Fair Use Disclaimer Notice

The material used in this presentation i.e., pictures/graphs/text, etc. is solely intended for educational/teaching purpose, offered free of cost to the students for use under special circumstances of Online Education due to COVID-19 Lockdown situation and may include copyrighted material - the use of which may not have been specifically authorized by Copyright Owners. It's application constitutes Fair Use of any such copyrighted material as provided in globally accepted law of many countries. The contents of presentations are intended only for the attendees of the class being conducted by the presenter.

Outlines

- Software Reviews
- Formal Technical Reviews
- Review guidelines for FTR
- Statistical Quality Assurance

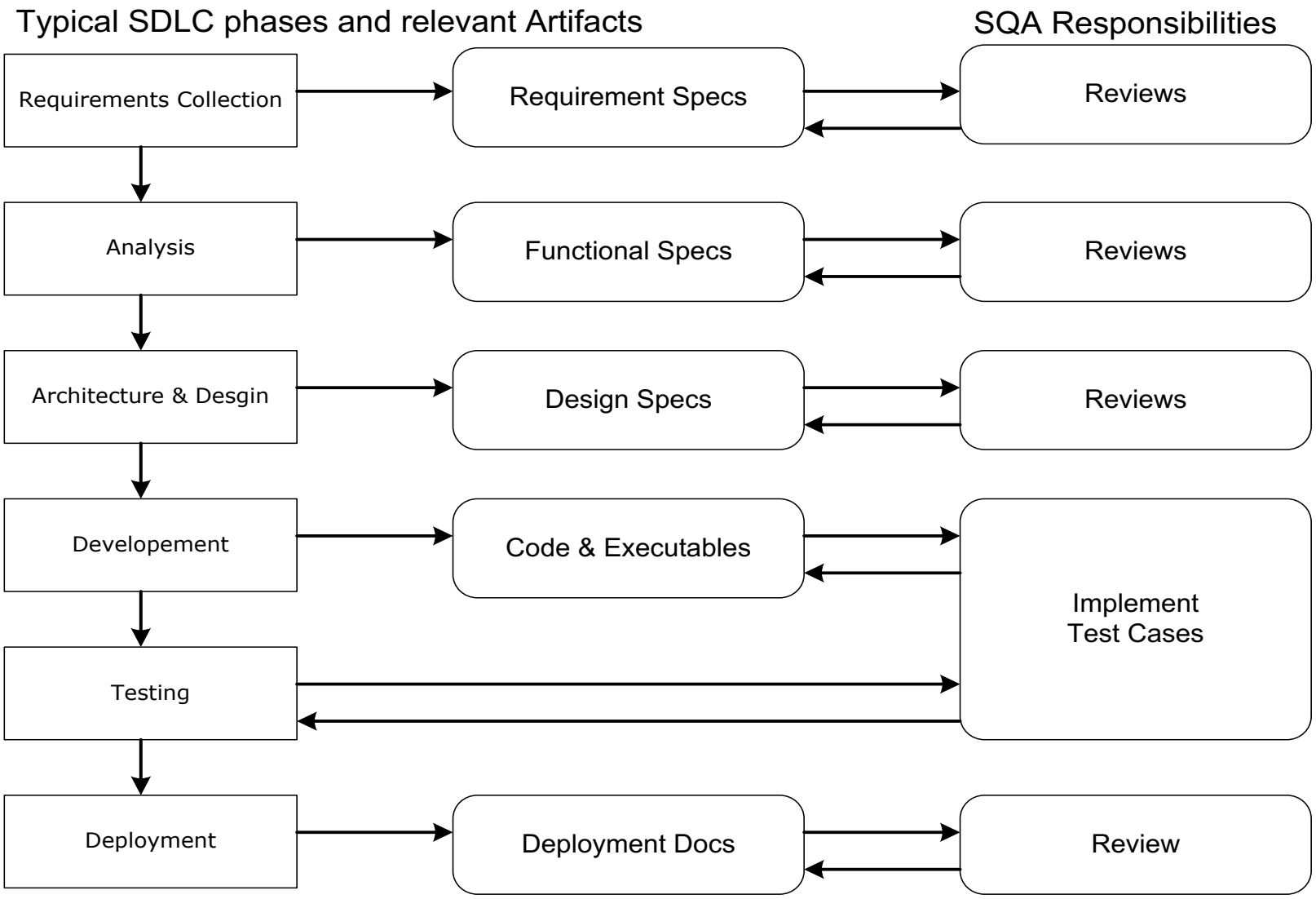
Software Quality Assurance: Objectives

- To "**improve**" quality.
- To **take preventive** as well as **corrective actions** to eliminate **defects**.
- To **verify** and **validate** that the product is meeting customer requirements and standards.

