Topic 9
Software Quality: Concepts
(Ch. 14)

Software Engineering Positions Open: 20,000+
### Total Software Quality

#### Quality Assurance Jobs

1 - 25 of 15,000+ jobs found

<table>
<thead>
<tr>
<th>Category</th>
<th>Company</th>
<th>Title</th>
<th>Location</th>
<th>Contact Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Software QA</td>
<td>Stack Overflow</td>
<td>Senior Software QA Engineer</td>
<td>San Francisco, CA</td>
<td>Email</td>
</tr>
<tr>
<td>Software QA</td>
<td>Amazon</td>
<td>Software QA Engineer</td>
<td>Seattle, WA</td>
<td>Phone</td>
</tr>
<tr>
<td>Software QA</td>
<td>Google</td>
<td>Software QA Lead</td>
<td>Mountain View, CA</td>
<td>Email</td>
</tr>
<tr>
<td>Software QA</td>
<td>Microsoft</td>
<td>Software QA Director</td>
<td>Redmond, WA</td>
<td>Phone</td>
</tr>
<tr>
<td>Software QA</td>
<td>Apple</td>
<td>Software QA Manager</td>
<td>Cupertino, CA</td>
<td>Email</td>
</tr>
</tbody>
</table>

**Search Results**

- **Director of Quality Assurance**: [View similar jobs](#)
- **Software QA Analyst**: [View similar jobs](#)
- **Software QA Engineer**: [View similar jobs](#)
- **Quality Assurance Manager**: [View similar jobs](#)
- **Quality Assurance Specialist**: [View similar jobs](#)

**Search Filters**

- **Category**: Software QA
- **Location**: [San Francisco, CA](#), [Seattle, WA](#), [Mountain View, CA](#), [Redmond, WA](#), [Cupertino, CA](#)
- **Contact Method**: Email, Phone, Email, Phone, Email
Total Software Quality: Overall Picture

Software quality can be defined as:
An effective process applied in a manner that creates a useful product that provides measurable value for those who produce it and those who use it.

Software Engineering Framework Activities

CONTRIBUTIONS to TOTAL SOFTWARE QUALITY

Total Software Quality

Total Software Quality, SW Quality Factors (groups), and SW Quality Metrics: A Hierarchical Approach

TSQ = F (SQF1, SQF2, ..., SQFn)
SQFn = P (sqm1, sqm2, ..., sqmk)

A key element of any engineering process is measurement.

Software quality factors and quality metrics provide a quantitative way to access the quality of internal product attributes, thereby enabling the software engineer to project quality BEFORE the final product is built.

Various types of software metrics provide the insight necessary to create effective analysis, reliable design models, solid source code, and thorough tests.
Software Quality Metrics (more than 400)

- Initiation/Communication (getting input initial)
- Planning (resources, time, cost, etc.)
- Analysis/Modeling/Prototyping
  - Analysis of requirements
  - Design Models (Diagrams)
- Development (construction)
  - Code generation
  - Testing
- Deployment/Implementation

Metrics for Software Project Planning
Metrics for Analysis Model
Metrics for Design Model
Metrics for Source Code
Metrics for Testing
Metrics for Software Maintenance

Software Quality Factors
(or, Software Quality Groups)
Software Quality: Main Dimensions
(Software Quality Factors or Software Quality Groups)

David Garvin [1987]:

1. **Performance Quality.**
   Does the software deliver all content, functions, and features that are specified by customer?

2. **Features’ (Functionality) Quality.**
   Does the software provide features that surprise and delight end-users? *(Wow! effect)*

3. **Reliability.**
   Does the software deliver all features and capability without failure? Is it available when it is needed? Does it deliver functionality that is error free?

4. **Conformance (Standards).** Does the software conform to local and external software standards that are relevant to the application?

5. **Durability (Changes).**
   Can the software be maintained (changed) or corrected (debugged) without unintended side effects? Will changes cause the error rate or reliability to degrade with time?

