FDA Validation Document Summary

Source: General Principles of Software Validation

Kwangmin Kim

2022-12-20

Table of contents

1 Information on the Document

1.1 Source

FDA: General Principles of Software Validation

1.2 Rationale

FDA has reported the following analysis:

- 242 of 3140 (7.7%) medical device recalls between 1992 and 1998 are attributable to software failures.
- 192 of the 242 (79.3%) failures were caused by software defects that were introduced when changes were made to the software after its initial production and distribution.
- The software validation check is a principal means of avoiding such defects and resultant recalls.

1.3 Main Institutions

- Center for Devices and Radiological Health (CDRH)
- U.S. Department Of Health and Human Services
- Food and Drug Administration
- Center for Biologics Evaluation and Research

2 Document Summary

2.1 Purpose

The purpose is to make a sketch of general validation principle of the validation of medical device software or software used to design or develop.

2.2 Scope

The scope of this guidance is broad. The important activities for the software validation include at least:

- planning,
- verfication,
- testing,
- traceability, and
- configuration management.

All of the activities above should be

- integrated
- be able to describe software life cycle management and
- be able to describe software risk management.

The software validation and verification activities should be focused into the entire software life cycle. (It does not necessarily mean that the activies must follow any technical models.)

The guidance is applicable to any software related to a regulated medical device and anyone who is employed in a bio or medical industry.

2.2.1 The Least Burdensome Approach

The guidance reflects that *the minimum list of the relavant scientific and legal requirements* that you must comply with.

2.2.2 Regulatory Requirements for Software Validation

- Software validation: a requirement of **the Quality System regulation**, which was published in the Federal Register on October 7, 1996 and took effect on June 1, 1997. (See Title 21 Code of Federal Regulations (CFR) Part 820, and 61 Federal Register (FR) 52602, respectively.)
- Specific requirements for validation of device software are found in 21 CFR §820.30(g). Other design controls, such as planning, input, verification, and reviews, are required for medical device software. (See 21 CFR §820.30.)
- computer systems used to create, modify, and maintain electronic records and to manage electronic signatures are also subject to the validation requirements. (See 21 CFR §11.10(a).)

2.2.2.1 Objective

The objective of software validation is to ensure:

- accuracy
- reliability
- consistent intended performance, and
- the ability to discern invalid or altered records.

2.2.2.2 What to validate

Any software used to automate device design, testing, component acceptance, manufacturing, labeling, packaging, distribution, complaint handling, or to automate any other aspect of the quality system, including any off-the-shelf software.

2.2.3 Quality System Regulation vs Pre-market Submissions

This document **does not address** any specific requirements **but** general ones. Specific issues should be addressed to

- the Office of Device Evaluation (ODE),
- Center for Devices and Radiological Health (CDRH)
- the Office of Blood Research and Review,
- Center for Biologics Evaluation and Research (CBER). See the references in Appendix A for applicable FDA guidance documents for pre-market submissions.

2.3 Context for Software Validation

- Validation elements that FDA expects to do for the Quality System regulation, using the principles and tasks are listed in Sections 4 and 5.
- Additional specific information is available from many of the references listed in Appendix A

2.3.1 Definition and Terminology

The medical device Quality System regulation (21 CFR 820.3(k)) defines

- "establish" = "define, document, and implement"
- "establish" = "established"
- Confusing terminology between the medical device Quality System regulation and the software industry:
  - requirements,
  - specification,
  - verification, and
  - validation.

2.3.1.1 Requirements and Specifications

The Quality System regulation states

1. that design input requirements must be documented and
2. that specified requirements must be verified

But, the regulation does not further clarify the distinction between the terms "requirement" and "specification."

- Requirement
  - can be any need or expectation for a system or for its software.
  - reflects the stated or implied needs of the customer: requirements may be
    - market-based,
    - contractual,
    - statutory, or
    - an organization's internal requirements.
  - various examples of requirements
    - design, functional, implementation, interface, performance, or physical requirements
  - Software requirements derived from the system requirements for those aspects of system functionality
  - Software requirements are typically stated in functional terms and are defined, refined, and updated as a development project progresses.
  - Success in accurately and completely documenting software requirements is a crucial factor in successful validation of the resulting software.
- Specification
  - defined as "a document that states requirements." (See 21 CFR §820.3(y).)
  - It may refer to or include drawings, patterns, or other relevant documents
  - It usually indicates the means and the criteria whereby conformity with the requirement can be checked.
  - Various examples of written specifications
    - system requirements specification,
    - software requirements specification,
    - software design specification,
    - software test specification,
    - software integration specification, etc.
  - All of these documents are design outputs for which various forms of verification are necessary.

2.3.1.2 Verifiaction and Validation

The Quality System regulation is harmonized with ISO 8402:1994, which treats "verification" and "validation" as separate and distinct terms.