Software Quality Assurance: Objectives

- **Review** of *documents* developed by development team.
- **Track** the *compliance* with standards.
- **Development** of *QA Plan* (test plan + test cases).
- **Implementation** of *test cases*.
- **Management** of *bug repository*.
- **Participating** in *code* and *design* reviews.

SQA Activities of a Project



Software Reviews

- A **review** is …..
  - **A way** to identify the needed improvements of the parts in a ***product***
  - **Confirm** the improvement parts of a ***product***.
  - **Achieve** technical work that is more uniform, predicable, and manageable.
- **What** is **software reviews**?
  - A "filter" for the software engineering process.
- **Purpose:** serves to uncover errors in analysis, design, coding, and testing.
- **Why software reviews**?
  - **To err** is human
  - **Easy** to catch the errors in engineers' work

Checklist for Reviews

- Check the followings:
  - Completeness
  - Consistency
  - Ambiguities
  - Thoroughness
  - Conformance to template
  - Suitability of Architecture and Design

Software Reviews

- ***THE BOTTOM LINE***
  - A staged approach to **Formal Technical Reviews (FTRs)**, and **Inspections** can improve the acceptability of *quality controls*, as well as encouraging very early *defect detection*.

Software Inspection refers to peer review of ***any work product*** by trained individuals who look for defects using a ***well defined process***

Formal Technical Reviews

- **Formal Technical Reviews** and **Inspections** of documents or software are performed to **identify** and **remove** defects.

- **Reviews of context**
  - *means* the need, prospects and risks of *future work*

- **Reviews of content**
  - Technical reviews evaluate an artifact's *quality* and *fitness* for purpose; they are reviews of *content*.

Formal Technical Reviews

- **Purposes of FTR**
  - **Performed early** in the software development lifecycle, **FTRs** can remove potentially expensive requirements or design defects.
  - **Performed later**, on code, they can indicate the defect densities to be expected during testing and help shape testing strategies.
  - **To uncover** errors in function, logic, or implementation
  - **To verify** the software under review **meets** its *requirements*
  - **To ensure** that the software has been represented according to predefined standards
  - **To develop** software in a uniform manner
  - **To make** projects more manageable

Formal Technical Reviews

- **FTRs** can be applied to:
  - Requirements specification
  - System design
  - Preliminary design
  - Detailed/Critical design
  - Program/code
  - User document
  - Any other defined development product

Formal Technical Reviews

- **Review** meeting's constraints:
  - **3-5 people involved** in a review
  - **Advanced preparation** (no more than 2 hours for each person)
  - The **duration** of the review meeting should be less than 2 hours
  - **focus** on a **specific** part of a software product

- **People involved** in a review meeting:
  - **Producer**, **review leader**, **2 or 3 reviewers** (one of them is recorder)

- **Formal Technical Review** Meeting
  - A **meeting** agenda and schedule (by *review leader*)
  - **Review** material and distribution (by *the producer*)
  - **Review** in advance (by *reviewers*)

Formal Technical Reviews

- **Review** meeting results:
  - *A review* issues **list**
  - *A simple review summary report* (called meeting minutes)

- **Meeting** decisions:
  - *Accept* the work product without **further modification**
  - *Reject* the work product due to **errors**
  - *Accept* the work **under** conditions (such as change and review)
  - *Sign-off* sheet

Review Guidelines for FTR

GUIDELINES FOR REVIEWS

Guidelines for the conduct of Formal Technical Reviews should be established in advance, distributed to all reviewers, agreed upon, and then followed.

1. **Review the product, not the producer.**
   FTR involves people and egos. Errors should be pointed out gently; the tone of the meeting should be loose and constructive; the intent is not to belittle or embarrass.

2. **Set an agenda and maintain it.**
   A malady of any meeting is drift. An FTR must be kept on track and on schedule.

3. **Limit debate and rebuttal.**
   When an issue is raised by a reviewer, if there is no universal agreement on its impact, record the issue for further discussion off-line, rather than spend time debating the question.

4. **Don't attempt to solve the problem.**
   A review is not a problem-solving session. Problem solving should be postponed until after the review meeting.

5. **Take written notes.**
   Sometimes a good idea for the recorder to make notes on a wall board so that wording and prioritization can be assessed by other reviewers as the information is recorded.

