General Principles of Software Validation; Final Guidance for Industry and FDA Staff

软件验证的基本原理；企业和FDA人员的最终指导准则

Document issued on发布日期: January 11, 2002

This document supersedes the draft document, "General Principles of Software Validation, Version 1.1, dated June 9, 1997.

本文件取代之前的草案“**1997**年**6**月**9**日的**1.1**版软件验证的基本原理 ”

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

Preface
前言

Public Comment 公众评论（意见/问题）

Comments and suggestions may be submitted at any time for Agency consideration to Dockets Management Branch, Division of Management Systems and Policy, Office of Human Resources and Management Services, Food and Drug Administration, 5630 Fishers Lane, Room 1061, (HFA-305), Rockville, MD, 20852. When submitting comments, please refer to the exact title of this guidance document. Comments may not be acted upon by the Agency until the document is next revised or updated.

For questions regarding the use or interpretation of this guidance which involve the Center for Devices and Radiological Health (CDRH), contact John F. Murray at (301) 594-4659 or email jfm@cdrh.fda.gov

For questions regarding the use or interpretation of this guidance which involve the Center for Biologics Evaluation and Research (CBER) contact Jerome Davis at (301) 827-6220 or email davis@cber.fda.gov.
（略）

Additional Copies 额外话题

CDRH Additional copies are available from the Internet at: www.fda.gov/MedicalDevices/DeviceRegulationandGuidance/GuidanceDocuments/UCM085281.htm. You may also send an e-mail request to dsmica@fda.hhs.gov to receive an electronic copy of the guidance or send a fax request to 301-847-8149 to receive a hard copy. Please use the document number (938) to identify the guidance you are requesting.

CBER Additional copies are available from the Internet at: http://www.fda.gov/cber/guidelines.htm, bywriting to CBER, Office of Communication, Training, and Manufacturers' Assistance (HFM40), 1401 Rockville Pike, Rockville, Maryland 20852-1448, or by telephone request at 1800-835-5709 or 301-827-1800. （略）

Table of Contents
目　录

General Principles of Software Validation
软件验证的一般原则

This document is intended to provide guidance. It represents the Agency's current thinking on this topic. It does not create or confer any rights for or on any person and does not operate to bind Food and Drug Administration (FDA) or the public. An alternative approach may be used if such approach satisfies the requirements of the applicable statutes and regulations.

SECTION 1. PURPOSE 第 1 部分 目的

This guidance outlines general validation principles that the Food and Drug Administration (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. This final guidance document, Version 2.0, supersedes the draft document, *General Principles of Software Validation, Version 1.1*, dated June 9, 1997. 本指导概括了FDA认为适用于医疗器械软件验证，或用于设计、开发，或生产医疗器械的软件验证的一般验证原则。

SECTION 2. SCOPE 第 2 部分 范围

This guidance describes how certain provisions of the medical device Quality System regulation apply to software and the agency's current approach to evaluating a software validation system. For example, this document lists elements that are acceptable to the FDA for the validation of software; however, it does not list all of the activities and tasks that must, in all instances, be used to comply with the law.

本指导描述了医疗器械质量系统法规的某些条款如何应用到软件，以及FDA评价一个软件验证系统的现行方法。例如，本文列出了FDA对于软件验证的可接受元素；但是，并未列出在一切情况下必须遵循法律的所有活动和任务。

The scope of this guidance is somewhat broader than the scope of validation in the strictest definition of that term. Planning, verification, testing, traceability, configuration management, and many other aspects of good software engineering discussed in this guidance are important activities that together help to support a final conclusion that software is validated.

严格上讲，本指导的应用范围比验证的范围更广泛一些。计划、确认、测试、追溯性、配置管理及本指导中讨论的良好软件工程的许多其他方面是重要的活动，它们有助于支持一个最终结论-软件是已验证过的。

This guidance recommends an integration of software life cycle management and risk management activities. Based on the intended use and the safety risk associated with the software to be developed, the software developer should determine the specific approach, the combination of techniques to be used, and the level of effort to be applied. While this guidance does not recommend any specific life cycle model or any specific technique or method, it does recommend that software validation and verification activities be conducted throughout the entire software life cycle.

