Using TechExcel’s DevSuite to Achieve FDA Software Validation Compliance For Medical Software Device Development

The FDA requires medical software development teams to comply with its standards for software validation. The FDA’s General Principles of Software Validation outlines how to achieve this compliance and recommends a comprehensive defect prevention strategy where software validation and verification activities such as code reviews, static analysis, unit testing, and regression testing are conducted throughout the entire software development life cycle.

With years of experience helping Fortune 500 companies incorporate these and other best practices throughout the SDLC, TechExcel knows what it takes to rapidly bring organizations into FDA compliance and evolve a sustainable process for continued compliance. TechExcel supports FDA compliance with TechExcel DevSuite for Medical Device Software Development. TechExcel DevSuite is an Application Lifecycle Management platform that ensures that software can be produced consistently and efficiently with minimal risk. TechExcel DevSuite for Medical Device Software Development is a pre-configured system with workflows and best practices that assists organizations to meet FDA guidelines and medical device industry standards for software development.

Organizations using TechExcel DevSuite integrate requirements management and task management with test case management, defect prevention and end-to-end software verification and validation. The powerful workflow engine helps teams to uniformly apply the least burdensome practices and mitigate the safety risks associated with medical device application development.

TechExcel DevSuite for Medical Device Software Development features:

- Built-in configurable templates for FDA and IEC compliance
- Requirements, development, and test management
- Comprehensive traceability through the entire SDLC
- Easily build process for defect prevention, validation and verification
- A continuous policy-driven compliance process with real-time visibility

Background: The FDA General Principles of Software Validation

Software validation is a requirement of the FDA Quality System regulation, which was published in the Federal Register on October 7, 1996 and took effect on June 1, 1997.

The General Principles of Software Validation outlines general validation principles that the FDA considers to be applicable to the validation of medical device software or the validation of software used to design, develop, or manufacture medical devices. These validation requirements apply to:

- Software used as a component, part, or accessory of a medical device;
- Software that is itself a medical device (e.g., blood establishment software);
- Software used in the production of a device (e.g., programmable logic controllers in manufacturing equipment); and
- Software used in implementation of the device manufacturer’s quality system (e.g., software that records and maintains the device history record).
# TechExcel DevSuite Support for FDA General Principles of Software Validation

<table>
<thead>
<tr>
<th>FDA Principle</th>
<th>DevSuite Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>3.1 Software Verification and Validation</strong></td>
<td>• Workflow automations for enforcing static and dynamic analysis, code and document inspections, walkthroughs, design reviews and other verification activities.</td>
</tr>
</tbody>
</table>

Software verification 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. Software verification looks for consistency, completeness, and correctness of the software and its supporting documentation, as it is being developed, and provides support for a subsequent conclusion that software is validated.

| **4.1 Requirements** | • Fully integrated requirements management system that maps requirements and specifications to development tasks and testing templates for progress monitoring and validation. • Graphical reporting on requirement status. |

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.

| **4.2 Defect Prevention** | • Linking test templates to requirements and specifications ensures that developers know what standard their code will be held to. This removes the overhead of basic testing so that QA teams can focus on more difficult scenarios • Workflow rules can be put into place to ensure adherence to defect prevention practices and policies. |

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.

| **4.3 Time and Effort** | • A fully integrated system with processes in place for defect prevention and detection allows for software validation to begin as soon as requirements are entered into the system. • Preconfigured FDA workflow templates |

Preparation of software validation should begin early; i.e., during design and development planning and design input.

| **4.4 Software Life Cycle** | • Virtually any life cycle model can be implemented using DevSuite's flexible platform. Choose from Agile, Traditional or hybrid models to find the one that suits your team best. • Integrations and automations for the SDLC that help to ensure a high level of quality and efficiency. • Workflow rules allow you to setup a foundation for repeatable, sustainable quality processes. |

Software validation takes place within the environment of an established software life cycle. The software life cycle contains specific verification and validation tasks that are appropriate for the intended use of the software. This guidance does not recommend any particular life cycle models – only that they should be selected and used for a software development project.
### 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.

- Validation plans are expressed through the use of schedules, resource assignments, tasks and work items that are linked throughout the system.
- A system for linking test plans/templates to developments tasks as well as mechanisms to monitor the implementation and validation to each requirement or specification.
- Workflows and rules to clearly define and enforce validation plans.

### 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.

- Fully integrated system allows teams to easily plan and execute
- Optimized workflows enforce and support quality processes so they can become a sustainable part of the SDLC.
- Graphical reporting to monitor quality gates and thresholds.
- Preconfigured FDA workflow 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.

- Plan regression-testing cycles that run in parallel to new development testing cycles.
- Plan test cycles linked directly to the newest code so that teams are testing the areas with the highest risk factors.

