PARASOFT®

TECHNICAL WHITEPAPER

Medical Device Software Development

Following FDA Guidelines for Software Validation

OVERVIEW

As a means of providing guidance to medical device software makers, the FDA issued the General Principles of Software Validation. This document outlines principles for validating medical device software, as well as the software used to design, develop, or manufacture medical devices, that the FDA considers acceptable. Organizations that produce software within these categories are therefore subject to FDA regulation and must comply with the General Principles of Software Validation.

Devices categorized as class II and III, as well as some class I devices, are subject to design controls.

Of these, the following types of software must be validated for FDA approval:

- Software used as a component, part, or accessory of a medical device.
- Software that is itself a medical device (like blood establishment software).
- Software used in the production of a device (such as programmable logic controllers in manufacturing equipment).
- Software used in implementation of the device manufacturer's quality system (for example, software that records and maintains the device history record).

In this paper, we identify software development challenges that medical device makers face when attempting to integrate the principles outlined by the FDA, and describe how Parasoft's automated defect prevention solutions help organizations overcome the challenges of an integrated SDLC approach. For clear compliance efforts moving forward, we provide a point-to-point index of FDA principles and the Parasoft capabilities that support them.

As an effective way to gain approval, the FDA recommends that medical device software development teams take a software development lifecycle (SDLC) approach that integrates risk management strategies with principles for software validation. An integrated SDLC merges validation and verification activities, including defect prevention practices such as unit testing, peer code reviews, static analysis, manual testing, and regression testing, throughout the SDLC. The result of such an approach is an emphasis on planning, verification, testing, traceability, and configuration management.

Developing software for medical devices that complies with the FDA's Quality System regulation is a challenging endeavor that's as much a business issue as it is an engineering feat

BURDENS OF THE LEAST BURDENSOME APPROACH

The FDA guidance does not prescribe specific practices, tools, coding methods or any other technical activity. The FDA instead prescribes the concept of the Least Burdensome Approach. In this approach, organizations determine, and strictly adhere to, their self-defined validation and verification processes.

Development activities and outcomes must be clearly defined, documented, verified, and validated against the organization's process. The goal of this approach is to give medical device makers the flexibility to determine how to best ensure public safety.

But for some organizations, the burden associated with regulatory compliance has actually been compounded. Not only must the organization validate and verify the software with extensive testing, they must also develop the basis on which the software would be considered safe for use. Furthermore, organizations must prove that they've followed their self-defined processes.

In terms of the extensive planning and testing requirements, the following examples are some of the challenges software engineers must overcome (FDA 21 CFR):

- The software validation process cannot be completed without an established software requirements specification, which specifies the intended use. Results must not only verify that the specifications are met, but they must be reproduced consistently. Testing methods like regression testing can be implemented to meet the requirement.
- Validation must be established and re-established for even small changes. This means that validation activities, including static analysis, unit testing, code review, and the like, must be repeated if the code has changed. Furthermore, as software continues to become more and more complex, tests that validate the changes should be conducted in scale with the application to ensure that no other part of the system is affected.
- Changes to the requirements deemed significantly different enough from the originally registered design may require the product to be re-registered per FDA Section 501(K).
- There are no "FDA certified" tools or methods. No person, organization, or tool can claim any form of supposed FDA certification, but any software used to automate any part of the device process or any part of the quality system must be validated. You must be able to run any tools used to assist in the verification and validation efforts on a control code base and confirm that the results are consistent, which may affect your time to market.

The FDA has established grounds for approval in a way that effectively puts the responsibility of ensuring quality and public safety back to the device makers. But device makers are often dealing with a bigger challenge: bridging the gap between business goals and the development process.

LACK OF SOFTWARE DEVELOPMENT POLICY

Software engineers often either don't know what's expected or do not understand the business objective behind the guidelines driving their products. They are expected to write code that meets the requirements, without necessarily understanding why requirements have been established in the first place. The best way to overcome these challenges, while satisfying the FDA's requirements for medical device software development, is to drive the development process in a platform based on policy-driven development.

