all errors are errors during the requirements stage.
- Requirements errors are typically nonclerical.
  - Estimates indicate that 77% requirements errors were nonclerical.

The Goal of Quality Assurance

- To provide management with the necessary to be informed about product quality, thereby gaining insight and confidence that product quality is meeting its goals

Software Quality Assurance



Software Quality Assurance (SQA)

- Software Quality Assurance is an essential activity for any business that produces products to be used by others.
- It is a "planned and systematic pattern of actions" that are required to ensure quality in software.
- The SQA group must look at software from the customer's perspective, as well as assessing its technical merits.
- The activities performed by the SQA group involve quality planning, oversight, record keeping, analysis and reporting.

Software Quality Management

- Concerned with ensuring that the required level of quality is achieved in a software product
- Involves defining appropriate quality standards and procedures and ensuring that these are followed
- Should aim to develop a 'quality culture' where quality is seen as everyone's responsibility

Measuring Quality

- The main factors that defined software quality includes
  - **Correctness** – the degree to which a program operates according to specification
  - **Maintainability** – the degree to which a program is amenable to change
  - **Integrity** – the degree to which a program is impervious to outside attack
  - **Usability** – the degree to which a program is easy to use

Software Quality – Correctness

- A program must operate correctly.
- Correctness is the degree to which the s/w performs its required operation.
- The most common measure for correctness is defects per KLOC, where a defect is defined as a verified lack of conformance to requirements.

Software Quality – Maintainability

- It cannot be measure directly. So we must use indirect measures.
- A simple time oriented metric is mean time -to-change (MTTC), the time it takes to analyse the change request, design an appropriate modification, implement the change, test it and distribute the change to all users.
- On average, programs that are maintainable will have a lower MTTC (for equivalent types of changes) than programs that are not maintainable.

Software Quality – Integrity

- S/w integrity has become increasingly important in the age of hackers and viruses. This attribute measures a system's ability to withstand attacks (both accidental and intentional) on its security. Attack can be made on all 3 components of s/w: programs, data and documents.
- The integrity of a system can be defined as :
  - Integrity = [1 - threat X (1 - security)]
- where Threat is the probability that can attack of a specific type will occur within a given time. Security is the probability that the attack of a specific type will be repelled.

Software Quality – Usability

- It is an attempt to quantify "user friendliness" and can be measured in terms of 4 characteristics:
  - The physical and or intellectual skill required to learn the system;
  - The time required to become moderately efficient in the use of the system
  - the net increase in productivity measured when the system is used by someone who moderately efficient and
  - A subjective assessment (using a questionnaire) of users attitude towards the system.

SQA Seven Major Activities

- Application of Technical Methods
- Conduct of Formal Technical Reviews
- Software Testing
- Enforcement of Standards
- Control of Change
- Measurement
- Record Keeping and Reporting

SQA Seven Major Activities – Application of Technical Methods

- It helps analyst to achieve a high-quality specification and the designer to develop a high-quality design.

SQA Seven Major Activities – Conduct of Formal Technical Reviews

- It uncovers quality problems.

SQA Seven Major Activities – Software Testing

- It combines a multi-step strategy with a series of test case design methods that help ensure effective error detection.

SQA Seven Major Activities – Enforcement of Standards

- SQA activity must be established to ensure that standards are followed.

SQA Seven Major Activities – Control of Change

- It contributes directly to software quality by formalizing requests for change, evaluating the nature of change, and controlling the impact of change.

SQA Seven Major Activities – Measurement

- An important objective of SQA is to track software quality and assess the impact of methodological and procedural changes on improved software quality.
- To accomplish this, measurement must be done.

SQA Seven Major Activities – Record Keeping and Reporting

- It provides procedures for the collection and dissemination of SQA information.

The SQA Plan

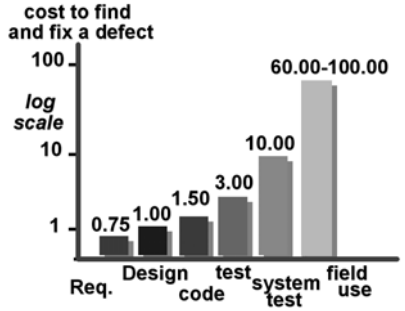
- The SQA plan provides a road map for instituting software quality assurance.
- Developed by the SQA group, the plan serves as a template for SQA activities that are instituted for each software project.
- The remainder of SQA plan identifies the tools and methods that support SQA activities and tasks, software configuration management procedures etc.

