HOW TO SPECIFY QUALITY ATTRIBUTES IN YOUR SOFTWARE REQUIREMENTS

SPECIFICATION

by

Boniface C. Nwugwo

October 21, 1998
Abstract

In keeping with the saying, "you can't achieve quality unless you specify it," we must be able to specify what it is we mean when we say for example, that the software should be maintainable, reusable, or expandable. This paper provides a method for specifying software quality requirements. Given any quality factor that one chooses to have in their software, a set of criteria to meet those quality factors can be implemented to meet such software quality attributes.
Introduction

What do you look for in terms of quality when you are shopping for a new automobile? Is a low maintenance cost vehicle important to you? Is it leather seat covers? Touch-sensitive command console? Resale value? Or is it Safety?

It probably depends on how you want to use the automobile, who will drive it, how much you're willing to spend, etc. Quality means different things to different people. If your need is to have a reliable car for traveling back and forth to work, obviously your needs are different from someone who wants to chauffeur dignitaries to corporate headquarters.

The analogy is true for software as well. Depending on the intended use of a piece of software, cost, environment and other factors, levels and type of quality needs of the customer will differ. Often times we state in our requirement documents the customer's perception of the quality attributes the software will exhibit without specifying how to achieve them. In keeping with the saying, "you can't achieve quality unless you specify it," we must be able to specify what it is we mean when we say for example, that the software should be maintainable, reusable, or expandable.

The following examples of a software quality requirement specification for Flexibility, Expandability, Maintainability,
and **Reusability** were written in accordance with DOD-STD-2167's data item description for software requirement specifications (DI-MCCR-80025). They are excerpts from "Software Quality Engineering: A total technical and management approach" by Michael S. Deutsch and Ronald R. Willis. The list was compiled by Boniface C. Nwugwo, Washington Development Center, Eastman Kodak Company.

The Table below is a mapping between User needs for quality (Quality Factors) and the Criteria that must be designed in during development to meet those quality attributes. Based on this mapping, we are able to specify what levels of quality should be implemented during design to meet certain levels of software quality requirements.

**Definitions**

First, some definitions of terms used. For the purposes of this paper, unless the context indicates otherwise, the following definitions apply:

*Cohesion* is a measure of the strength of association of elements within a module.

*Common Coupling* -- Modules are common coupled if they share data that was not passed along a normal module call (e.g., data held in a common area).
Computer Software Component (CSC) -- A manageable subpart of the whole software program that could have an entire software requirements specification written for it.

Computer Software Configuration Item (CSCI) -- A collection of CSCs treated as a unit for the purpose of configuration management.

Control Coupling -- A couple is a control couple if it affects the flow of control in a receiver. Two modules are control coupled if they communicate using at least one control couple.

Coupling is the measure of interdependence of one module on another. We can measure this interdependence by examining the interfaces between modules.

Data Coupling is merely the necessary data communication between modules.

Functional Cohesion -- A module is functionally cohesive if all of the elements contribute to one and only one - complete task.

Higher Order Language (HOL) -- A programming language that usually includes features such as nested expressions, user defined data types, and parameter passing not normally found in lower order languages.

Stamp Coupling -- Two modules are stamp-coupled if they communicate using at least one data structure.
Specifying Software Quality Requirements

The following requirements specify static attributes of designs, code and tests that can be observed during the software development process to ensure that the specified quality criteria are achieved. For each quality factor, there is at least one quality criterion that is applicable.

**Expandability Requirements** - Expandability is the extent of effort required to expand (add new) software capabilities or performance. It deals with the perfective aspects of software maintenance, that is, increasing the software's functionality or performance to meet new needs.

**Flexibility Requirements** - Flexibility is the extent of effort required to change (modify existing) software to accommodate changes in requirements. It deals with the adaptive aspects of software maintenance, that is, modifying the software to work in different environments.

**Maintainability Requirements** - Maintainability is the extent of effort required to find and fix errors in the CSCI. It deals with the ease of finding and fixing errors. (Note that according to this definition, maintainability does not entail adaptive or perfective maintenance; see flexibility and expandability for these)

**Reliability Requirements** - Reliability is the extent to which the CSCI consistently performs the functions specified in the Functional Specifications section of the Software Requirements Specification (SRS). It deals with the rate of failures in the software that renders it unusable. Some common failures are that the software is not accurate enough, that it gives incorrect results that the response time is too slow or the software "hangs up".