- Software verification
  - It provides objective evidence that the design outputs of a particular phase of the software development life cycle meet all of the specified requirements for that phase.
  - It looks for
    * consistency,
    * completeness, and
    * correctness of the software and its supporting documentation
  - **Software testing**
    * verification activities intended to confirm that software development output meets its input requirements.
  - Types of verification activities include
    * various static and dynamic analyses,
    * code and document inspections,
    * walkthroughs, and other techniques.
- Software Validation
  - Confirmation by examination and provision of the following objective evidence:
  - Evidence 1: software specifications conform to **user needs and intended uses**, and
  - Evidnece 2: the particular requirements implemented through software can be consistently fulfilled.
  - Evidnece 3: all software requirements have been implemented **correctly and completely and are traceable** to system requirements.
  - A conclusion that software is validated is highly dependent upon **comprehensive** software testing, inspections, analyses, and other verification tasks performed **at each stage of the software development life cycle**.
  - **Testing** of device software functionality in a **simulated* use environment**, and **user site testing** are typically included as components of an overall design validation program for a software automated device.
- Difficulty in Software verification and validation
  - a developer cannot test forever, and
  - it is difficult to know how much evidence is enough.
  - In large measure, software validation is a matter of developing a **"level of confidence"** that the device meets all requirements and user expectations for the software automated functions and features of the device.
  - Considerations for an acceptable level of confidence
    * measures such as defects found in specifications documents,
    * estimates of defects remaining,
    * testing coverage, and other techniques are all used to develop before shipping the product.
    * However, a level of confidence varies depending upon the safety risk (hazard) posed by the automated functions of the device. (Info on safety risk is found

in Section 4 and in the international standards ISO/IEC 14971-1 and IEC 60601-1-4 referenced in Appendix A).

2.3.1.3 IQ/OQ/PQ

IQ/OQ/PQ are the terminology related to user site software validation

- Installation qualification (IQ)
- Operational qualification (OQ)
- Performance qualification (PQ).

Definitions of these terms may be found in FDA's Guideline on General Principles of Process Validation, dated May 11, 1987, and in FDA's Glossary of Computerized System and Software Development Terminology, dated August 1995. Both FDA personnel and device manufacturers need to be aware of these differences in terminology as they ask for and provide information regarding software validation.

2.3.2 Software Development as Part of System Design

Software validation must be considered within the context of the overall design validation for the system. A documented requirements specification represents

- the user's needs
- intended uses from which the product is developed.

A primary goal of software validation is to then demonstrate that all completed software products comply with all documented software and system requirements.

2.3.3 Software Is Different from Hardware

Software engineering needs an even greater level of managerial scrutiny and control than does hardware engineering.

2.3.4 Benefits of Software Validation

- Increase the usability and reliability of the device,
- Resulting in decreased failure rates, fewer recalls and corrective actions, less risk to patients and users, and reduced liability to device manufacturers.
- Software validation can also reduce long term costs by making it easier and less costly to reliably modify software and revalidate software changes.

2.3.5 Design Review

Design reviews are documented, comprehensive, and systematic examinations of a design to evaluate

- the adequacy of the design requirements,
- the capability of the design to meet these requirements, and
- to identify problems.

Design review is a primary tool for managing and evaluating development projects.

- It is strongly recommended that it should be formal design because it is more structured than the informal one.
- It includes participation from others outside the development team.
- It may review reference or include results from other formal and informal reviews.
- Design reviews should include
  - examination of development plans,
  - requirements specifications,
  - design specifications,
  - testing plans and procedures,
  - all other documents and activities associated with the project,
  - verification results from each stage of the defined life cycle, and
  - validation results for the overall device.
- The Quality System regulation requires that at least one formal design review be conducted during the device design process. **However, it is recommended that multiple design reviews be conducted**
  - (e.g., at the end of each software life cycle activity, in preparation for proceeding to the next activity).
- Formal design reviews documented should include:
  - the appropriate tasks and expected results, outputs, or products been established for each software life cycle activity
  - correctness, completeness, consistency, and accuracy
  - satisfaction for the standards, practices, and conventions of that activity
  - establishment of a proper basis for initiating tasks for the next software life cycle activity

2.4 Principles of Software Validation

2.4.1 Requirements

A documented software requirements specification provides a baseline for both validation and verification. **The software validation process must include an established software requirements specification** (Ref: 21 CFR 820.3(z) and (aa) and 820.30(f) and (g)).

2.4.2 Defect Prevention

In order to establish that confidence, software developers should use a mixture of methods and techniques to prevent software errors and to detect software errors that do occur.

2.4.3 Time and Effort

Preparation for software validation should begin early, i.e., **during design and development planning and design input**. The final conclusion that the software is validated should be **based on evidence** collected from planned efforts conducted throughout the software lifecycle.

2.4.4 Software Life Cycle

- Software validation takes place within the environment of an established software life cycle.
- The software life cycle contains **software engineering tasks and documentation** necessary to support the software validation effort.
- specific verification and validation tasks need to be appropriate for the intended use of the software

2.4.5 Plans

- The software validation process is defined and controlled through the use of a plan.
- The software validation plan defines "what" is to be accomplished through the software validation effort.
- Software validation plans specify areas such as
  - scope,
  - approach,
  - resources,
  - schedules and the types and extent of activities,
  - tasks, and
  - work items.

2.4.6 Procedures

The software validation process is executed through the use of procedures. These procedures establish "how" to conduct the software validation effort. The procedures should identify the specific actions or sequence of actions that must be taken to complete individual validation activities, tasks, and work items.