Policy-driven development involves:

- Clearly defining expectations and documenting them in understandable policies.
- Training engineers on the business objectives driving those policies.
- Monitoring policy adherence in an automated, unobtrusive way.

Integrating these principles into the development process gives businesses the ability to accurately and objectively measure productivity and application quality. In addition to reducing risk, the result is lower cost over the total software development lifecycle from build to support.

Adopting a policy-driven development process is crucial for achieving the following goals:

- Ensuring that engineers don't make trade-offs that compromise reliability or performance.
- Ensuring that engineers build security into the application, safeguarding it from potential attacks.
- Preventing defects that could result in costly recalls, litigation, or a damaged market position.
- Accurately and consistently applying quality processes.
- Gaining the traceability and auditability required to ensure continued policy compliance.

Software engineers are continually making business decisions. Every line of code, test conducted (or left undone), and guideline or standard followed (or ignored) have profound effects on the business. With public safety, potential litigation, market position, and other consequences on the line, it behooves software development teams and people in traditional business management positions to come together on policy, and implement the strategy into their software development lifecycle.

PARASOFT'S SOLUTIONS FOR MEDICAL DEVICE SOFTWARE DEVELOPMENT

Parasoft's solutions for medical device software development feature the following technologies:

- Code analysis configurations that enforce coding best practices according to CWE, OWASP, MISRA, and other sources
- Integrated defect prevention, validation, and verification
- A continuous, policy-driven compliance process with real-time visibility
- Correlation of all key artifacts, including tests, code and test coverage, requirements, source code, analysis violations, metrics analysis, project tasks, and so on for comprehensive requirements traceability

PARASOFT'S SUPPORT FOR FDA PRINCIPLES OF SOFTWARE VALIDATION

Parasoft supports the FDA's vision of an integrated SDLC for C, C++, Java, and .NET by continuously and consistently applying software testing practices specified in the General Principles of Software Validation. Parasoft's software-development management platform designed for medical device software development is pre-configured with processes and best practices described in the FDA guidelines, enabling organizations to implement an automated defect-prevention strategy.

Leveraging policy-driven development, Parasoft provides an environment that drives productivity and software quality.

Figure 1:
FDA Medical Device Software Compliance Productivity Dashboard

Sections 1, 2, and 3 of the FDA's guidelines set the purpose, scope, and context for software validation. Since these sections focus on identifying terms rather than outlining expectations, we will use Section 4 (Principles of Software Validation) and Section 5 (Activities and Tasks) to highlight how Parasoft delivers end-to-end solutions for the medical device software industry.

Software testing is one of many verification activities intended to confirm that software development output meets its input requirements. However, quality software cannot be delivered by testing alone. Quality software is delivered consistently via a solid, repeatable process, which requires an integrated system that assists with defining requirements, ensuring good coding practices, and testing effectively. This process needs to be visible, measurable, and — most importantly — repeatable. The following table matches FDA principles with Parasoft features and functionalities that help organizations achieve compliance.

4.1 REQUIREMENTS

A documented software requirements specification provides a baseline for both validation and verification.

The software validation process cannot be completed without an established software requirements specification.

Parasoft Support

- A system for mapping requirements to development tasks and monitoring the implementation and validation of each requirement.
- An open API and out-of-the-box configurations for the most popular resource management and bug management systems and tools like Excel, Word and MS Project,
- Requirements testing highlights which requirements need to be tested.
- Requirements traceability correlates requirements to iterations, tasks, code, tests, builds, and artifacts.
- Graphical reporting of requirement status as indicated by developers.

4.2 DEFECT PREVENTION

Software quality assurance needs to focus on preventing the introduction of defects into the software development process rather than trying to “test quality into” the software code after it is written.

Software testing is limited in its ability to surface all latent defects in code.