The SQA Plan Standard (IEEE)

- Initial section describes the purpose and scope of the document and software process activities.
  - Management section describes organizational structure, SQA activities and tasks etc.
  - The documentation section describes each of the work products produced as part of the software process.
  - The standards, practices and conventions section describes all applicable standards and practices.
  - The reviews and audits section describes reviews and audits to be conducted by different group.
  - The test section describes software test plan and procedure and record keeping requirements.

Tradeoff between Cost and Reliability

- It demands more time and resources for testing and debugging.
- It requires more internal safety checks, and this makes programs larger and slower.



What are Reviews?

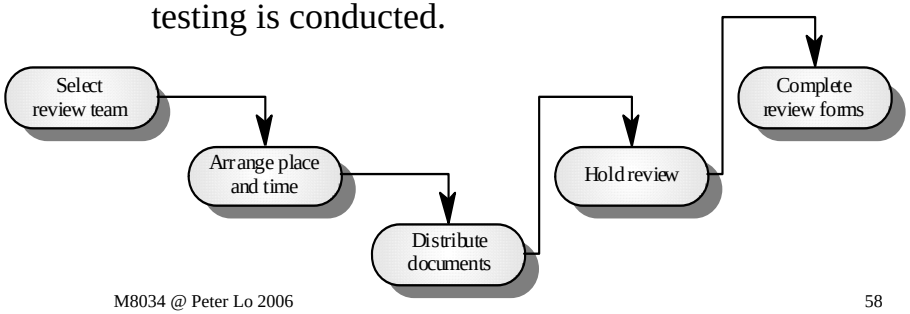
- A meeting conducted by technical people for technical people
- A technical assessment of a work product created during the software engineering process
- A software quality assurance mechanism
- A training ground

What Reviews are Not!

- They are not:
  - A project budget summary
  - A scheduling assessment
  - An overall progress report
  - A mechanism for reprisal or political intrigue!!

Software Reviews

- It is important to note that any work product (including documents) should be reviewed.
- Conducting timely reviews of all work products can often eliminate 80% of the defects before any testing is conducted.



Types of Reviews

Review type	Principal purpose
Design or program inspections	To detect detailed errors in the design or code and to check whether standards have been followed. Thereview should be driven by a checklist of possible errors.
Progress reviews	To provide information for management about the overall progress of the project. This is both a process and a product review and is concerned with costs, plans and schedules.
Quality reviews	To carry out a technical analysis of product components or documentationto find faults or mismatches between the specification and the design, code or documentation. It may also be concerned with broader quality issues such as adherence to standards and other quality attributes.

Quality Reviews

- Objective is the discovery of system defects and inconsistencies
- Any documents produced in the process may be reviewed
- Review teams should be relatively small and reviews should be fairly short
- Review should be recorded and records maintained

Review Results

- Comments made during the review should be classified.
  - No action.
  - Refer for repair – Designer or programmer should correct an identified fault.
  - Reconsider overall design – The problem identified in the review impacts other parts of the design. Some overall judgement must be made about the most cost-effective way of solving the problem.
- Requirements and specification errors may have to be referred to the client.

Formal Technical Reviews

- Formal technical review is
  - A class of reviews that include walkthroughs, inspections, round-robin reviews, and other small group technical assessments of software
  - A planned and controlled meeting attended by a group of diversified people
- Formal technical reviews can be conducted during each step in the software engineering process.
- A brief checklist can be used to access products that are derived as part of software development.

Objectives of Formal Technical Review

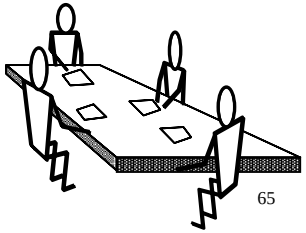
- To uncover errors in function, logic, or implementation for any representation of the software
- To verify that the software under review meets its requirements
- To ensure that the software has been represented according to predefined standards
- To achieve software that is developed in a uniform manner
- To make projects more manageable

Effects of Formal Technical Review

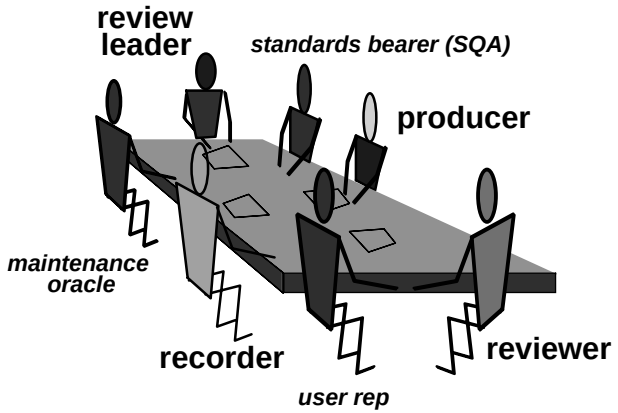
- Early discovery of software defects so the development and maintenance phase is substantially reduced
- Serves as a training ground, enabling junior engineers to observe different approaches to software analysis, design, and implementation
- Serves to promote backup and continuity because a number of people become familiar with parts of the software that they may not have otherwise seen