本指导建议将软件生命周期管理和风险管理活动进行整合。根据预期用途和与开发的软件相关联的安全风险，软件开发人员应确定特定方法，使用的多个技术的组合，以及应用尝试程度。虽然本指导未推荐任何特定生命周期模式，或任何特定技术或方法，但建议整个软件生命周期需进行软件验证和确认活动。

Where the software is developed by someone other than the device manufacturer (e.g., off-the-shelf software) the software developer may not be directly responsible for compliance with FDA regulations.

若软件由某人而非器械生产商开发，如成品组件软件，这个软件开发者可能不直接负责FDA法规的符合性。

In that case, the party with regulatory responsibility (i.e., the device manufacturer) needs to assess the adequacy of the off-the-shelf software developer's activities and determine what additional efforts are needed to establish that the software is validated for the device manufacturer's intended use.

在这种情况下，负责合规的一方（即器械生产商）需要评价成品组件软件开发者活动的充分性，并确定是否需要其他尝试来证明软件已经验证过，并符合器械生产商的预期用途。

2.1. APPLICABILITY 适用范围

This guidance applies to:本指导适用于：

Software used as a component, part, or accessory of a medical device;
用作一个医疗器械的一个组件、部件，或配件的软件；
Software that is itself a medical device (e.g., blood establishment software);
本身即是一个医疗器械的软件（例如，血制品企业软件）
Software used in the production of a device (e.g., programmable logic controllers in manufacturing equipment); and
器械生产中使用的软件（例如，生产设备中使用的可编程逻辑控制器）；及
Software used in implementation of the device manufacturer's quality system (e.g., software that records and maintains the device history record).
用作器械生产商质量系统的软件（例如，记录和维护器械历史记录的软件）。

This document is based on generally recognized software validation principles and, therefore, can be applied to any software. For FDA purposes, this guidance applies to any software related to a regulated medical device, as defined by Section 201(h) of the Federal Food, Drug, and Cosmetic Act (the Act) and by current FDA software and regulatory policy. This document does not specifically identify which software is or is not regulated. 普遍认为，本文件是基于软件验证的原则，因此，适用于任何软件。从FDA角度看，按照食品药品化妆品法第201(h)条和现行FDA软件和监管方针的规定，本指导适用于任何与管制医疗器械相关的软件。本文件并未特地说明软件是否被监管。

2.2. AUDIENCE 公众

This guidance provides useful information and recommendations to the following individuals:
本指导为下列个人提供了有价值的信息和建议：

Persons subject to the medical device Quality System regulation
受医疗器械质量系统监管的人员
Persons responsible for the design, development, or production of medical device software
负责医疗器械软件的设计、开发或生产的人员
Persons responsible for the design, development, production, or procurement of automated tools used for the design, development, or manufacture of medical devices or software tools used to implement the quality system itself
负责自动工具或软件工具的设计、开发、生产或采购的人员，自动工具是用于医疗器械的设计、开发或制造，软件工具用于质量系统本身
FDA Investigators
FDA 调研人员
FDA Compliance Officers
FDA 合规办公室
FDA Scientific Reviewers
FDA 科学评审员

2.3. THE LEAST BURDENSOME APPROACH 最简便的方法

We believe we should consider the least burdensome approach in all areas of medical device regulation. This guidance reflects our careful review of the relevant scientific and legal requirements and what we believe is the least burdensome way for you to comply with those requirements. However, if you believe that an alternative approach would be less burdensome, please contact us so we can consider your point of view. You may send your written comments to the contact person listed in the preface to this guidance or to the

CDRH Ombudsman. Comprehensive information on CDRH's Ombudsman, including ways to contact him, can be found on the Internet at:
我们认为我们应考虑医疗器械法规所有领域中的最简便的方法。本指导反映了我们对相关科学和法律要求的仔细评审，以及我们认为的对于你来说符合那些要求的最简便的方式。然而，如果你认为一个替代的方法可能更好，请与我们联系，我们可给予考虑。你可发送书面评论至本指导封面列出的联系人或至CDRH的调查专员。更多关于CDRH调查专员的信息，包括联系方式，详见下面链接：
http://www.fda.gov/cdrh/resolvingdisputes/ombudsman.html.