6. **Serviceability (Min service time).**
   Can the software be maintained (changed) or corrected (debugged) in an acceptably short time period. Can support staff acquire all information they need to make changes or correct defects?
Well-Known Approaches to SW Quality

- McCall’s Quality Factors
- ISO 9126 Quality Factors
- Targeted Factors

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.
McCall’s Triangle of Quality

They have tight relationship with ISO 9126 SW Quality Factors (next slide)
McCall’s Software Quality Factors

- **Correctness**: extent to which a program satisfies its specification and fulfills the customer’s mission objectives.
- **Reliability**: extent to which a program can be expected to perform its intended function with required precision.
- **Efficiency**: amount of computing resources and code required by a program to perform its function.
- **Integrity**: extent to which access to software or data by unauthorized persons can be controlled.
- **Usability**: effort required to learn, operate, prepare input for, and interpret output of a program.
- **Maintainability**: effort required to locate and fix an error in a program.
- **Flexibility**: effort required to modify an operational program.
- **Testability**: effort required to test a program to ensure that it performs its intended function.
- **Portability**: effort required to transfer the program from one hardware and/or software system environment to another.
- **Reusability**: extent to which a program [or parts of a program] can be reused in other applications.
- **Interoperability**: effort required to couple one system to another.

Examples of Formulas to Calculate Software Quality Factors

1. **Correctness**:
   - Extent to which a program satisfies its specifications and fulfills the user’s mission objectives. It can be calculated as:
     \[ \text{Correctness} = \frac{\text{No. of non-conformance to requirements}}{\text{Total no. of requirements}} \times 100 \]
   - Correctness can also be calculated as:
     \[ \text{Correctness} = \frac{\text{No. of requirements fulfilled}}{\text{Total no. of requirements}} \times 100 \]

2. **Reliability**:
   - Extent to which a program can be expected to perform its intended function with required precision. The formula of reliability is:
     \[ \text{Reliability} = \frac{\text{Mean Time To Failure}}{\text{Total Run Time}} \times 100 \]
   - It can also be calculated as:
     \[ \text{Reliability} = \frac{\text{Mean Time Between Failure}}{\text{Total Run Time}} \times 100 \]

Examples of Formulas to Calculate Software Quality Factors

3. Efficiency:
   - The amount of computing resources and code required by a program to perform a function.
   - Efficiency
     \[
     \text{Efficiency} = \frac{\text{Memory Usage}}{\text{Total Memory}} \times 100
     \]

4. Integrity:
   - Extent to which access to software or data by unauthorized persons can be controlled.
   - Integrity
     \[
     \text{Integrity} = \frac{\text{No. of successful attempts}}{\text{Total no. of attempts}} \times 100
     \]

5. Usability:
   - Effort required learning, operating, preparing input, and interpreting output of a program.
   - Usability
     \[
     \text{Usability} = \frac{\text{Total Training Time}}{\text{Total development time}} \times 100
     \]

6. Maintainability:
   - Effort required locating and fixing an error in an operational program.
   - Maintainability
     \[
     \text{Maintainability} = \frac{\text{Time spent to fix a bug}}{\text{Total development time}} \times 100
     \]

7. Testability:
   - Effort required testing a program to ensure that it performs its intended function.
   - Testability can be calculated as
     \[
     \text{Testability} = \frac{\text{Time spent in testing the functionality}}{\text{Total development time}} \times 100
     \]

8. Flexibility:
   - Effort required modifying an operational program.
   - Flexibility
     \[
     \text{Flexibility} = \frac{\text{Time spent to fix a bug}}{\text{Total development time}} \times 100
     \]

Examples of Formulas to Calculate Software Quality Factors

9. Portability:
- Portability is the software code base feature to be able to reuse the existing code instead of creating new code when moving software from an environment to another.
- Portability
  \[ \text{Portability} = \frac{\text{No. of successful ports}}{\text{Total no. of ports}} \times 100 \]

10. Reusability:
- Extent to which a program can be used in other applications. It is related to the packaging and scope of the functions that programs perform.
- Reusability
  \[ \text{Reusability} = \frac{\text{No. of reusable components}}{\text{Total no. of components}} \times 100 \]

11. Interoperability:
- Effort required to couple one system with another.
- Interoperability
  \[ \text{Interoperability} = \frac{\text{Time spent in coupling the system}}{\text{Installation Time}} \times 100 \]


Software Quality Factors: an example - Software Reliability
Software Reliability

- **Software Reliability** is defined as the probability of failure free operation of a software program in a specified environment for a specified time period.

- Can be measured directly and estimated using historical and developmental data (unlike many other software quality factors).