Conducting the Review

- Be prepared – evaluate product before the review
- Review the product, not the producer
- Keep your tone mild, ask questions instead of making accusations
- Stick to the review agenda
- Raise issues, don't resolve them
- Avoid discussions of style – stick to technical correctness
- schedule reviews as project tasks
- record and report all review results



The Players



Metrics Derived from Reviews

- Inspection time per page of documentation
- Inspection time per KLOC or FP
- Errors uncovered per reviewer hour
- Errors uncovered per preparation hour
- Errors uncovered per SE task (e.g., design)
- Number of minor errors (e.g., typos)
- Number of errors found during preparation
- Number of major errors (e.g., nonconformance to requirement)
- Inspection effort per KLOC or FP

Review Options Matrix

	IPR*	WT	IN	RRR
trained leader	no	yes	yes	yes
agenda established	maybe	yes	yes	yes
reviewers prepare in advance	maybe	yes	yes	yes
producer presents product	maybe	yes	no	no
"reader" presents product	no	no	yes	no
recorder takes notes	maybe	yes	yes	yes
checklists used to find errors	no	no	yes	no
errors categorized as found	no	no	yes	no
issues list created	no	yes	yes	yes
team must sign-off on result	no	yes	yes	maybe

IPR—informal peer review WT—Walkthrough
*IN—Inspection RRR—round robin review

The checklists

- The checklists are not to be comprehensive, but rather to provide a point of departure for each review.
  - System Engineering
  - Software Project Planning
  - Software Requirements Analysis
  - Software Design
  - Preliminary Design Review
  - Design Walkthrough
  - Coding
  - Software Testing
  - Test Plan
  - Test Procedure
  - Maintenance

Fagan Inspection

- Michael Fagan noticed that no inspection of designs was routinely practiced during software development in IBM during 1970s
- He developed a set of procedures for inspection of software designs, source code, test plans and test cases.
- Fagan argued that software development should
  - Clearly define the programming process as a series of operations, each with its own exit criteria.
  - Measure the completeness of the product at any point of its development by inspections or tests.
  - Use measurements to control the process.

Types of Inspection

Phases	Inspections
I0 Initial Design	Initial Design Specification (include functional & module specification) is inspected against Statement of requirements.
I1 Detailed Design	Logic Specifications are inspected against the Initial Design Specification
I2 Coding	Source Code is inspected against Logic Specifications.
IT1 Test Plan Preparation	Test Plan is inspected against Functional Specification.
IT2 Test Case Preparation	Test Case Listings are inspected against Test Plan.

Inspection Team

Role	Task
Moderator	In charge of the whole inspection. Chair the meeting. He should not be involved in develop material under inspection. A technical person.
Author	Give initial presentation. Rework to remove defect.
Inspector	Normally two inspectors participate, usually members of development team that produced the material under inspection (but not the same phase)
Secretary	Recording, enter data to database.

Inspection Procedure

Steps	Participants	Objectives	Remarks
1. Planning	Moderator	Schedule Activities & Distribute Material	Include higher-level document, checklists etc.
2. Overview	Author & others	Education	Author gives presentation to other participants
3. Preparation	All Participants	Familiarization	Study privately
4. Inspection	Entire Team	Find Defects	A 2 hour meeting. Defects are recorded. Moderator decides the material pass or not. Produce report.
5. Rework	Author	Correct Defects	
6. Follow Up	Moderator & Author	Assure Rework is Correct; Improve Development Process; Improve Inspection Efficiency	

The Inspection Meeting

- 3-5 people only
- Lasts for two hours only
- The decision is based on standards which are specific to the organization.
- The moderator first outlines the material to be inspected and describes the intended function of the corresponding part of the system.
- The reader then reads through all the material.
- The inspection is extremely thorough
- At the end of the meeting, the defect list is approved by all participants, and the moderator decides whether or not a re-inspection will be necessary after the rework.