2.4. REGULATORY REQUIREMENTS FOR SOFTWARE VALIDATION 软件验证的监管要求

The FDA's analysis of 3140 medical device recalls conducted between 1992 and 1998 reveals that242 of them (7.7%) are attributable to software failures. Of those software related recalls, 192 (or79%) were caused by software defects that were introduced when changes were made to the software after its initial production and distribution. Software validation and other related good software engineering practices discussed in this guidance are a principal means of avoiding such defects and resultant recalls.

从FDA对1992~1998年间认证的3140例医疗器械的召回事件的原因回顾分析看出，有242例是因软件失效导致，而其中192例是因为对软件最初版本进行修改后，导致软件系统出现缺陷并造成软件系统失效。软件验证及其相关良好软件工程开发规范（本指导讨论的）是避免此类缺陷及随后导致的召回事件的一个最重要的方式。

Software validation is 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.) Validation requirements apply to software used as components in medical devices, to software that is itself a medical device, and to software used in production of the device or in implementation of the device manufacturer's quality system.

软件验证是质量体系法规的要求，即1996年10月7日发表在FR（联合公报）上，并于1997年6月1日生效的法规（分别参见21CFR820部分和FR61卷52602）。验证需求的应用范围，包括用作医疗器械组件的软件，本身即是一个医疗器械的软件和生产设备使用的或在器械生产商质量系统中执行的软件。

Unless specifically exempted in a classification regulation, any medical device software product developed after June 1, 1997, regardless of its device class, is subject to applicable design control provisions. (See of 21 CFR §820.30.) This requirement includes the completion of current development projects, all new development projects, and all changes made to existing medical device software. Specific requirements for validation of device software are found in21 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.) The corresponding documented results from these activities can provide additional support for a conclusion that medical device software is validated.

除非分类法规中有特殊豁免，任何1997年6月1日后研发的医疗器械软件产品，不管其类别，应与设计控制条款相适用（参见21CFR§820.30.）这个要求包括现有研发项目，所有新研发项目及所有目前医疗器械软件的变更的完成。医疗器械软件验证的特殊需要参见21 CFR §820.30(g)。其他设计控制，如计划、输入、确认和审核，均是医疗器械软件必须具有的（参见21 CFR §820.30）。来自这些活动的相应记录的结果可为医疗器械软件验证的结论提供额外支持。

Any software used to automate any part of the device production process or any part of the quality system must be validated for its intended use, as required by 21 CFR §820.70(i). This requirement applies to 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.

按照21 CFR §820.70(i)，用于控制器械生产工艺的或质量系统的任何部分的所有软件必须验证以符合预期用途，这项规定适用于以下用途的软件，如自动控制器械的设计、检测、组件验收、生产、贴标、包装、销售、投诉处理或质量系统任何其他部分的自动化操作。

In addition, 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).) Such computer systems must be validated to ensure accuracy, reliability, consistent intended performance, and the ability to discern invalid or altered records.

另外，用于创建、修改及维护电子记录和管理电子签名的计算机系统也应进行验证(见 21 CFR §11.10(a))。这些计算机系统必须验证，以确保准确性、可靠性，始终符合预期性能，识别无效或变更记录的能力。

Software for the above applications may be developed in-house or under contract. However, software is frequently purchased off-the-shelf for a particular intended use. All production and/or quality system software, even if purchased off-the-shelf, should have documented requirements that fully define its intended use, and information against which testing results and other evidence can be compared, to show that the software is validated for its intended use.

上述应用程序软件可能由内部或合同商开发。然而，由于特殊预期用途，经常购买成品软件。所有生产和/或质量系统软件，即便是购买的软件，也应备有完全明确其预期用途的要求。

The use of off-the-shelf software in automated medical devices and in automated manufacturing and quality system operations is increasing. Off-the-shelf software may have many capabilities, only a few of which are needed by the device manufacturer. Device manufacturers are responsible for the adequacy of the software used in their devices, and used to produce devices. When device manufacturers purchase "off-the-shelf" software, they must ensure that it will perform as intended in their chosen application. For off-the-shelf software used in manufacturing or in the quality system, additional guidance is included in Section 6.3 of this document. For device software, additional useful information may be found in FDA's *Guidance for Industry, FDA Reviewers, and Compliance on Off-The-Shelf Software Use in Medical Devices.*