2.4.7 Software Validation After a Change

- Due to the complexity of software, a small local change may have a significant global system impact.
- If a change exists in the software, the whole validation status of the software needs to be re-established.
- need to determine the extent and impact of that change on the entire software system.
- the software developer should then conduct an appropriate level of software regression testing to show that unchanged but vulnerable portions of the system have not been adversely affected.

2.4.8 Validation Coverage

- Validation coverage should be based on the software's complexity and safety risk.
- The selection of validation activities, tasks, and work items should be commensurate with the complexity of the software design and the risk associated with the use of the software for the specified intended use.

2.4.9 Independence of Review

- Validation activities should be based on the basic quality assurance precept of "independence of review."
- Self-validation is extremely difficult.
- When possible, an independent evaluation is always better (like a contracted third-party independent verification and validation)
- Another approach is to assign internal staff members that are not involved in a particular design or its implementation, but who have sufficient knowledge to evaluate the project and conduct the verification and validation activities.

2.4.10 Flexibility and Responsibility

The device manufacturer has flexibility in choosing how to apply these validation principles, but retains ultimate responsibility for demonstrating that the software has been validated. FDA regulated medical device applications include software that:

- Is a component, part, or accessory of a medical device;
  - components: e.g., application software, operating systems, compilers, debuggers, configuration management tools, and many more
- Is itself a medical device; or
- Is used in manufacturing, design and development, or other parts of the quality system.
- No matter how complex and disperse the software is, the manufacturer is in charge of responsibility for software validation.

2.5 Activities and Tasks

Software validation is accomplished through **a series of activities and tasks** that are planned and executed at various stages of the software development life cycle. These tasks may be

- one time occurrences
- iterated many times

2.5.1 Software Life Cycle Activities

- Software developers should establish a software life cycle model that is appropriate for their product and organization.
- The selected software life cycle model should cover the software from its birth to its retirement.
- Activities in a typical software life cycle model:
  - Quality Planning
  - System Requirements Definition
  - Detailed Software Requirements Specification
  - Software Design Specification
  - Construction or Coding
  - Testing
  - Installation
  - Operation and Support
  - Maintenance
  - Retirement
- Verification, testing, and other tasks that support software validation occur during each of these activities.
- Several software life cycle models defined in FDA's Glossary of Computerized System and Software Development

Terminology dated August 1995:

- waterfall
- spiral
- rapid prototyping
- incremental development, etc.

2.5.2 Typical Tasks Supporting Validation

the software developer should at least consider each of the risk-related tasks and should define and document which tasks are or are not appropriate for their specific application.

2.5.2.1 Quality Planning

Design and development planning should culminate in a plan that identifies

- necessary tasks,
- procedures for anomaly reporting and resolution,
- necessary resources, and
- management review requirements including formal design reviews.

The plan should include:

- The specific tasks for each life cycle activity;
- Enumeration of important quality factors (e.g., reliability, maintainability, and usability);
- Methods and procedures for each task;
- Task acceptance criteria;
- Criteria for defining and documenting outputs in terms that will allow evaluation of their conformance to input requirements;
- Inputs for each task;
- Outputs from each task;
- Roles, resources, and responsibilities for each task;
- Risks and assumptions; and
- Documentation of user needs.

The plan should identify

- the personnel,
- the facility and equipment resources for each task, and
- the role that risk (hazard) management will play.

A configuration management plan should be developed that will guide and control multiple parallel development activities and ensure proper communications and documentation.

Controls are necessary to ensure positive and correct correspondence among all approved versions of the specifications documents, source code, object code, and test suites that comprise a software system. The controls also should ensure accurate identification of, and access to, the currently approved versions.

Procedures should be created for reporting and resolving software anomalies found through validation or other activities.

Management should identify the reports and specify the contents, format, and responsible organizational elements for each report. Procedures also are necessary for the review and approval of software development results, including the responsible organizational elements for such reviews and approvals.

Typical Tasks – Quality Planning

- Risk (Hazard) Management Plan
- Configuration Management Plan

- Software Quality Assurance Plan
  - Software Verification and Validation Plan
    - Verification and Validation Tasks, and Acceptance Criteria
    - Schedule and Resource Allocation (for software verification and validation activities)
    - Reporting Requirements
  - Formal Design Review Requirements
  - Other Technical Review Requirements
- Problem Reporting and Resolution Procedures
- Other Support Activities

2.5.2.2 Requirements

Requirements development includes the

- identification,
- analysis, and
- documentation of information about the device and its intended use.

Areas of special importance include allocation of system functions to

- hardware/software,
- operating conditions,
- user characteristics,
- potential hazards, and
- anticipated tasks.

In addition, the requirements should state clearly the intended use of the software. It is not possible to validate software without predetermined and documented software requirements. Typical software requirements specify the following:

- All software system inputs;
- All software system outputs;
- All functions that the software system will perform;
- All performance requirements that the software will meet, (e.g., data throughput, reliability, and timing);
- The definition of all external and user interfaces, as well as any internal software-to-system interfaces;
- How users will interact with the system;
- What constitutes an error and how errors should be handled;
- Required response times;
- The intended operating environment for the software, if this is a design constraint (e.g., hardware platform, operating system);
- All ranges, limits, defaults, and specific values that the software will accept; and
- All safety related requirements, specifications, features, or functions that will be implemented in software.