Software testing by itself is not sufficient to establish confidence that the software is fit for its intended use.

Parasoft Support

- The industry’s most comprehensive automated defect prevention system.
- A proven automated defect prevention system that can be implemented into any software development environment
- Technologies that automate defect prevention practices to ensure their consistent and comprehensive application.
- An automated infrastructure that drives the defect prevention process to ensure that it remains on track and does not disrupt the team’s workflow.
- A system that monitors adherence to defect prevention policies.
- Capabilities include:
  - Quality Policy Management
  - Static Code Analysis
    - Pattern-Based
    - Flow-Based
    - Metrics-Based
  - Automated Peer Code Review
  - Contextual Peer Code Review
  - Unit Testing Framework
  - Code Coverage Analysis

4.3 TIME AND EFFORT

Preparation of software validation should begin early. For example, during design and development planning and design input.

Parasoft Support

- Preconfigured FDA templates.
- A central system that documents and defines requirements, expected tasks, timelines and outcomes — as well as manages by exception to ensure that the project is meeting expectations.
- A continuous, end-to-end quality process that ensures defect prevention and detection tasks are not only deployed across every stage of the SDLC, but also ingrained into the team's workflow.
- A system that answers in real-time:
  - Will I be on time?
  - Will I be on budget?
  - Will I have the expected functionality?
  - Will it work?

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. In addition, the software life cycle contains specific verification and validation tasks that are appropriate for the intended use of the software.

Parasoft Support

- Software development management platform integrates SDLC into the broader development infrastructure; flexible process/workflow definition tool allows for a visible and repeatable SDLC.
- Process-based implementation drives manual and automated validation tasks across the SDLC, ensuring consistency and traceability.
- Services that integrate and automate the SDLC to ensure that quality software can be produced consistently and efficiently.
- Services that improve development productivity and form the foundation for a repeatable, sustainable quality process.

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 are a significant quality system tool. Software validation plans specify areas such as scope, approach, resources, schedules and the types and extent of activities, tasks, and work items.

Parasoft Support

- Plans are expressed as customizable templates that define common software development and validation plans.
- A system for mapping quality plan requirements to development tasks and monitoring the implementation and validation of each requirement.
- Services that ensure the validation plan is clearly defined and enforceable.
- Centralized definition and management of organization-level and team-level policies for implementing the validation plan.

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.

Parasoft Support

- Policy defines procedures and the Parasoft software development management system automatically orchestrates the all tasks in the appropriate sequence with complete traceability. In this way, checklist items are converted into an executable process.
- Automated application of quality policies across the SDLC.
- Monitorable quality gates and thresholds throughout the SDLC.
- Workflow optimization to ensure that tasks to support quality policies can become a sustainable part of the team's existing workflow.
- Preconfigured FDA templates.

4.7 SOFTWARE VALIDATION AFTER A CHANGE

Due to the complexity of software, a seemingly small local change may have a significant global system impact.

Whenever software is changed, a validation analysis should be conducted not just for validation of the individual change, but also to determine the extent and impact of that change on the entire software system.

Parasoft Support

- Continuous regression testing, which applies a broad range of validation methods to immediately alert the team when modifications impact application behavior.
- Change-based testing, which helps teams identify and execute only the test cases directly related to the most recent source code modifications.
- Requirements traceability correlates requirements to iterations, tasks, code, tests, builds, and artifacts.

4.8 VALIDATION COVERAGE

Validation coverage should be based on the software's complexity and safety risk - not on firm size or resource constraints. 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.

Validation documentation should be sufficient to demonstrate that all software validation plans and procedures have been completed successfully.

Parasoft Support

- Automated assessment of high-risk code using industry-standard metrics.
- Identification of specific pieces of code that exceed industrystandard or customized complexity metrics thresholds.
- Coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge test suite efficacy and completeness.
- Archived reports and trend graphs document validation efforts and quality improvements.