自动医疗设备和自动生产/质量系统操作用成品软件的使用日益增加。成品软件可有许多性能，但器械生产商仅需要其中的几个。器械生产商负责其器械使用的软件是否适当。当器械生产商购买“成品”软件时，他们必须确保软件将按照选用的应用程序运行。对于生产或质量系统使用的成品软件，也适用于本文第6.3部分的指导。对于器械软件，其他有用的信息可参见FDA的工业指导：*FDA评审人员，及医疗器械用成品软件的合规。*

2.5. QUALITY SYSTEM REGULATION VS PRE-MARKET SUBMISSIONS 质量系统法规 VS 上市前申请

This document addresses Quality System regulation issues that involve the implementation of software validation. It provides guidance for the management and control of the software validation process. The management and control of the software validation process should not be confused with any other validation requirements, such as process validation for an automated manufacturing process.

本文所述质量系统法规问题包括软件验证的实施，并为软件验证过程的管理和控制提供指导。软件验证过程的管理和控制不应与任何其他验证要求相混淆，例如一个自动生产过程的工艺验证。

Device manufacturers may use the same procedures and records for compliance with quality system and design control requirements, as well as for pre-market submissions to FDA. This document does not cover any specific safety or efficacy issues related to software validation. Design issues and documentation requirements for pre-market submissions of regulated software are not addressed by this document. Specific issues related to safety and efficacy, and the documentation required in pre-market submissions, should be addressed to the Office of Device Evaluation (ODE), Center for Devices and Radiological Health (CDRH) or to 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.

器械生产商可以使用符合质量系统和设计控制要求的相同的规程和记录，也可使用符合上市前递交的申请（FDA）。本文件不包括与软件验证有关的所有特殊安全性或有效性问题。本文没有叙述管制软件上市前申请的设计问题和文件要求。与安全性或有效性有关的特殊问题，以及上市前申请需要的文件，应写信给CDRH中心的ODE办公室或CBER的血液研究与评审办公室。FDA上市前申请指导文件参见附件A。

SECTION 3. CONTEXT FOR SOFTWARE VALIDATION 软件验证的背景

Many people have asked for specific guidance on what FDA expects them to do to ensure compliance with the Quality System regulation with regard to software validation. Information on software validation presented in this document is not new. Validation of software, using the principles and tasks listed in Sections 4 and 5, has been conducted in many segments of the software industry for well over 20 years.

许多人已在寻求关于FDA期待他们保证遵循与软件验证相关的质量系统法规的特殊指导。本文展现的软件验证信息不是新的。使用第4、5部分列出的原则和任务的软件验证已在软件产业的许多领域应用20多年了。

Due to the great variety of medical devices, processes, and manufacturing facilities, it is not possible to state in one document all of the specific validation elements that are applicable. However, a general application of several broad concepts can be used successfully as guidance for software validation. These broad concepts provide an acceptable framework for building a comprehensive approach to software validation. Additional specific information is available from many of the references listed in Appendix A.

由于医疗器械和生产设施的多样性，不可能在一个文件中阐述所有使用的特定验证元素。然而，几个广义概念的通用可成功用作软件验证的指导。这些广义的概念为构建一个广泛的软件验证方法提供了一个可接受框架。其他具体信息参见附件A中列出的参考。

3.1. DEFINITIONS AND TERMINOLOGY 定义和术语

Unless defined in the Quality System regulation, or otherwise specified below, all other terms used in this guidance are as defined in the current edition of the FDA *Glossary of Computerized System and Software Development Terminology*.

除非质量系统法规中已定义，或另有说明，本指导用到的所有其他术语在现行版FDA的*Glossary of Computerized System and Software Development Terminology*中均有定义。

The medical device Quality System regulation (21 CFR 820.3(k)) defines "**establish**" to mean "define, document, and implement." Where it appears in this guidance, the words "establish" and “established” should be interpreted to have this same meaning.

医疗器械质量系统法规(21 CFR 820.3(k))将"**establish**"定义为“define, document, and implement”。本文出现的"establish"和“established”应诠释为同上的意思。

Some definitions found in the medical device Quality System regulation can be confusing when compared to commonly used terminology in the software industry. Examples are requirements, specification, verification, and validation.