6. **Limit the number of participants.**
   Two heads are better than one, but 14 are not necessarily better than 4.

Review Guidelines for FTR

7. Insist upon advance preparation.
All review members must prepare in advance. Written comments should be solicited by the review leader.

8. Develop a checklist for each product that is likely to be reviewed.
A checklist helps the review leader to structure the FTR meeting, and helps each reviewer to focus on important issues.

9. Allocate resources and time schedule for FTRs.
To be effective, FTRs should be scheduled as a task during the software engineering process. In addition, time should be scheduled for the inevitable modifications that will occur as the result of an FTR.

10. Review your early reviews.
Debriefing can be beneficial in uncovering problems with the review process itself. The very first product to be reviewed might be the review guidelines themselves and your development standards.

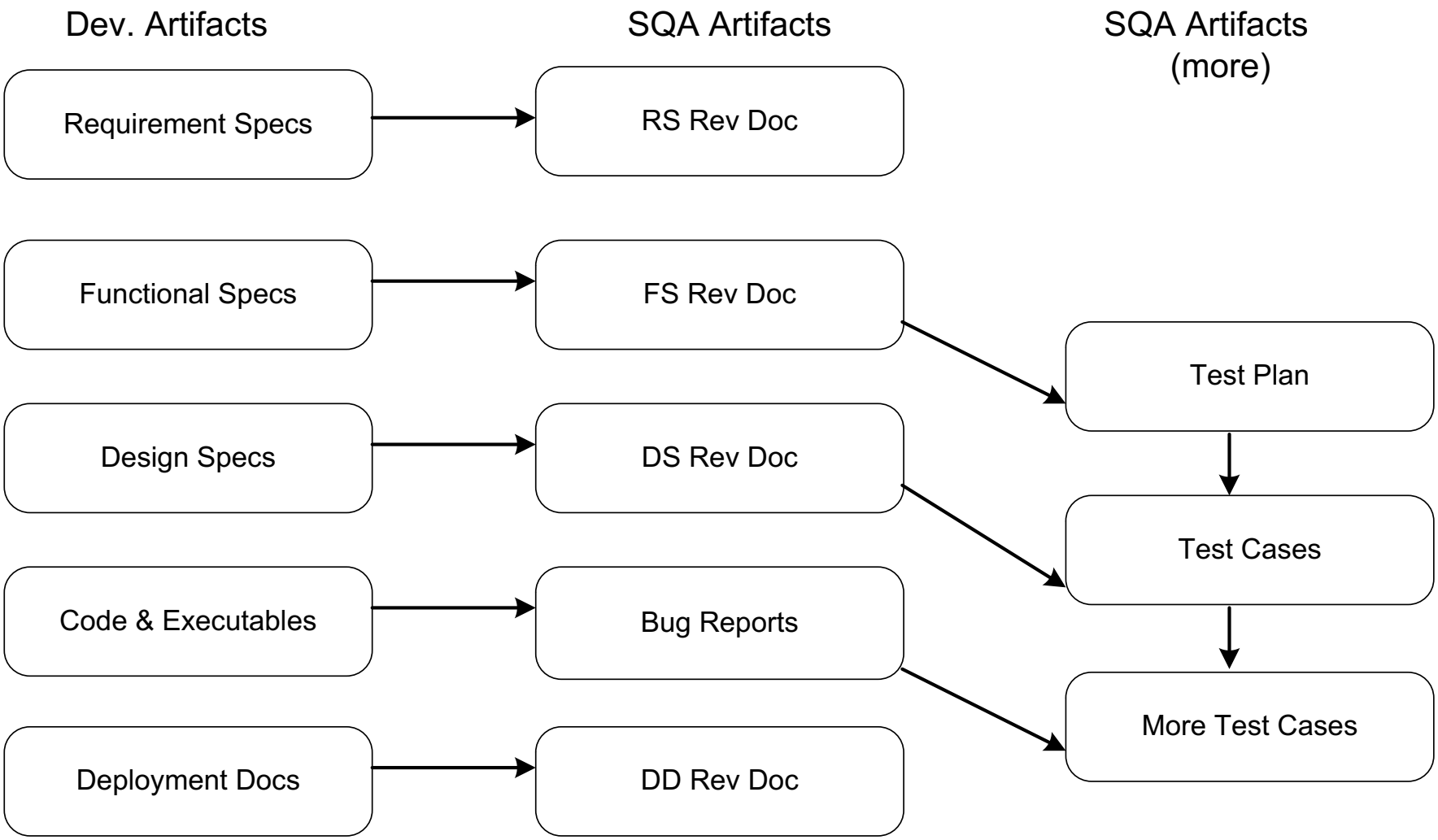
11. Restrict a Design Review to Reviewing One Design.
Don't use a design review to compare two or more designs, but use two or more design reviews to compare two or more designs. When you try to do two designs at once, the review may turn into a yelling contest for the advocates of the various alternatives.