Exit Criteria

- To judge whether a given development phase is complete.
- It must be defined by individual organizations to meet the needs of their own development environment.
- Some examples:
  - I0: external specifications are completed
  - I1: design specifications must be structured
  - I2: module prologue up-to-date and complete

Checklists

- Set of questions inspectors to ask themselves.
- Different set of checklists corresponding to different type of inspection & programming language.
- Should be continually added to improve inspection experience.

Defect Recording and Classification

- Defect is record with serial number, type, category, severity, line number in material, description and estimate time to fix.

Class	Value
Severity	Major, Minor
Category	Missing, Wrong, Extra
Type	As defined in checklist

Inspection Database

- Record effort required for preparation, number of defect detected, size of material inspected.
- Use to monitor and control development process, and improve the effectiveness of the inspections.

Measures for Inspection

Attributes	Measures
Effort	Number of person-hours
Code Size	NCSS {non-Comment Source Statements}
Specification Size	NCSS
Rework Size	NCSS {statement added, modified or deleted}
Inspection Rate	Size of material / Person-hours
Rework Rate	Size of material / Person-hours
Defects Detected	Count {can grouped by inspection, module, type, category or severity}
Defects Present	Count {estimate number of defects before inspection}
Defects Remaining	Count {estimate number of defects after inspection}
Defect Density	Defects Count / Size of Module {can apply to present, detected or remaining}
Defect Removal Efficiency	Defects Removed / Defects present {in %, for an inspection step}

Use of Defect Type Distribution

- Inspection improvement (Can focus on prevalent defect)
- Programmer self-improvement
- Development process improvement (Guide management in providing training, standard and procedures)

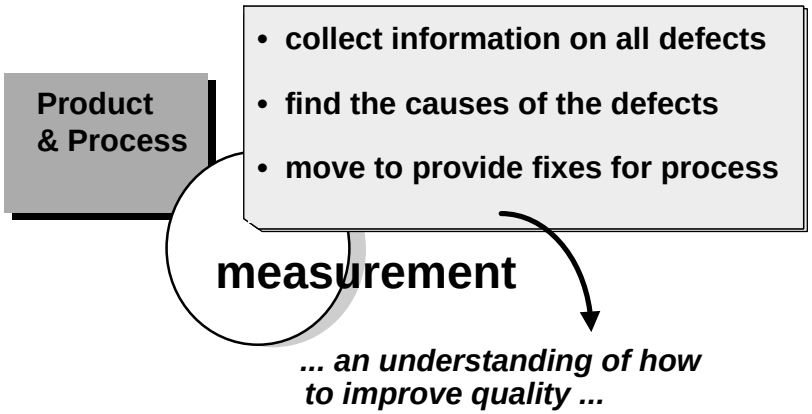
Use of Defect Type Distribution

- Defect-prone modules are those in which a higher than average number of defects are detected (or estimated to remain).
- It may due to badly written or more complex. Special attention should be paid on such modules.
- Inspection data MUST NOT used to assess people. Punishment will delay development process.

Effectiveness of Fagan Inspection

- "Amplification and detection" – single defect in high level spec. may give rise to many defects in detailed spec.
- Cost of defect detection – 6 to 8 times less than failure during test.
- Reduction in defect density – experiment show 38% few faults than informal walk-through.
- Improve productivity – experiment show 23% increase due to reduction of rework time.

Statistical SQA



Statistical Software Process Improvement (SSPI)

1. All errors and defects are categorised by origin. (e.g., flaw in specification, flaw in logic, non-conformance to standards).
2. The cost to correct each error and defect is recorded.
3. The number of errors and defects in each category are counted and ordered in descending order.
4. The overall cost of errors and defects in each category is computed.
5. Resultant data are analysed to uncover the categories that result in highest cost to the organisation.
6. Plans are developed to modify the process with the intent of eliminating (or reducing the frequency of occurrence of) the class of errors and defects that is most costly.

Factors that Influence Software Productivity

- **People Factors** – The size and expertise of the development organisation
- **Problem Factors** – The complexity of the problem and the number of changes in design requirements
- **Process Factors** – Analysis and design techniques that are used, languages and tools available.
- **Product Factors** – Reliability and performance of the computer-based system.
- **Resource Factors** – Availability of CASE tools and hardware and software resources.

Defect Removal Efficiency (DRE)

- DRE is a measure of the filtering ability of quality assurance and control activities as they applied throughout all process framework activities.
- DRE can be defined as **DRE = E/(E+D)**
- where
  - E = no. of errors found before the delivery of the software to the end user.
  - D = no. of defects found after delivery.
- The ideal value for DRE is 1. (i.e., no defect). If used as metric, DRE encourages a software project team to apply techniques for finding errors as many as possible before delivery.