当与软件产业中使用的通用术语相比较时，可能会对医疗器械质量系统法规中规定的一些定义发生混淆。例如需求、规约（标准）、确认及验证。

3.1.1 Requirements and Specifications 法规要求和标准规范

While the Quality System regulation states that design input requirements must be documented, and that specified requirements must be verified, the regulation does not further clarify the distinction between the terms “requirement” and “specification.” A **requirement** can be any need or expectation for a system or for its software. Requirements reflect the stated or implied needs of the customer, and may be market-based, contractual, or statutory, as well as an organization's internal requirements. There can be many different kinds of requirements (e.g., design, functional, implementation, interface, performance, or physical requirements). Software requirements are typically derived from the system requirements for those aspects of system functionality that have been allocated to software. 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.

尽管质量系统法规规定必须记录设计输入要求和验证特殊要求，但法规未进一步阐明术语“requirement”和“specification”的根本区别。 **Requirement 是指对一个系统及其软件的任何要求或预期。**Requirements反映了客户规定的或指定的需要，也可能是基于市场的，合同的或法定的，还

有组织内部的要求。可能有许多不同种类的要求（例如，有关设计、功能、安装启用、接口、性能或物理的要求）。软件需求通常来自分配到软件的系统功能方面的系统需求。软件需求通常以功能术语阐明，并随着一个开发项目的进展进行明确、提炼和更新。能够准确地、完整地记录软件需求是生成软件成功验证的一个至关重要因素。

A **specification** is 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 and usually indicates the means and the criteria whereby conformity with the requirement can be checked. There are many different kinds of written specifications, e.g., system requirements specification, software requirements specification, software design specification, software test specification, software integration specification, etc. All of these documents establish “specified requirements” and are design outputs for which various forms of verification are necessary.

一个需求规约定义为“阐明需求的文件” (见 21 CFR §820.3(y).)。它可指或包括图示、模型，或其他相关文件，通常指明与被检查的需求相一致的方法和标准。有不同种类的书面需求，例如，系统需求规约、软件需求规约、软件设计标准、软件测试标准、软件集成标准等等。所有这些文件均建立“特定需求”，均是各种必要验证形式的设计输出。

3.1.2 Verification and Validation 软件确认和验证

The Quality System regulation is harmonized with *ISO 8402*:1994, which treats “verification” and “validation” as separate and distinct terms. On the other hand, many software engineering journal articles and textbooks use the terms "verification" and "validation" interchangeably, or in some cases refer to software "verification, validation, and testing (VV&T)" as if it is a single concept, with no distinction among the three terms.

质量系统法规与*ISO 8402*:1994一致，将“确认”和“验证”作为独立的和有区别的名词。一方面，许多软件工程期刊论文和教科书交互使用这两个名词，或有时候称为软件“确认、验证和测试(VV&T)”，好像是一个概念，没有区别。

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. Software testing is one of many verification activities intended to confirm that software development output meets its input requirements. Other verification activities include various static and dynamic analyses, code and document inspections, walkthroughs, and other techniques.

软件确认可提供客观证据，证明软件开发生命周期的一个特殊阶段的设计输出可满足那个阶段的所有特定需求。在开发过程中，软件确认旨在寻求软件及其支持文件记录的一致性、完整性和正确性，并为后面的结论（已经验证的软件）提供支持。软件测试是许多确认活动的一项内容，旨在确认软件开发输出满足输入要求。其他的确认活动包括各种静态和动态分析、代码和文件安全检查、走查和其他技术。

Software validation is a part of the design validation for a finished device, but is not separately defined in the Quality System regulation. For purposes of this guidance, FDA considers software validation to be “**confirmation by examination and provision of objective evidence that software specifications conform to user needs and intended uses, and that the particular requirements implemented through software can be consistently fulfilled.**” In practice, software validation activities may occur both during, as well as at the end of the software development life cycle to ensure that all requirements have been fulfilled. Since software is usually part of a larger hardware system, the validation of software typically includes evidence that 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.