### 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.

- Flexible test planning and templates allow for more complex testing matrices to be executed. These tests help ensure high quality for complex pieces of code.
- Easily gauge test coverage across multiple environments through the use of environment variables.
- Reports and trend graphs document validation work and improvements in quality.

### 4.9 Independence of Review

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

- Enforceable processes for review tasks can be built into the workflow along with Part 11 compliant identity verification.
## FDA Principle

**4.10 Flexibility and Responsibility**

Specific implementation of these software validation principles may be quite different from one application to another. 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.

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.

### DevSuite Support

- Easily implement a policy-driven, flexible, repeatable and traceable process that can span distributed environments and include both automated and manual tasks.
- Define general test templates and then schedule them for usage across multiple environments.
- Mitigate risk in outsourced or distributed environments with a truly multi-site system.
- Automatic defect creation when tests fail, ability to schedule and prioritize defects into development cycles.

## 5.1 Software Life Cycle Activities

This guidance does not recommend the use of any specific software life cycle model. Software developers should establish a software life cycle model that is appropriate for their product and organization. The software life cycle model that is selected should cover the software from its birth to its retirement. 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 occurs 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.

### DevSuite Support

- Fully adaptable and customizable system lets developers establish a software life cycle model that is appropriate for their product and organization.
- Enforceable workflows establish expectations and keep teams on track.
- Provide a repeatable framework that makes outcomes significantly easier to predict.
- Preconfigured FDA workflow 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.

### DevSuite Support

- Validation plans are expressed through the use of schedules, resource assignments, tasks and work items that are linked throughout the system
- A system for linking test plans/templates to developments asks as well as mechanisms to monitor the implementation and validation to each requirement or specification.
- Workflows and rules to clearly define and enforce validation plans.
## 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.

<table>
<thead>
<tr>
<th>FDA Principle</th>
<th>DevSuite Support</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>5.2.2. Requirements</strong></td>
<td>• Fully integrated requirements management system that maps requirements and specifications to development tasks and testing templates for progress monitoring and validation.&lt;br&gt;• Graphical reporting on requirement status.&lt;br&gt;• Workflow automations to handle design document reviews.&lt;br&gt;• Automated orchestration of approval/sign-off tasks in the appropriate sequence, and with complete traceability.&lt;br&gt;• Full traceability through requirements, development and testing with everything linked to the original requirement.</td>
</tr>
</tbody>
</table>

| **5.2.3. Design** | • Workflow specifies and enforces best practices to prevent common pitfalls and ensure that the design is accurate, complete, and testable.<br>• Automations and events for design document reviews.<br>• Automated orchestration of approval/sign-off tasks in the appropriate sequence, and with complete traceability. |

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.

| **5.2.4. Construction or Coding** | • Centralized knowledge base for storing coding guidelines.<br>• Linking test templates to requirements and specifications ensures that developers know what standard their code will be held to.<br>• Full traceability through requirements, coding and testing phases.<br>• Workflow states and events to enforce practices like code reviews, peer reviews.<br>• Integration with many SCM tools so that code can be linked directly to development tasks, tests and requirements. |

Firms frequently adopt specific coding guidelines that establish quality policies and procedures related to the software coding process. 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
### FDA Principle

<table>
<thead>
<tr>
<th><strong>5.2.5. Testing by the Software Developer</strong></th>
<th><strong>DevSuite Support</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Test plans and test cases should be created as early in the software development process as feasible. Once the prerequisite tasks (e.g., code inspection) have been successfully completed, software testing begins. It starts with unit level testing and concludes with system level testing. Code-based testing is also known as structural testing or &quot;white-box&quot; testing. It identifies test cases based on knowledge obtained from the source code, detailed design specification, and other development documents. Structural testing can identify &quot;dead&quot; 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 &quot;coverage&quot; and are a measure of completeness with respect to test selection criteria.</td>
<td>• Test plans linked directly to requirements and specifications can be created as soon as possible. Events can help enforce that test are created before an item is finalized. • Build a library of “white box” tests that can be linked to requirements and development items. • Framework supports rapid addition of user-defined tests to verify and validate tasks. • Environment variables allow for effective test coverage without over burdening QA teams. • Framework supports regression testing in tandem with new development testing to ensure that new code does not break existing functionality.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>5.2.6. User Site Testing</strong></th>
<th><strong>DevSuite Support</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td>• Testing matrices allow for step by step testing with ever step and its results captured.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>5.2.7. Maintenance and Software Changes</strong></th>
<th><strong>DevSuite Support</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>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 change(s).</td>
<td>• Baselining of requirements allow teams to easily see what’s changed since a previous version. • Continuous regression testing prevents defects from being introduced into stable functions. • New tests can be rapidly added to verify and validate new functionality.</td>
</tr>
</tbody>
</table>