Software requirement specifications should identify clearly the potential hazards that can result from a software failure in the system as well as any safety requirements to be implemented in software.

The consequences of software failure should be evaluated, along with means of mitigating such failures (e.g., hardware mitigation, defensive programming, etc.).

The Quality System regulation requires a mechanism for addressing incomplete, ambiguous, or conflicting requirements. (See 21 CFR 820.30(c).) Each requirement (e.g., hardware, software, user, operator interface, and safety) identified in the software requirements specification should be evaluated for accuracy, completeness, consistency, testability, correctness, and clarity.

For example, software requirements should be evaluated to verify that:

- There are no internal inconsistencies among requirements;
- All of the performance requirements for the system have been spelled out;
- Fault tolerance, safety, and security requirements are complete and correct;
- Allocation of software functions is accurate and complete;
- Software requirements are appropriate for the system hazards; and
- All requirements are expressed in terms that are measurable or objectively verifiable.

A software requirements traceability analysis should be conducted to trace software requirements to (and from) system requirements and to risk analysis results. In addition to any other analyses and documentation used to verify software requirements, a formal design review is recommended to confirm that requirements are fully specified and appropriate before extensive software design efforts begin. Requirements can be approved and released incrementally, but care should be taken that interactions and interfaces among software (and hardware) requirements are properly reviewed, analyzed, and controlled.

Typical Tasks – Requirements

- Preliminary Risk Analysis
- Traceability Analysis
  - Software Requirements to System Requirements (and vice versa)
  - Software Requirements to Risk Analysis
- Description of User Characteristics
- Listing of Characteristics and Limitations of Primary and Secondary Memory
- Software Requirements Evaluation
- Software User Interface Requirements Analysis
- System Test Plan Generation
- Acceptance Test Plan Generation
- Ambiguity Review or Analysis

2.5.2.3 Design

In the design process, the software requirements specification is translated into a logical and physical representation of the software to be implemented. The software design specification is a description of what the software should do and how it should do it. The design specification may contain both a high level summary of the design and detailed design information. Human factors engineering should be woven into

- the entire design and development process,
- the device design requirements,
- analyses, and
- tests.

Device safety and usability issues should be considered when developing

- flowcharts,
- state diagrams,
- prototyping tools, and
- test plans.

Also, task and function analyses, risk analyses, prototype tests and reviews, and full usability tests should be performed. Participants from the user population should be included when applying these methodologies.

The software design specification should include:

- Software requirements specification, including predetermined criteria for acceptance of the software;
- Software risk analysis;
- Development procedures and coding guidelines (or other programming procedures);
- Systems documentation (e.g., a narrative or a context diagram) that describes the systems context in which the program is intended to function, including the relationship of hardware, software, and the physical environment;
- Hardware to be used;
- Parameters to be measured or recorded;
- Logical structure (including control logic) and logical processing steps (e.g., algorithms);
- Data structures and data flow diagrams;
- Definitions of variables (control and data) and description of where they are used;
- Error, alarm, and warning messages;
- Supporting software (e.g., operating systems, drivers, other application software);
- Communication links (links among internal modules of the software, links with the supporting software, links with the hardware, and links with the user);
- Security measures (both physical and logical security); and
- Any additional constraints not identified in the above elements.

The first four of the elements noted above usually are separate pre-existing documents that are included by reference in the software design specification. Software requirements specification was discussed in the preceding section, as was software risk analysis.

Software design evaluations criteria:

- complete,
- correct,
- consistent,
- unambiguous,
- feasible,
- maintainable,
- analyses of control flow,
- data flow,
- complexity,
- timing,
- sizing,
- memory allocation,
- criticality analysis, and many other aspects of the design

Appropriate consideration of software architecture (e.g., modular structure) during design can reduce the magnitude of future validation efforts when software changes are needed.

A traceability analysis should be conducted to verify that the software design implements all of the software requirements. As a technique for identifying where requirements are not sufficient, the traceability analysis should also verify that all aspects of the design are traceable to software requirements.

An analysis of communication links should be conducted to evaluate the proposed design with respect to hardware, user, and related software requirements. At the end of the software design activity, a Formal Design Review should be conducted to verify that the design is correct, consistent, complete, accurate, and testable, before moving to implement the design.

Several versions of both the software requirement specification and the software design specification should be maintained. All approved versions should be archived and controlled in accordance with established configuration management procedures.

Typical Tasks – Design

- Updated Software Risk Analysis
- Traceability Analysis - Design Specification to Software Requirements (and vice versa)
- Software Design Evaluation
- Design Communication Link Analysis
- Module Test Plan Generation
- Integration Test Plan Generation
- Test Design Generation (module, integration, system, and acceptance)

2.5.2.4 Construction or Coding

Software may be constructed either by coding. Coding is the software activity where the detailed design specification is implemented as source code. It is the last stage in decomposition of the software requirements where module specifications are translated into a programming language.

Coding usually involves the use of a high-level programming language, but may also entail the use of assembly language (or microcode) for time-critical operations.

A source code traceability analysis is an important tool to verify that all code is linked to established specifications and established test procedures. A source code traceability analysis should be conducted and documented to verify that:

- Each element of the software design specification has been implemented in code;
- Modules and functions implemented in code can be traced back to an element in the software design specification and to the risk analysis;
- Tests for modules and functions can be traced back to an element in the software design specification and to the risk analysis; and
- Tests for modules and functions can be traced to source code for the same modules and functions.