**Reusability Requirements** - Reusability is the extent of effort required to convert a portion of this CSCI for use in another application. It deals with the use of portions of the software for other applications. It includes the use of mathematical libraries in both statistical and scientific applications.
<table>
<thead>
<tr>
<th>QUALITY FACTORS</th>
<th>Correctness</th>
<th>Reliability</th>
<th>Efficiency</th>
<th>Integrity</th>
<th>Usability</th>
<th>Maintainability</th>
<th>Testability</th>
<th>Flexibility</th>
<th>Portability</th>
<th>Reusability</th>
<th>Interoperability</th>
<th>Survivability</th>
<th>Expandability</th>
<th>Safety</th>
<th>Verifiability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accuracy</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Anomaly Management</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application Independence</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Augmentability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Commonality</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Completeness</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Consistency</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Distributivity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency of communication</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency of processing</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Efficiency of storage</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional overlap</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Functional scope</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Generality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Independence</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modularity</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operability</td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Reconfigurability</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Safety management</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Self-descriptiveness</td>
<td></td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Simplicity</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System accessibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>System clarity</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>System compatibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Traceability</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Training</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Virtuality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Visibility</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Quality Criteria Requirements

Accuracy - Accuracy is the software characteristic that provides the required precision in calculations and outputs.

a. The specified accuracy requirements for individual functions shall be allocated to CSCs and units.

b. The specified quantitative accuracy requirements for outputs from functions shall be implemented.

c. The outputs associated with CSCs and units shall have enough precision to meet the specified accuracy requirements.

d. Existing (i.e., off-the-shelf) mathematical library CSCs and Units planned for use shall be implemented with enough precision to meet the specified accuracy requirements.

e. Numerical techniques used in implementing CSCs and Units shall provide enough precision to meet the specified accuracy requirements.

Anomalty Management - Anomaly management is the software characteristic that ensures continuity of operations under and recovery from abnormal conditions.

a. Concurrent tasks that require synchronization shall be centrally controlled.

b. Critical inputs from interfacing CSCIs and external systems shall be checked with respect to their specified range prior to their use.

c. Critical inputs from interfacing CSCIs and external systems shall be checked with respect to specified conflicts prior to their use.

d. Critical inputs from interfacing CSCIs and external systems shall be checked with respect to specified reasonableness criteria prior to their use.
e. Critical inputs from interfacing CSCIs and external systems shall be checked with respect to specified invalid combinations prior to their use.

f. The design of the CSCI shall include fault-tolerant mechanisms, which establish alternative means for continued error-free system operation within response-time requirements in the presence of detected software errors.

g. The design of the CSCI shall include fault containment mechanisms that prevent the propagation of failures resulting from individually or multiply detected software errors.

h. Detection of and recovery from computational errors in units shall be provided.

i. Central control shall be used for concurrent or redundant processing of tasks, which are required to execute more than once for comparison purposes.

j. The design of the CSCI shall include fault-tolerant mechanisms, which provide uninterrupted service in the presence of individually or multiply detected software errors.

k. The design of the CSCI shall include recovery from detected hardware and software errors (i.e., arithmetic faults, hardware failures, clock interrupts, I/O device errors, and communication transmission errors).

l. For all detected errors, the time of occurrence, the unit in which the error occurred and its calling unit, and the data elements associated with the error shall be recorded when detected.

m. For processing that is dependent on data received from interfacing CSCIs or external systems, a check shall be performed before the processing begins to determine that the data are available.

n. Critical data output to interfacing CSCIs and external systems shall be checked for reasonable values prior to outputting.

o. For critical units, all control variables and array indices shall be checked for out-of-range values prior to their use.
The design of the CSCI will assure that the CSCI is initialized to a correct state upon recovery from a detected fault and that processing is continued after recovery.

The design of the CSCI will include detecting when a task has exceeded predetermined execution time limits and taking remedial action.

All detected errors shall be reported to the operator.

**Augmentability** - Augmentability is the software characteristic that ensures expansion of capability for functions and data. CSCs shall be partitioned to be logically complete and self-contained (i.e., functionally cohesive).

- The specified spare memory requirements shall be met or exceeded.
- The specified spare auxiliary storage requirements shall be met or exceeded.
- The specified spare CPU utilization requirements shall be met or exceeded.
- The specified spare I/O channel utilization requirements shall be met or exceeded.
- The specified spare communication channel utilization requirements shall be met or exceeded.
- Where practical, commercial or reusable software will be utilized to meet the requirements of the CSCI.
- Provision will be made in unit source code to accommodate additional functions and new equipment.
- Provision will be made in physical database to accommodate additional functions, new equipment, and new data.
- The design of the CSCI shall include software that tests the operating system and the communication links, memory devices and peripheral devices.
- The number of units performing physical layer protocol processing for a hardware device interface shall not exceed two (one for input and one for output).
k. CSCs and units that handle hardware and device interface protocol shall not include unrelated processing.

l. Variable dimensions and sizes of dynamic arrays shall be defined parametrically.

m. Data base references by units shall be symbolic.

n. Application software shall be independent of the specific details of the underlying data base structure.