12. Conduct meaningful training for all reviewers.
To be effective, all review participants should receive some formal training. (Freedman & Weinberg [FRE82] estimate a one-month learning curve for every 20 people who are to participate effectively in reviews).

Types Of Reviews

- System Requirements Review (SRR)
- Preliminary Design Review (PDR)
- Critical Design Review (CDR)
- Production Readiness Review (PRR)
- System Test

Example: Test Plan & Test Cases



Example: Test Case Execution

- **Take printouts** of test case(s).
- **Follow the instructions** to execute test case, as mentioned in it.
- **Check the results**, is it pass or fail?
- **Write the result** using PEN on test case.
- **Sign it**.

Example: Reporting Defects

- **Objective:**
  - To get defects rectified.
- **Clients of reported defect:**
  - Development
  - SQA Manager
- **How to report defects?**
  - Any good tools can be used for reporting and tracking **defects**.
  - **A simple Excel sheet** can also be used, however it is **difficult** to *maintain* and is error-prone.

Example: Recommended Defects Severity

Defect Severity	Defect Name	Description
1	**Critical**	Testing has **stopped** early in the test process and **cannot continue** without a ***fix***.
2	Major	Testing is **severely limited**, usually caused by an **inoperative** or **undelivered** piece of **major *functionality***.
3	**Significant**	A **problem** with the functionality is **detected**. It is **not** a "***show – stopper***," but has an ***impact*** to the overall quality of the release.
4	Minor	A **very minimal problem** having little or no impact on ***production*** with the current release.
5	**Enhancement**	A **suggested improvement** to the product, e.g., usability, performance, or expanded functionality.

Example: Test Cycle

- **A test cycle** is a complete **testing activity** for any *specified* component or system.
- **All the relevant test cases** are executed in a *single* Test cycle.
- **At least two test cycles** are strongly recommended.
- ***Signing SQA Certificate***
  - After going through **all the phases**, the SQA issues a *certificate*, provided that the software meets pre-agreed exit criteria

Top Management Support to SQA

- **Proper** staffing and hiring:
  - **Hire** on merit only (**do not** *put relatives in SQA*)
  - **Hire** all-rounder people in SQA
  - **Hire** technical as well as non-technical personnel (e.g. domain specialists, language gurus)
  - **Focus** on people with *relatively higher analytical abilities*
  - **Provide** on-job trainings, career path and competitive compensation package
- **Separate** budget for *payroll* and *running expenses*
  - **SQA payroll** should **not be handled** by *Development*
- **Separate** capital budget
  - For hardware required.
  - For automated test & test management tools

Top Management Support to SQA

- **Proper** test facilities and environment
  - Test Servers
  - Client machines
  - Separate sitting space (minimal interference from development)
  - Other equipment (if required) to facilitate SQA process

- **Trust**, **confidence** and **authority**
  - **Don't force** to close defects.
  - **Don't force** to sign acceptance.
  - **Act as an arbitrator** for the conflicts between QA/Dev

- **Framework** for evaluating the ***performance*** of SQA members

Responsibilities of SQA Team

- **Responsibilities** of a tester
  - Design test cases
  - Write test scripts
  - Implement test cases

- **Responsibilities** of SQA manager
  - Give approximately 25% time for reviewing the written test cases
  - Lead review meetings
  - Resolve conflicts

Characteristics of a good SQA Member

- **Experience** and/or **education** as a programmer or analyst
- A thick skin and a good sense of humor
- **Tolerance** for chaos
- **Tenacity** (*firmness*)
- **Evidence oriented** (*no assumptions*)
- **Logical**
- **Honest**
- **Bold**
- **Self-sufficient**

Statistical Quality Assurance

- **Statistical Software Quality Assurance** is a **technique** that measures the *quality* in quantitative fashion.
- It is used to **identify** the potential **variations** in the process and **predict** potential **defects**.
- It **provides** a statistical description of the final product and **addresses quality** and **safety** issues that arise during development.