- **Measures of Software Reliability (examples):**
  - Time to Failure (TTF)
  - MTTF = mean time to failure (mean of given values)
  - MTTR = mean time to repair
  - Mean time between failure (MTBF) = MTTF + MTTR
  - OS Availability = [(MTTF) / (MTTF + MTTR)] x 100%

- Number of units under review: ONE or MANY

![Software Reliability vs Hardware Reliability](http://www.ece.cmu.edu/~koopman/des_s99/sw_reliability/)

<table>
<thead>
<tr>
<th>DATA REFERENCE</th>
<th>PREDICTION MODELS</th>
<th>ESTIMATION MODELS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Usually made prior to development</td>
<td>Usually made after all units are in use</td>
<td>Usually made late in life, preferably after all data has been collected, or after units are in service.</td>
</tr>
<tr>
<td>WHEN USED IN DEVELOPMENT CYCLE</td>
<td>DEVELOPMENT PERIOD</td>
<td>SERVICE LIFE PERIOD</td>
</tr>
<tr>
<td>TIME FRAME</td>
<td>PREDICT RELIABILITY AT SOME FUTURE TIME</td>
<td>ESTIMATE RELIABILITY AT SOME PRESENT OR SOME FUTURE TIME</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>#</th>
<th>TTF</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8376.00</td>
</tr>
<tr>
<td>2</td>
<td>2952.00</td>
</tr>
<tr>
<td>3</td>
<td>3312.00</td>
</tr>
<tr>
<td>4</td>
<td>2616.00</td>
</tr>
<tr>
<td>5</td>
<td>1152.00</td>
</tr>
<tr>
<td>6</td>
<td>888.00</td>
</tr>
<tr>
<td>7</td>
<td>1896.00</td>
</tr>
<tr>
<td>8</td>
<td>3552.00</td>
</tr>
<tr>
<td>9</td>
<td>4824.00</td>
</tr>
<tr>
<td>10</td>
<td>3384.00</td>
</tr>
<tr>
<td>11</td>
<td>5286.00</td>
</tr>
<tr>
<td>12</td>
<td>336.00</td>
</tr>
</tbody>
</table>

OS Failure Data from over 5 years (unit of measure = in hour)
Software Reliability: An Example

- **Important Issue:**
  Number of units under review: ONE or MANY

ISO Chapter 9126
ISO 9126 Six Key Software Quality Factors
(or, Groups of Quality Metrics)

1. Functionality (incl. 5 attributes)
2. Reliability (incl. 3 attributes)
3. Usability (incl. 3 attributes)
4. Efficiency (incl. 2 attributes)
5. Maintainability (incl. 4 attributes)
6. Portability (incl. 4 attributes)

In general, each attribute may be characterized by 1...10...20...50 various technical software metrics (for example, testability).

- Metrics for Analysis Model.
- Metrics for Design Model.
- Metrics for Source Code.
- Metrics for Testing.
ISO 9126 Six Key Groups of Software Quality Factors (Groups of Metrics)

Software Quality Dilemma

- IF you produce software with terrible quality
- THEN you lose because no one will use it

- IF you spend a lot of time AND a lot of money to create software system
- THEN you lose because you will be late on market or even go out of business without bringing the software to market

The trick is to balance
- 1) the development (construction and testing) time and costs, and
- 2) the software product quality
The Bottom Line: Software System’s Costs (an example)

The relative costs to find and repair a software defect (bug, flaw) increase dramatically as we go from prevention to detection to internal failure to external failure costs.

- **External failure costs** are:
  - complaint resolution
  - product return and replacement
  - help line support
  - warranty work

- **Internal failure costs** include:
  - rework
  - repair
  - failure mode analysis

- **Prevention costs** include:
  - quality planning
  - formal technical reviews (FTRs)
  - test procedures and equipment
  - training

Wow!!! 100+ times difference per a SW defect !!!

Achieving Software Quality

- Software quality is the result of good project management and solid engineering practice

- **Software Engineering methods**: to build high quality software you must understand the problem to be solved and be capable of creating a quality analysis, architecture and design models using various types of SW metrics.

- **Project Management techniques (CS 591)**: project plan includes explicit techniques for quality and change management

- **Quality Control (Ch. 15)**: series of inspections, reviews, and tests used to ensure conformance of a work product to its specifications

- **Quality Assurance (Ch. 16)**: consists of the auditing and reporting procedures used to provide management with data needed to make proactive decisions
Software Quality Assurance (SQA): An Overview

- **Software quality assurance (SQA)** is the concern of every software engineer to reduce costs and improve product time-to-market.

- **Use of Software Engineering metrics** is an important part of developing a strategy to improve the quality of both software processes and work products.

- Phil Crosby once said: The problem of quality management is not what people don’t know about it. The problem is what they think they do know”.