软件验证是一个成品器械设计验证的一部分，但是在质量系统法规中没有单独定义。针对本指南的多个用途，FDA认为软件验证是指“通过检查和客观证据规定，证实软件规约符合用户需求和预期用途，并且这个通过软件实现的特有需求可始终如一地得到满足”。实际上，软件验证活动也出现在软件开发生命周期的开始、中间和结束时段，以确保所有需求已被实现。尽管软件通常是一个较大的硬件系统的一部分，但软件验证应证明，所有软件需求已正确和完整实现，并可追溯到系统需求。已验证软件的结论高度依赖广泛的软件测试、审查、分析和软件开发生命周期每个阶段执行的其他确认任务。模拟使用环境进行的器械软件功能测试及用户现场测试通常作为自动化软件器械的一个总体设计验证程序的组成部分。

Software verification and validation are difficult because a developer cannot test forever, and it is hard 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. Measures such as defects found in specifications documents, estimates of defects remaining, testing coverage, and other techniques are all used to develop an acceptable level of confidence before shipping the product. The level of confidence, and therefore the level of software validation, verification, and testing effort needed, will vary depending upon the safety risk (hazard) posed by the automated functions of the device. Additional guidance regarding safety risk management for software may be found in Section 4 of FDA’s *Guidance for the Content of Pre-market Submissions for Software Contained in Medical Devices,* and in the international standards *ISO/IEC 14971-1* and *IEC 60601-1-4* referenced in Appendix A.

软件确认与验证是不同的，因为一个开发人员不能进行永远测试，而且很难知道多少证据是充足的。很大程度上，软件验证是件开发“信任符合度”的事情，即器械符合对自动软件功能和器械特性的所有需求和用户预期。诸如标准文件中缺陷发现、缺陷残留估算、测试覆盖及其它技术等措施，应在产品发布前，均用来开发一个信任符合度的可接受标准。因此，这个符合度，软件验证、确认和需要的测试尝试的程度将取决于器械自动功能产生的安全风险（危害）。其他软件安全风险管理指导可参见FDA指导“ *Guidance for the Content of Pre-market Submissions for Software Contained in Medical Devices* ”的第4部分，以及附件A中引用的国际标准*ISO/IEC 14971-1* and *IEC 60601-1-4*。

3.1.3 IQ/OQ/PQ

For many years, both FDA and regulated industry have attempted to understand and define software validation within the context of process validation terminology. For example, industry documents and other FDA validation guidance sometimes describe user site software validation in terms of installation qualification (IQ), operational qualification (OQ) and performance qualification (PQ). Definitions of these terms and additional information regarding IQ/OQ/PQ 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.

多年来，FDA同受监管的企业尝试着以工艺验证的理论来理解和定义软件验证。例如，企业文件和其他FDA验证指导有时用IQ 、OQ、 PQ来描述用户现场软件验证。这些术语定义和其他IQ/OQ/PQ的信息可参见FDA的两个指导“*Guideline on General Principles of Process Validation*, dated May 11, 1987,”和“*Glossary of Computerized System and Software Development Terminology,* dated August 1995”。

While IQ/OQ/PQ terminology has served its purpose well and is one of many legitimate ways to organize software validation tasks at the user site, this terminology may not be well understood among many software professionals, and it is not used elsewhere in this document. However, 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.

虽然IQ/OQ/PQ术语已能很好的实现用途，并作为一个合理方式组织用户现场软件验证活动，但是许多软件专业人员可能不太理解这些术语，而且也没在本文件的其他地方使用。但不管怎样，当寻求和提供有关软件验证信息时，FDA人员和器械生产商仍需要意识到这些术语的差别。

3.2. SOFTWARE DEVELOPMENT AS PART OF SYSTEM DESIGN 软件设计--软件开发

The decision to implement system functionality using software is one that is typically made during system design. Software requirements are typically derived from the overall system requirements and design for those aspects in the system that are to be implemented using software. There are user needs and intended uses for a finished device, but users typically do not specify whether those requirements are to be met by hardware, software, or some combination of both. Therefore, software validation must be considered within the context of the overall design validation for the system.

使用软件实现系统功能是系统设计过程中的一个代表性的决定。软件需求通常来自整体系统需求和使用软件实现的系统的那些方面的设计。有一个成品器械的用户需求和预期用途，但用户通常没有详细说明是否硬件、软件或某些两者的组合满足那些需求。因此，软件验证必须在的系统总体设计验证的背景下考虑。

A documented requirements specification represents the user's needs and 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. The correctness and completeness of both the system requirements and the software requirements should be addressed as part of the design validation process for the device. Software validation includes confirmation of conformance to all software specifications and confirmation that all software requirements are traceable to the system specifications. Confirmation is an important part of the overall design validation to ensure that all aspects of the medical device conform to user needs and intended uses.