Typical Tasks – Construction or Coding

- Traceability Analyses
  - Source Code to Design Specification (and vice versa)
  - Test Cases to Source Code and to Design Specification
- Source Code and Source Code Documentation Evaluation
- Source Code Interface Analysis
- Test Procedure and Test Case Generation (module, integration, system, and acceptance)

2.5.2.5 Testing by the Software Developer

Software testing entails running software products under known conditions with defined inputs and documented outcomes that can be compared to their predefined expectations. It is a time consuming, difficult, and imperfect activity.

As such, it requires early planning in order to be effective and efficient. Test plans and test cases should be created as early in the software development process as feasible.

They should identify

- the schedules,
- environments,
- resources (personnel, tools, etc.),
- methodologies,
- cases (inputs, procedures, outputs, expected results),

- documentation, and
- reporting criteria.

Descriptions of categories of software and software testing effort appear in the literature

- NIST Special Publication 500-235, Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric;
- NUREG/CR-6293, Verification and Validation Guidelines for High Integrity Systems; and
- IEEE Computer Society Press, Handbook of Software Reliability Engineering.

Testing of all program functionality does not mean all of the program has been tested. Testing of all of a program's code does not mean all necessary functionality is present in the program. Testing of all program functionality and all program code does not mean the program is 100% correct! Software testing that finds no errors should not be interpreted to mean that errors do not exist in the software product; it may mean the testing was superficial.

An essential element of a software test case is the expected result. It is the key detail that permits objective evaluation of the actual test result. This necessary testing information is obtained from the corresponding, predefined definition or specification.

A software testing process should be based on principles that foster effective examinations of a software product. Applicable software testing tenets include:

- The expected test outcome is predefined;
- A good test case has a high probability of exposing an error;
- A successful test is one that finds an error;
- There is independence from coding;
- Both application (user) and software (programming) expertise are employed;
- Testers use different tools from coders;
- Examining only the usual case is insufficient;
- Test documentation permits its reuse and an independent confirmation of the pass/fail status of a test outcome during subsequent review.

Code-based testing is also known as structural testing or "white-box" testing. It identifies test cases based on knowledge obtained from the source code, detailed design specification, and other development documents. Structural testing can identify "dead" code that is never executed when the program is run. Structural testing is accomplished primarily with unit (module) level testing, but can be extended to other levels of software testing.

The level of structural testing can be evaluated using **metrics that are designed to show what percentage of the software structure has been evaluated during structural testing**. These metrics are typically referred to as **"coverage"** and are a measure of completeness with respect to test selection criteria. The amount of structural coverage should be commensurate with the level of risk posed by the software. Use of the term "coverage" usually means 100% coverage. Common structural coverage metrics include:

- Statement Coverage – This criteria requires sufficient test cases for each program statement to be executed at least once; however, its achievement is insufficient to provide confidence in a software product's behavior.
- Decision (Branch) Coverage – This criteria requires sufficient test cases for each program decision or branch to be executed so that each possible outcome occurs at least once. It is considered to be a minimum level of coverage for most software products, but decision coverage alone is insufficient for high-integrity applications.
- Condition Coverage – This criteria requires sufficient test cases for each condition in a program decision to take on all possible outcomes at least once. It differs from branch coverage only when multiple conditions must be evaluated to reach a decision.
- Multi-Condition Coverage – This criteria requires sufficient test cases to exercise all possible combinations of conditions in a program decision.
- Loop Coverage – This criteria requires sufficient test cases for all program loops to be executed for zero, one, two, and many iterations covering initialization, typical running and termination (boundary) conditions.
- Path Coverage – This criteria requires sufficient test cases for each feasible path, basis path, etc., from start to exit of a defined program segment, to be executed at least once. Because of the very large number of possible paths through a software program, path coverage is generally not achievable. The amount of path coverage is normally established based on the risk or criticality of the software under test.
- Data Flow Coverage – This criteria requires sufficient test cases for each feasible data flow to be executed at least once. A number of data flow testing strategies are available.

The following types of functional software testing involve generally increasing levels of effort:

- Normal Case – Testing with usual inputs is necessary. However, testing a software product only with expected, valid inputs does not thoroughly test that software product. By itself, normal case testing cannot provide sufficient confidence in the dependability of the software product.
- Output Forcing – Choosing test inputs to ensure that selected (or all) software outputs are generated by testing.
- Robustness – Software testing should demonstrate that a software product behaves correctly when given unexpected, invalid inputs. Methods for identifying a sufficient set of such test cases include Equivalence Class Partitioning, Boundary Value Analysis, and Special Case Identification (Error Guessing). While important and necessary, these techniques do not ensure that all of the most appropriate challenges to a software product have been identified for testing.
- Combinations of Inputs – The functional testing methods identified above all emphasize individual or single test inputs. Most software products operate with multiple inputs under their conditions of use. Thorough software product testing should consider the combinations of inputs a software unit or system may encounter during operation. Error guessing can be extended to identify combinations of inputs, but it is an ad hoc technique. Cause-effect graphing is one functional software testing technique that systematically identifies combinations of inputs to a software product for inclusion in test cases.