4.9 INDEPENDENCE OF REVIEW

Self-validation is extremely difficult. When possible, an independent evaluation is always better, especially for higher risk applications.

Parasoft Support

- Policy defines procedures and the Parasoft software development management system automatically orchestrates the all tasks in the appropriate sequence with complete traceability. In this way, checklist items are converted into an executable process.
- Automated application of quality policies across the SDLC.
- Monitorable quality gates and thresholds throughout the SDLC.
- Workflow optimization to ensure that tasks to support quality policies can become a sustainable part of the team's existing workflow.
- Preconfigured FDA templates.

4.10 FLEXIBILITY AND RESPONSIBILITY

Software is designed, developed, validated, and regulated in a wide spectrum of environments, and for a wide variety of devices with varying levels of risk.

Software validation activities and tasks may be dispersed, occurring at different locations and being conducted by different organizations.

However, regardless of the distribution of tasks, contractual relations, source of components, or the development environment, the device manufacturer or specification developer retains ultimate responsibility for ensuring that the software is validated.

Parasoft Support

- A policy-driven, flexible, repeatable, and traceable validation process that can span distributed environments and include both automated and manual tasks.
- The ability to define a test suite that starts verifying software on the "host" development environment then reuse that same test suite to validate software functionality in other environments — on simulators, target devices, and other platforms.
- The visibility and consistency needed to reduce the risks of outsourcing and geographically-distributed development.
- An automated framework that manages software verification methods to ensure that all software development activities meet expectations.
- Support for defect resolution, not just defect prevention and detection. Each issue detected is prioritized, automatically correlated to the developer who introduced it, then distributed to his or her IDE with direct links to the problematic code. Eventually, developers start writing compliant code as a matter of habit. Moreover, through integration with the development infrastructure, results are correlated with requirements, bugs, and source code changes — converting data into actionable information.

5.1 SOFTWARE LIFE CYCLE ACTIVITIES

Activities in a typical software life cycle model include the following:

- 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. A life cycle model organizes these software development activities in various ways and provides a framework for monitoring and controlling the software development project.

Parasoft Support

- A policy-based approach that defines the organization's expectations for quality across each of these SDLC phases, ingrains practices for measuring policy compliance into the team's workflow across the SDLC, and automatically monitors policy compliance for visibility and traceability.
- A centralized and enforceable policy that not only establishes the organization's expectations, but also keeps the team on track towards achieving those expectations — providing a framework for producing predictable outcomes.
- The ability to define a truly comprehensive policy that not only enforces coding requirements through static analysis, but also addresses dynamic testing requirements regarding unit, integration, and system-level testing, coverage analysis, and regression testing.
- Preconfigured FDA templates.

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.

A software life cycle model and associated activities should be identified, as well as those tasks necessary for each software life cycle activity.

Parasoft Support

- Plans are expressed as an interoperable business process. Preconfigured, customizable templates define common software quality plans.
- A system for mapping quality plan requirements to development tasks and monitoring the implementation and validation of each requirement.
- Services that ensure the validation plan is clearly defined and enforceable.
- Centralized definition and management of organization-level and team-level policies for implementing the quality plan.

5.2.2 REQUIREMENTS

The software requirements specification document should contain a written definition of the software functions.

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.

Parasoft Support

- A system for mapping quality plan requirements to development tasks and monitoring the implementation and validation of each requirement.
- Traceability through requirements-based testing, which links test cases, the requirements defined in the specification, and the related source code — providing real- time visibility into which requirements are actually working as expected, and which still require testing.
- Workflow automation for design document reviews.
- Automated orchestration of approval/sign-off tasks in the appropriate sequence, and with complete traceability.

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.

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.

Parasoft Support

- Policies specify design best practices that prevent common design pitfalls; ensure that the design is correct, consistent, complete, accurate, and testable; and help teams satisfy critical design attributes such as usability, performance, efficiency, scalability, or modularity.
- Workflow automation for design document reviews.
- Automated orchestration of approval/sign-off tasks in the appropriate sequence, and with complete traceability.