- **A Software Quality Assurance Plan** is not merely another name for a test plan, though test plans are included in an SQA plan. SQA activities are performed on every software project.
What is Software Quality Assurance (SQA) 
or Software Quality Management (SQM) 

SQA Goals

Requirements (analysis) quality
- Elimination of Ambiguity
- Completeness
- Volatility
- Traceability
- Model clarity

Design quality
- Architectural integrity
- Component completeness
- Interface complexity
- Use of Patterns

Code quality
- Code quality + optimization
- Complexity
- Maintainability
- Understandability
- Reusability
- Documentation

Quality control effectiveness
- Resource allocation
- Completion rate
- Review effectiveness
- Testing effectiveness
Main Components (Elements) of SQA

- **Standards**
  - ensure that standards are adopted and followed

- **Reviews and audits**
  - audits are reviews performed by SQA personnel to ensure that quality guidelines are followed for all software engineering work

- **Testing**
  - ensure that testing is properly planned and conducted

- **Error/defect collection and analysis**
  - collects and analyses error and defect data to better understand how errors are introduced and can be eliminated

- **Changes Management**
  - ensures that adequate change (software increment, version, or generation) management practices have been instituted (for ex. move from Word 2003 to Word 2007)

- **Security Management**
  - ensures use of appropriate process and technology to achieve desired security level

- **Education**
  - takes lead in software process improvement and educational program (ex: a resistance from corp. users)

- **Risk Management**

- **Vendor Management**
  - suggests specific quality practices vendor should follow and incorporates quality mandates in vendor contracts

---

Statistical SQA

- **Information about software errors and defects is collected and categorized.**

- **An attempt is made to trace each error and defect** to its underlying cause (e.g., non-conformance to specifications, design error, violation of standards, poor communication with the customer).

- **Using the Pareto Principle for Software Engineering / Software Development**

- Once the vital few causes have been identified, move to correct the problems that have caused the errors and defects.
Pareto Principle and Software Engineering

- Attributed to a 19th century economist by the name of Vilfredo Pareto, who realized that 80% of the wealth in a given population was concentrated in the hands of 20% of the population, the observation has found its way into many disciplines.

- Using the Pareto principle in SE/SD:
  a) 80 percent of the defects can be traced to 20 percent of all possible causes;
  b) 20 percent of software modules will produce about 80 percent of bugs;
  c) 80 percent of users will use 20 percent of functions;
  etc.

Total Quality Management (TQM)

- Total Quality Management (TQM) describes the culture, attitude and organization of a company that strives to provide customers (internal and external) with products (incl. software systems) and services that satisfy their needs.

- TQM is a combination of quality tools and management specific tools to achieve increased business while reducing costs and waste.

Six Sigma Methodology for TQM (Total Quality Management)

- A defect is any instance where the product or service fails to meet customer requirements.

- The main purpose of the Six Sigma quality methodology is to reduce variation, thus reducing the number of product or service defects. It uses data and statistical analysis to measure and improve a company’s operational performance by identifying and eliminating defects.

- The Greek letter Sigma (σ) is used in the field of statistics to represent the standard deviation to measure variability from the mean or average.

- A small standard deviation means that data cluster closely around the middle (mean) and there is little variability among the data.

- A normal distribution is a bell-shaped curve that is symmetrical about the mean.

Six Sigma in SW Development Projects

- Six Sigma for SW D&D projects is calculated based on the number of defects per million opportunities.

- To reach Six Sigma level of quality, you would have no more than 3.4 defects per million opportunities on your IT projects, for example, no more than 3.4 bugs per million of LOC (lines of code).
Six Sigma: Just 99.00 % vs. 6 Sigma (99.99966%)  

<table>
<thead>
<tr>
<th>3.8 σ</th>
<th>6 σ</th>
</tr>
</thead>
<tbody>
<tr>
<td>99% Good</td>
<td>99.99966% Good</td>
</tr>
<tr>
<td>20,000</td>
<td>Seven</td>
</tr>
<tr>
<td>15 minutes per day</td>
<td>One minute per seven months</td>
</tr>
<tr>
<td>5,000</td>
<td>1.7</td>
</tr>
<tr>
<td>Two per day</td>
<td>One every five years</td>
</tr>
<tr>
<td>200,000</td>
<td>88</td>
</tr>
</tbody>
</table>

Source: http://www.aqsn.com/LeanGreen-Six%20Sigma_040509_PDF.pdf

Design For Six-Sigma (DFSS) and Motorola

“Right now the culture is to test in quality. But we’re asking our engineers to design in quality.”

Motorola, which led the way in Six Sigma for manufacturing, is also an early adopter of Six Sigma for software. “Right now the culture is to test in quality. But we’re asking our engineers to design in quality,” said Tricia McNair, director of Motorola’s Software Design for Six Sigma (SDFSS) program and chairman of the Software Development Consortium and Six Sigma Software Academy.