Functional and structural software test case identification techniques provide specific inputs for testing, rather than random test inputs. One weakness of these techniques is the difficulty in linking structural and functional test completion criteria to a software product's reliability.

Advanced software testing methods, such as statistical testing, can be employed to provide further assurance that a software product is dependable. Statistical testing uses randomly generated test data from defined distributions based on an operational profile (e.g., expected use, hazardous use, or malicious use of the software product). Large amounts of test data are generated and can be targeted to cover particular areas or concerns, providing an increased possibility of identifying individual and multiple rare operating conditions that were not anticipated by either the software product's designers or its testers. Statistical testing also provides high structural coverage. It does require a stable software product. Thus, structural and functional testing are prerequisites for statistical testing of a software product.

Another aspect of software testing is the testing of software changes. Changes occur frequently during software development. These changes are the result of

1. debugging that finds an error and it is corrected,
2. new or changed requirements ("requirements creep"), and
3. modified designs as more effective or efficient implementations are found.

Once a software product has been baselined (approved), any change to that product should have its own "mini life cycle," including testing. Testing of a changed software product requires additional effort. It should demonstrate

- that the change was implemented correctly, and
- that the change did not adversely impact other parts of the software product.

Regression analysis is the determination of the impact of a change based on review of the relevant documentation in order to identify the necessary regression tests to be run. **Regression testing** is the rerunning of test cases that a program has previously executed correctly and **comparing the current result to the previous result in order to detect unintended effects of a software change**. Regression analysis and regression testing should also be employed when using integration methods to build a software product to ensure that newly integrated modules do not adversely impact the operation of previously integrated modules.

In order to provide a thorough and rigorous examination of a software product, development testing is typically organized into levels: unit, integration, and system levels of testing.

(1) **Unit (module or component) level testing** focuses on the early examination of sub-program functionality and ensures that functionality not visible at the system level is examined by testing. Unit testing ensures that quality software units are furnished for integration into the finished software product.
(2) **Integration level testing** focuses on *the transfer of data* and *control across a program's internal and external interfaces.* External interfaces are those with

i) other software (including operating system software),
ii) system hardware, and
iii) the users and can be described as communications links.

(3) **System level testing** demonstrates that all specified functionality exists and that the software product is trustworthy. This testing verifies the as-built program's functionality and performance with respect to the requirements for the software product as exhibited on the specified operating platform(s). System level software testing addresses functional concerns and the following elements of a device's software that are related to the intended use(s):

- Performance issues (e.g., response times, reliability measurements);
- Responses to stress conditions, e.g., behavior under maximum load, continuous use;
- Operation of internal and external security features;
- Effectiveness of recovery procedures, including disaster recovery;
- Usability; (Usability vs Utility??)
- Compatibility with other software products;
- Behavior in each of the defined hardware configurations; and
- Accuracy of documentation.

Control measures (e.g., a traceability analysis) should be used to ensure that the intended coverage is achieved.

System level testing also exhibits the software product's behavior in the intended operating environment. The location of such testing is dependent upon the software developer's ability to produce the target operating environment(s). Depending upon the circumstances, simulation and/or testing at (potential) customer locations may be utilized.

Test plans should identify the controls needed to ensure

- that the intended coverage is achieved and
- that proper documentation is prepared when planned system level testing is conducted at sites not directly controlled by the software developer.

Test procedures, test data, and test results

- should be documented in a manner permitting objective pass/fail decisions to be reached.
- should also be suitable for review and objective decision making subsequent to running the test,
- should be suitable for use in any subsequent regression testing.

Errors detected during testing should be

- logged,
- classified,
- reviewed, and
- resolved prior to release of the software.

Software error data that is collected and analyzed during a development life cycle may be used to determine the suitability of the software product for release for commercial distribution. Test reports should comply with the requirements of the corresponding test plans.

Software testing tools are frequently used to ensure consistency, thoroughness, and efficiency in the testing of such software products and to fulfill the requirements of the planned testing activities.

Appropriate documentation providing evidence of the validation of these software tools for their intended use should be maintained (see section 6 of this guidance).

Typical Tasks – Testing by the Software Developer

- Test Planning
- Structural Test Case Identification
- Functional Test Case Identification
- Traceability Analysis - Testing
- Unit (Module) Tests to Detailed Design
- Integration Tests to High Level Design
- System Tests to Software Requirements
- Unit (Module) Test Execution
- Integration Test Execution
- Functional Test Execution
- System Test Execution
- Acceptance Test Execution
- Test Results Evaluation
- Error Evaluation/Resolution
- Final Test Report

2.5.2.6 User Site Testing

Testing at the user site is *an essential part of software validation.* The Quality System regulation requires

- installation and
- inspection procedures (including testing where appropriate) as well as
- documentation of inspection and
- testing to demonstrate proper installation. (See 21 CFR §820.170.)

Likewise, manufacturing equipment must meet specified requirements, and automated systems must be validated for their intended use. (See 21 CFR §820.70(g) and 21 CFR §820.70(i) respectively.)

Terminology regarding user site testing can be confusing. Terms such as

- beta test,
- site validation,
- user acceptance test,

- installation verification, and
- installation testing have all been used to describe user site testing.