- **Statistical quality assurance** intimates the following steps:
  - **Step:1** Information about software defects is collected and categorized.
  - **Step:2** An attempt is made to trace each defect to its underlying cause (e.g., nonconformance to specification, design error, violation of standards, poor communication with customer)
  - **Step:3** Using the Pareto principle (80 percent of the defects can be traced to 20 percent of all possible causes), isolate the 20 percent (the “vital few”).
  - **Step: 4** Once the vital few causes have been identified, move to correct the problems that have caused the defects.

Statistical Quality Assurance: Example

- A *software development organization* collects and categories information on defects for a *period of one year*.
  - Some errors are uncovered as software is being developed
  - Other defects are encountered after the software has been released to its end user
- The collected information on defects is categorized as:
  - Incomplete or erroneous specification (IES)
  - Misinterpretation of customer communication (MCC)
  - Intentional deviation from specification (IDS)
  - Violation of programming standards (VPS)
  - Error in data representation (EDR)
  - Inconsistent module interface (IMI)
  - Error in design logic (EDL)
  - Incomplete or erroneous testing (IET)
  - Inaccurate or incomplete documentation (IID)
  - Error in programming language translation of design (PLT)
  - Ambiguous or inconsistent ;human-computer interface (HCI)

Statistical Quality Assurance: Example

Data Collection Statistical SQA

	Total		Serious		Moderate		Minor	
Error	No	%	No	%	No	%	N0	%
IES	205	22%	34	27%	68	18%	103	24%
MCC	156	17%	12	9%	68	18%	76	17%
IDS	48	5%	1	1%	24	6%	23	5%
VPS	25	3%	0	0%	15	4%	10	2%
EDR	130	14%	26	20%	68	18%	36	8%
IMI	58	6%	9	7%	18	5%	31	7%
EDL	45	5%	14	11%	12	3%	19	4%
IET	95	10%	12	9%	35	9%	48	11%
IID	36	4%	2	2%	20	5%	14	3%
PLT	60	6%	15	12%	19	5%	26	6%
HCI	28	3%	3	2%	17	4%	8	2%
MIS	56	6%	0	0%	15	4%	41	9%
Totals	942	100%	128	100%	379	100%	435	100%

Statistical Quality Assurance: Example

- The **collected data** indicates that *IES*, *MCC*, and *EDR* are the *vital few cases* that account for *53 percent* of **all errors**

- *IES*, *EDR PLT*, and *EDL* would be selected as the *vital few cases* if *only serious errors* are considered
  - Once the *vital few cases* are determined, the software development organization can **begin** corrective action

Statistical Quality Assurance: Example

- **Phase Index (PI): Technique** used to develop an *indication* of improvement in **software quality** in each *iteration* of Software Engineering Process (*PI is computed at each iteration*)

Let

Ei = total number of errors uncovered during the i^{th} step in Software Engineering process

Si = number of serious errors

Mi = number of moderate errors

Ti = number of minor errors

Weighting Factors:
ws (serious) = 10
wm (moderate) = 3
wt (trivial) = 1

PS = size of the product (KLOC design statements pages of documentation) at the i^{th} step

$$PI_i = ws(Si/Ei) + wm(Mi/Ei) + wt(Ti/Ei)$$

Statistical Quality Assurance: Example

- **Error Index (EI): Technique** used to develop an *overall indication* of improvement in **Software quality**

Let
Ei = total number of errors uncovered during the i^{th} step (i) in Software Engineering process

Si = number of serious errors

Mi = number of moderate errors

Ti = number of minor errors

Weighting Factors:
ws (serious) = 10
wm (moderate) = 3
wt (trivial) = 1

PS = size of the product (KLOC design statements pages of documentation) at the i^{th} step

Error Index (EI) = $\sum$(i*PIi)/PS = ($1*PI_1 + 2*PI_2 + 3*PI_3 + \ldots + iPI_i$)/KLOC

Statistical Software Quality Assurance

- The **application** of **statistical SQA** can be summarized in a ***single sentence***

"*Spend time focusing on things that, really matter but first be sure that what really matters is understood!*"

Statistical Software Quality Assurance

Solve Phase Index Problem

Find out the **error index** of the software project which has following types and *quantity of error in code*. **Analyze** the results and provide analysis about either the *project is of quality or it is not*?

- 1st phase: Si=40 Mi= 78 Ti=100
- 2nd phase: Si=35 Mi= 12 Ti=90
- Product Size in KLOC= 2000

Weighting Factors:
ws (serious) = 10
wm (moderate) = 3
wt (trivial) = 1