Completeness - Completeness is the software characteristic that ensures full implementation of the functions required.

a. All specified requirements of the CSCI shall be allocated to CSCs of the CSCI.

b. Input, processing, and output requirements of each CSC and unit shall be defined in accordance with specified standard.

c. Each defined data item in each CSC and unit shall be set or used.

d. Global and local data shall bear comments in CSCI design documentation with regard to purpose and format.

e. The top-level and detailed designs of the CSCI shall be complete in themselves.

f. Conditions and alternative processing options shall be defined and documented for each decision point for all units.

g. A complete flow of data and execution control within each CSC down to the unit level shall be determined in the detailed design.

h. The detailed design shall include the logic to be employed and the algorithms and interrupt capabilities to be implemented in each unit.

i. All developed units shall be tested in at least one CSC integration test.
**Consistency** - Consistency is the software characteristic that ensures uniform design and implementation techniques and notations.

a. References to the same CSC shall use a single unique name, the CSC designator.

b. The naming of global and local databases and formal parameters within CSCs and units shall comply with specified standards.

c. References to the same data files or items within CSCs and units shall use a single unique name.

d. Data representation in CSCs and units shall comply with a specified standard.

e. The peripheral I/O protocol and format shall comply with a specified standard.

f. The handling of detected error conditions shall comply with a specified standard (e.g., formats for error messages and diagnostic messages).

g. The definition and use of global data base items shall be in accordance with a specified standard.

h. The calling sequence protocol between CSCs and between units shall comply with a specified standard.

i. The data base management design shall provide a common and controlled approach to adding new data and to modifying and retrieving existing data from the global data bases.

j. References to a unit shall use the unit name or its assigned abbreviation.

k. Units shall be implemented in accordance with specified programming standards.

l. Comments associated with executable source code shall be uniformly indented.

m. A single Program Design Language (PDL) shall be used in all unit design representations.
**Functional Scope** - Functional scope is the commonality of functions within a CSCI.

a. CSC and unit inputs shall be documented in the design as to the specific meaning and limitations of the data.

b. A description of the function(s) of a unit shall be provided in the unit's comments.

**Generality** - Generality is the characteristic of software that ensures the breadth of the functions performed with respect to the application.

a. Units will be designed to be common units where practical.

b. Common subprograms shall not mix any of the following processing categories:

   - CSCI input, CSCI output, Algorithmic processing.

   c. Common units will be free from strict limitations on the number of data items processed (e.g., the data number limits shall be parameterized).

   d. Common units will be free from strict limitations on the values of input data (e.g., error tolerances, range tests, and reasonableness checks should be parameterized).

**Independence** - Independence is the software characteristic that ensures that it does not depend on its environment (the computing system, operating system, utilities, I/O routines, and libraries).

a. Application software shall not rely on specific architectural details of lower level software and hardware.

b. Developed code shall be regenerative and maintainable using only existing or delivered support software.

c. The CSCI design shall provide logical and physical data independence for global data.
d. The number of units containing operations dependent on word or character size will be minimized.

e. The number of units containing data item representations that are machine dependent will be minimized.

f. Unit code constructs shall use specified coding standards (i.e., code will be free from nonstandard constructs of the specified programming language).

g. Unit references to services unique to the operating system or language implementation (e.g., environment-dependent library routines and utilities) will be minimized.

h. Assembly language shall be used only when specified memory or processing performance requirements cannot be met with the use of the specified programming language, and each such use shall be justified in a waiver request.