For the purposes of this guidance, the term "user site testing" encompasses all of these and any other testing that takes place *outside of the developer's controlled environment.*

This testing should take place at a user's site with the actual hardware and software that will be part of the installed system configuration. The testing is accomplished through either actual or simulated use of the software being tested within the context in which it is intended to function.

Test planners should check with the FDA Center(s) with the corresponding product jurisdiction to determine whether there are any additional regulatory requirements for user site testing.

User site testing should follow a pre-defined written plan with

- a formal summary of testing and
- a record of formal acceptance.

The following documented evidence should be retained:

- all testing procedures,
- test input data, and
- test results

There should be evidence that hardware and software are installed and configured as specified. Measures should ensure that all system components are exercised during the testing and that the versions of these components are those specified. The testing plan should specify testing throughout the full range of operating conditions and should specify continuation for a sufficient time to allow the system to encounter a wide spectrum of conditions and events in an effort to detect any latent faults that are not apparent during more normal activities.

Some of the evaluations of the system's ability that have been performed earlier by the software developer at the developer's site should be repeated at the site of actual use. These may include tests for:

- a high volume of data,
- heavy loads or stresses,
- security,
- fault testing (avoidance, detection, tolerance, and recovery),
- error messages, and
- implementation of safety requirements.

There should be an evaluation of the ability of the users of the system to understand and correctly interface with it.
Operators should be able to perform the intended functions and respond in an appropriate and timely manner to all alarms, warnings, and error messages.
Records should be maintained of both proper system performance and any system failures

that are encountered.
The revision of the system to compensate for faults detected during this user site testing should follow the same procedures and controls as for any other software change.

The developers of the software may or may not be involved in the user site testing.

- If the developers are involved, they may seamlessly carry over to the user's site the last portions of design-level systems testing.
- If the developers are not involved, it is all the more important that the user have persons who understand the importance of careful test planning, the definition of expected test results, and the recording of all test outputs.

Typical Tasks – User Site Testing

- Acceptance Test Execution
- Test Results Evaluation
- Error Evaluation/Resolution
- Final Test Report

2.5.2.7 Maintenance and Software Changes

2.5.2.7.1 Hardware vs Software

Hardware maintenance typically includes

- preventive hardware maintenance actions,
- component replacement, and
- corrective changes.

Software maintenance includes

- corrective,
- perfective, and
- adaptive maintenance
- but does not include preventive maintenance actions or software component replacement.

2.5.2.7.2 Maintenance Types

- Corrective maintenance: Changes made to correct errors and faults in the software.
- Perfective maintenance: Changes made to the software to improve the performance, maintainability, or other attributes of the software system .
- Adaptive maintenance: Changes to make the software system usable in a changed environment.

Sufficient regression analysis and testing should be conducted to demonstrate that portions of the software not involved in the change were not adversely impacted. When changes are made to a software system,

- either during initial development or
- during post release maintenance,

This is in addition to testing that evaluates the correctness of the implemented change(s). The specific validation effort necessary for each software change is determined by

- the type of change,
- the development products affected, and the
- impact of those products on the operation of the software.

2.5.2.7.3 Factors of Limitting Validation Effort Needed When a Change Is Made

- careful and complete documentation of the design structure and
- careful and complete documentation of interrelationships of various modules,
- interfaces, etc.
- For example,
  - test documentation,
  - test cases, and
  - results of previous verification and validation testing All of them need to be archived if they are to be available for performing subsequent regression testing.

The following additional maintenance tasks should be addressed:

- Software Validation Plan Revision - For software that was previously validated, the existing software validation plan should be revised to support the validation of the revised software. If no previous software validation plan exists, such a plan should be established to support the validation of the revised software.
- Anomaly Evaluation – Software organizations frequently maintain documentation, such as software problem reports that describe software anomalies discovered and the specific corrective action taken to fix each anomaly.
  - Too often, however, mistakes are repeated because software developers do not take the next step to determine the root causes of problems and make the process and procedural changes needed to avoid recurrence of the problem.
  - Software anomalies should be evaluated in terms of their severity and their effects on system operation and safety,
  - but they should also be treated as symptoms of process deficiencies in the quality system.
  - A root cause analysis of anomalies can identify specific quality system deficiencies.
  - Where trends are identified (e.g., recurrence of similar software anomalies), appropriate corrective and preventive actions must be implemented and documented to avoid further recurrence of similar quality problems. (See 21 CFR 820.100.)

- Problem Identification and Resolution Tracking - All problems discovered during maintenance of the software should be documented. The resolution of each problem should be tracked to ensure it is fixed, for historical reference, and for trending.
- Proposed Change Assessment - All proposed modifications, enhancements, or additions should be assessed to determine the effect each change would have on the system. This information should determine the extent to which verification and/or validation tasks need to be iterated.
- Task Iteration - For approved software changes, all necessary verification and validation tasks should be performed to ensure that planned changes are implemented correctly, all documentation is complete and up to date, and no unacceptable changes have occurred in software performance.
- Documentation Updating – Documentation should be carefully reviewed to determine which documents have been impacted by a change. All approved documents (e.g., specifications, test procedures, user manuals, etc.) that have been affected should be updated in accordance with configuration management procedures. Specifications should be updated before any maintenance and software changes are made.