5.2.4 CONSTRUCTION OR CODING

Source code should be evaluated to verify its compliance with specified coding guidelines. Such guidelines should include coding conventions regarding clarity, style, complexity management, and commenting.

Source code evaluations are often implemented as code inspections and code walkthroughs. Such static analyses provide a very effective means to detect errors before execution of the code.

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.
- Tests for modules and functions can be traced to source code for the same modules and functions.

Parasoft Support

- Pattern-based static analysis ensures that the code meets uniform expectations around reliability, performance, security, and maintainability. Includes preconfigured templates for FDA.
- Data flow static analysis detects complex runtime errors without requiring test cases or application execution.
- Metrics analysis not only calculates metrics but also identifies specific pieces of code that exceed industry standard or customized metrics thresholds.
- Peer code inspection process automation automates and manages the peer code review workflow — including preparation, notification, and tracking — and reduces overhead by enabling code review on the desktop.
- Traceability through requirements-based testing, which links test cases, the requirements defined in the specification, and the related source code — providing real- time visibility into which requirements are actually working as expected, and which still require testing.

5.2.5 TESTING BY THE SOFTWARE DEVELOPER

Test plans and test cases should be created as early in the software development process as feasible.

Once the prerequisite tasks like code inspection have been successfully completed, software testing begins. It starts with unit level testing and concludes with system level testing.

Codebased 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.

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.

Parasoft Support

- A framework that allows developers to start testing each unit as soon as it is completed, and that supports the rapid addition of user-defined tests that verify software correctness and functionality.
- After examining the source code to determine how to test it, a wide variety of white-box test cases are automatically generated to check code robustness, exposing potential reliability problems.
- Automated identification and refactoring of unused code, duplicate code, and dead code.
- Coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge test suite efficacy and completeness. Coverage is defined as code coverage obtained by actually executing code with test cases — not simulated coverage.
- Automated integration-level and system-level testing.
- Runtime error detection efficiently identifies defects.only manifested at runtime.
- Memory error detection identifies difficult-to-track programming and memory-access errors, as well as potential defects and memory usage inefficiencies.

5.2.6 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. Documented evidence of all testing procedures, test input data, and test results should be retained.

Parasoft Support

- Step-by-step capture of user acceptance test processes. Each manual step is captured so the complete manual sequence can be easily retrieved, reviewed, and repeated — adding objective traceability to the process.

5.2.7 MAINTENANCE AND SOFTWARE CHANGES

When changes are made to a software system, either during initial development or during post release maintenance, sufficient regression analysis and testing should be conducted to demonstrate that portions of the software not involved in the change were not adversely impacted. This is in addition to testing that evaluates the correctness of the implemented changes.

Parasoft Support

- Automated execution of regression test suite that captures the code's current behavior as a baseline. Daily execution of this test suite ensures that the team is immediately alerted if code modifications impact or break existing functionality.
- A continuous regression testing process which ensures that the impacts of code modifications are identified and addressed daily, and the regression test suite stays in synch with the evolving application.
- A framework that supports the rapid addition of new tests that verify the correctness of the implemented changes.

TAKE THE NEXT STEP

Learn more about improving software quality with medical device software development testing solutions. Talk to one of our experts today.

ABOUT PARASOFT

Parasoft helps organizations continuously deliver quality software with its market-proven, integrated suite of automated software testing tools. Supporting the embedded, enterprise, and IoT markets, Parasoft's technologies reduce the time, effort, and cost of delivering secure, reliable, and compliant software by integrating everything from deep code analysis and unit testing to web UI and API testing, plus service virtualization and complete code coverage, into the delivery pipeline. Bringing all this together, Parasoft's award winning reporting and analytics dashboard delivers a centralized view of quality enabling organizations to deliver with confidence and succeed in today's most strategic ecosystems and development initiatives — cybersecure, safety-critical, agile, DevOps, and continuous testing.