Modularity - Modularity is the characteristic of software that ensures a highly cohesive component structure with optimum coupling.

a. The design of the CSCI will minimize the use of control variables as formal parameters.

b. The design of the CSCI will result in CSC and unit interfaces that exhibit the following types of coupling in the following order: data, stamp, control, external, common (with data being the most desirable).

c. The design of the CSCI shall be partitioned into one or more CSCs which, in turn, shall be partitioned into one or more units.

d. The CSCI design representation will be comprised of successive independent levels of abstraction, i.e., each level will be complete and independent, and each level will contain complete definitions of data and operations on those data.

e. Items in each logical file of the database shall be functionally dependent at the identifier level (functional cohesion).
f. The design of the CSCI will result in CSCs and units that exhibit the following types of cohesion in the following order: functional, informational, communicational, procedural, classical, logical, and coincidental (with functional being the most desirable).

g. Unit interfaces will be implemented in such manner as to minimize coupling (i.e., interfaces will be parsimonious).

h. Data input to a called unit shall be passed to the unit through formal parameters or through the unit's access to global data items.

i. Data output from a called unit shall be passed back to the calling unit through formal parameters or through updates to global data items.

j. A subprogram shall consist of not more than 200 executable HOL statements.

k. When (either normal or exceptional) execution is completed, a called unit shall return control to the calling unit.

l. Each unit shall be separately compilable.

m. Each unit shall consist of a Specification, Data declarations, and a Sequence of executable statements.

Self-Descriptiveness - Self-descriptiveness is the characteristic of software that ensures explanation of the implementation of functions.

a. The CSCI design representation will explicitly document the results of design decisions.

b. The design description of a CSC shall identify interfacing CSCs, CSCIs, and external systems.

c. A standard method shall be utilized for comments accompanying global data within a unit. Such comments shall include both meaning (i.e., what the item is) and limitations.

d. A standard method shall be utilized for comments accompanying parameter input and output, and local variables
within a unit. Such comments shall include both meaning (i.e., what the item is) and limitations.

e. Unit prologue comments that contain all information in accordance with an established standard shall exist.

f. The identification and placement of comments within a unit shall be in accordance with an established standard.

g. Machine-dependent code in a unit shall bear comments to the effect that it is machine-dependent.

h. HOL statements within a unit that do not comply with an established standard shall bear comments.

i. Comments related to a unit's operations shall describe the purpose or intent of those operations.

j. No keywords shall be used as variable names.

k. Unit variable names will be descriptive of the physical or functional property they represent.

l. Source code within a unit, excluding comments, will be free from continuation lines.

m. Source code shall be blocked and indented to denote logical levels of constructs.

n. Source code shall have comments to explain inputs, outputs, branches, conditional statements, case statements, loop statements, and other features that are not obvious in the code.

o. Data names and labels within units will be meaningful.

**Simplicity** - Simplicity is the software characteristic that ensures definition and implementation of functions in the most direct and understandable way.

a. The number of unit accesses (i.e., the utilization) of common data blocks and global data items will be minimized.

b. The unit design shall permit only internal procedures to access a unit's data, while restricting other units to formal interface access.
c. Macros, procedures, functions, and other such reusability packages will be used to avoid repeated and redundant code within CSCs and units.

d. Units shall be implemented according to an established programming standard

e. The flow of control within a unit will be from top (i.e., point of entry) to bottom.

f. The use of negative Boolean and compound Boolean expressions in unit source code will be minimized.

g. Unnatural exits (e.g., jumps and returns) from loops will be minimized.

h. Block nesting levels in the unit source code beyond three (3) levels will be avoided; nesting beyond five (5) levels shall be avoided.

i. Each data item in unit source code shall be used in accordance with its description in an accompanying comment (i.e., the item serves only one purpose).

j. Except for comments, nonexecutable statements shall be grouped in one area in each unit.

k. Unit data declarations will be grouped and arranged in a meaningful order in the source code (e.g., in a columnar arrangement rather than horizontally).

l. Except for data declarations, each line of source code shall contain at most, one executable language statement.

m. Iteration loop indices will not be explicitly modified in the source code within the loop.

n. The unit will be free from self-modifying code.

o. Except for exit points for exception handling, units shall have a single entry point and a single exit point.

**Traceability** - Traceability is the software characteristic that provides a thread of origin from the implementation to the requirements with respect to the specified development envelope and operational environment.
a. The design of the CSCI will be represented in a manner, which facilitates traceability to its specification.

b. The CSCI implementation shall conform functionally and structurally to the software design as specified in the design documentation.

**Virtuality** - Virtuality is the characteristic of software that provides representation of different physical components by the same logical or virtual component.

a. The design of the CSCI will present a logical (not physical) view of the system to the user.

b. The CSCI will isolate I/O interfaces to single units.

c. The design of the CSCI will maximize use of logical record interfaces.

**Visibility** - Visibility is the characteristic of software that provides monitoring of the development and operation of the software.

a. The CSCI will include rapid and positive detection and reporting of hardware and software malfunctions, intermittent errors, and marginal performance of the software.

b. The CSCI will isolate defective components and tasks to facilitate

References