2.6 Validation of Automated Process Equipment and Quality System Software

The Quality System regulation requires that "when computers or automated data processing systems are used as part of production or the quality system, the [device] manufacturer shall validate computer software for its intended use according to an established protocol." (See 21 CFR §820.70(i)). This has been a regulatory requirement of FDA's medical device Good Manufacturing Practice (GMP) regulations since 1978.

Computer systems that implement part of a device manufacturer's production processes or quality system (or that are used to create and maintain records required by any other FDA regulation) are subject to the Electronic Records; Electronic Signatures regulation. (See 21 CFR Part 11.) This regulation establishes additional security, data integrity, and validation requirements when records are created or maintained electronically. These additional Part 11 requirements should be carefully considered and included in system requirements and software requirements for any automated record `keeping systems`. System validation and software validation should demonstrate that all Part 11 requirements have been met.

Computers and automated equipment are used extensively throughout all aspects of

- medical device design,
- laboratory testing and analysis,
- product inspection and acceptance,
- production and process control,
- environmental controls,
- packaging,
- labeling,
- traceability,
- document control,
- complaint management, and many other aspects of the quality system.

Increasingly, automated plant floor operations can involve extensive use of embedded systems in:

- programmable logic controllers;
- digital function controllers;
- statistical process control;
- supervisory control and data acquisition;
- robotics;
- human-machine interfaces;
- input/output devices; and
- computer operating systems.

All software tools used for software design are subject to the requirement for software validation, but the validation approach used for each application can vary widely.

Validation is typically supported by:

- verifications of the outputs from each stage of that software development life cycle; and
- checking for proper operation of the finished software in the device manufacturer's intended use environment.

2.6.1 How Much Validation Evidence Is Needed?

The level of validation effort should be commensurate with

- the risk posed by the automated operation,
- the complexity of the process software,
- the degree to which the device manufacturer is dependent upon that automated process to produce a safe and effective device

Documented requirements and risk analysis of the automated process help to define the scope of the evidence needed to show that the software is validated for its intended use. Without a plan, extensive testing may be needed for:

- a plant-wide electronic record and electronic signature system;
- an automated controller for a sterilization cycle; or
- automated test equipment used for inspection and acceptance of finished circuit boards in a lifesustaining / life-supporting device.

High risk applications should not be running in the same operating environment with non-validated software functions, even if those software functions are not used. Risk mitigation techniques such as memory partitioning or other approaches to resource protection may need to be considered when high risk applications and lower risk applications are to be used in the same operating environment.

When software is upgraded or any changes are made to the software, the device manufacturer should consider how those changes may impact the "used portions" of the software and

must reconfirm the validation of those portions of the software that are used. (See 21 CFR §820.70(i).)

2.6.2 Defined User Equipment

A very important key to software validation is a documented user requirements specification that defines:

- the "intended use" of the software or automated equipment; and
- the extent to which the device manufacturer is dependent upon that software or equipment for production of a quality medical device.

The device manufacturer (user) needs to define the expected operating environment including any required hardware and software configurations, software versions, utilities, etc. The user also needs to:

- document requirements for system performance, quality, error handling, startup, shutdown, security, etc.;
- identify any safety related functions or features, such as sensors, alarms, interlocks, logical processing steps, or command sequences; and
- define objective criteria for determining acceptable performance.

The validation must be conducted in accordance with a documented protocol, and the validation results must also be documented. (See 21 CFR §820.70(i).) Test cases should be documented that will exercise the system to challenge its performance against the predetermined criteria, especially for its most critical parameters.

Test cases should address

- error and alarm conditions,
- startup, shutdown,
- all applicable user functions and operator controls,
- potential operator errors,
- maximum and minimum ranges of allowed values, and
- stress conditions applicable to the intended use of the equipment.

The test cases should be executed and the results should be recorded and evaluated to determine whether the results support a conclusion that the software is validated for its intended use.

A device manufacturer may conduct a validation using their own personnel or may depend on a third party such as the equipment/software vendor or a consultant. In any case, the device manufacturer retains the ultimate responsibility for ensuring that the production and quality system software:

- is validated according to a written procedure for the particular intended use; and
- will perform as intended in the chosen application.

The device manufacturer should have documentation including:

- defined user requirements;
- validation protocol used;
- acceptance criteria;
- test cases and results; and
- a validation summary that objectively confirms that the software is validated for its intended use.

2.6.3 Validation of Off-The-Shelf Software and Automated Equipment

Most of the automated equipment and systems used by device manufacturers are supplied by thirdparty vendors and are purchased off-the-shelf (OTS). *The device manufacturer is responsible for ensuring that the product development methodologies used by the OTS software developer are appropriate and sufficient for the device manufacturer's intended use of that OTS software.*

Where possible and depending upon the device risk involved, the device manufacturer should consider auditing the vendor's design and development methodologies used in the construction of the OTS software and should assess the development and validation documentation generated for the OTS software. Such audits can be conducted by the device manufacturer or by a qualified third party.

The audit should demonstrate that the vendor's procedures for and results of the verification and validation activities performed the OTS software are appropriate and sufficient for the safety and effectiveness requirements of the medical device to be produced using that software.