**Goals:** To address major performance flaws in requirements and architecture  
- To improve on-time delivery and reduce defects  
- To improve robust design and performance benchmarking  
- To employ metric-based decision making for product development  
- To utilize templates and criteria to support rigorous gate reviews  
- To enable leadership to realistically set high expectations and demand evidence  
- To provide tools, tasks and deliverables that clarify and support meeting those high expectations

The term “six sigma” is derived from six standard deviations — 3.4 instances (defects) per million occurrences — implying an extremely high quality standard.
The term “six sigma” is derived from six standard deviations — 3.4 instances (defects) per million occurrences — implying an extremely high quality standard.

The Six Sigma methodology defines three core steps:

- **Define** customer requirements and deliverables and project goals via well-defined methods of customer communication
- **Measure** the existing process and its output to determine current quality performance (collect defect metrics)
- **Analyze** defect metrics and determine the vital few causes.

If an existing SW process is in place, but improvement is required, then 2 additional steps should be performed:

- **Improve** the process by eliminating the root causes of defects.
- **Control** the process to ensure that future work does not reintroduce the causes of defects.
Software Quality
Positions

Elective Courses  (as Directed Individual Studies – DIS – courses)

1) Software Total Quality Management (TQM)  
   (incl. TQM, SQA, Six Sigma, CMMI )

2) Design for Six Sigma (DFSS)

3) CMMI (Capability Maturity Model Integrated)

available during Jan and/or May interims  and Summer sessions
Topic 9
Software Quality Control (Ch. 15) – OPTIONAL

Topic 9
Software Quality Concepts and Assurance: Additional Information
The Software Quality Dilemma

- If you produce a software system that has terrible quality, you lose because no one will want to buy it.

- If on the other hand you spend infinite time, extremely large effort, and huge sums of money to build the absolutely perfect piece of software, then it's going to take so long to complete and it will be so expensive to produce that you'll be out of business anyway.

- Either you missed the market window, or you simply exhausted all your resources.

- So people in industry try to get to that magical middle ground where
  a) the software product is good enough not to be rejected right away, such as during evaluation, but also
  b) not the object of so much perfectionism and so much work that it would take too long or cost too much to complete.

Quality and Security

Gary McGraw comments [Wil05]:

Software security relates entirely and completely to quality. You must think about security, reliability, availability, dependability—
at the beginning, in the design, architecture, test, and coding phases, all through the software life cycle (process).

Even people aware of the software security problem have focused on late life-cycle stuff.

The earlier you find the software problem, the better.

And there are two kinds of software problems,
1) **One is bugs**, which are implementation problems (wrong coding, wrong technical platform, etc.)
2) The other is **software flaws** - architectural problems in the design.

People pay too much attention to bugs and not enough on flaws.”
“Good Enough” Software

- Good enough software delivers high quality functions and features that end-users desire, but at the same time it delivers other more obscure or specialized functions and features that contain known bugs.

- Arguments against “good enough” approach:
  - It is true that “good enough” may work in some application domains and for a few major software companies. After all, if a company has a large marketing budget and can convince enough people to buy version 1.0, it has succeeded in locking them in.
  - If you work for a small company be wary of this philosophy. If you deliver a “good enough” (buggy) product, you risk permanent damage to your company’s reputation.
  - You may never get a chance to deliver version 2.0 because bad buzz may cause your sales to plummet and your company to fold.
  - If you work in certain application domains (e.g., real time embedded software, application software that is integrated with hardware) can be negligent and open your company to expensive litigation.

Software Engineering Metrics: Main Areas

1. **Metrics for Analysis Model.**
2. **Metrics for Design Model.**
3. **Metrics for Source Code.**
4. **Metrics for Testing.**
Sections of Software Quality Assurance (SQA) Plan

- **Management section** - describes the place of SQA in the structure of the organization
- **Documentation section** - describes each work product produced as part of the software process
- **Standards, practices, and conventions section** - lists all applicable standards/practices applied during the software process and any metrics to be collected as part of the software engineering work
- **Reviews and audits section** - provides an overview of the approach used in the reviews and audits to be conducted during the project
- **Test section** - references the test plan and procedure document and defines test record keeping requirements
- **Problem reporting and corrective action section** - defines procedures for reporting, tracking, and resolving errors or defects, identifies organizational responsibilities for these activities
- **Other** - tools, SQA methods, change control, record keeping, training, and risk management