一个需求规约文件代表了产品研发的用户需求和预期用途。那么软件验证的首要目标是证明所有的完成的软件产品符合所有文件化的软件和系统需求。系统需求和软件需求的正确和完整性应是器械设计验证过程的一部分。软件验证包括确认符合所有软件标准，及确认所有软件需求可追溯至系统规约。确认是整体设计验证的一个重要部分，它确保医疗器械的所有方面符合用户需求和预期用途。

3.3. SOFTWARE IS DIFFERENT FROM HARDWARE 软件不同于硬件

While software shares many of the same engineering tasks as hardware, it has some very important differences. For example:

尽管软件拥有如同硬件一样的工程任务，但是却有很大不同。例如：

The vast majority of software problems are traceable to errors made during the design and development process. While the quality of a hardware product is highly dependent on design, development and manufacture, the quality of a software product is dependent primarily on design and development with a minimum concern for software manufacture. Software manufacturing consists of reproduction that can be easily verified. It is not difficult to manufacture thousands of program copies that function exactly the same as the original; the difficulty comes in getting the original program to meet all specifications.

绝大多数的软件问题可归咎于设计和开发过程中出现的错误。尽管硬件产品的质量高度依赖设计、开发和生产，但是软件产品的质量主要依赖极少关注软件生产的设计和开发。软件生产包括复制，其很容易被确认。生产成千上万的同源程序一样准确运行的程序副本并不困难；困难在于使源程序符合所有标准。

One of the most significant features of software is branching, i.e., the ability to execute alternative series of commands, based on differing inputs. This feature is a major contributing factor for another characteristic of software – its complexity. Even short programs can be very complex and difficult to fully understand.

软件最重要的一个特点是分支模型，即基于不同的输入执行交错命令的能力。这个特点是软件另一特性的主要贡献因素-复杂性。若不采用分支，甚至完全理解简短编写的程序也是极其复杂和困难的。

Typically, testing alone cannot fully verify that software is complete and correct. In addition to testing, other verification techniques and a structured and documented development process should be combined to ensure a comprehensive validation approach.

通常，单独测试不能完全确认软件是完整和正确的。除了测试，应将其他确认技术和一个结构

化和文件化的开发过程进行整合，以确保一个综合的验证方法。

Unlike hardware, software is not a physical entity and does not wear out. In fact, software may improve with age, as latent defects are discovered and removed. However, as software is constantly updated and changed, such improvements are sometimes countered by new defects introduced into the software during the change.

与硬件不同，软件不是一个物理实体，也不能耗竭。事实上，当潜在的缺陷被发现和移除时，软件可以随着使用年限得到改进。然而，当软件经历持续更新和变更时，这些改进有时会遭遇变更过程中引入的新缺陷。

Unlike some hardware failures, software failures occur without advanced warning. The software's branching that allows it to follow differing paths during execution, may hide some latent defects until long after a software product has been introduced into the marketplace.

与有些硬件故障不同， 软件故障的出现没有事先预警。软件配置管理的分支模型容许在执行过程中跟踪不同的路径，也可隐藏一些潜在缺陷，直至一个软件产品进入卖场后很长一段时间。

Another related characteristic of software is the speed and ease with which it can be changed. This factor can cause both software and non-software professionals to believe that software problems can be corrected easily. Combined with a lack of understanding of software, it can lead managers to believe that tightly controlled engineering is not needed as much for software as it is for hardware. In fact, the opposite is true. **Because of its complexity, the development process for software should be even more tightly controlled than for hardware, in order to prevent problems that cannot be easily detected later in the development process**.

另一软件相关特性是可改变的速度和便捷性。这个因素可引起软件和非软件专家认为软件问题很容易被改正。加上对软件缺乏理解，导致管理者认为软件同硬件一样并不需要严格控制工程。事实上，则相反。因为软件的复杂性，软件开发过程的严格控制程度甚至应超过硬件，以阻止开发过程中不易检测的问题。

Seemingly insignificant changes in software code can create unexpected and very significant problems elsewhere in the software program. The software development process should be sufficiently well planned, controlled, and documented to detect and correct unexpected results from software changes.

看起来似乎无关紧要的软件代码变更可以在软件程序的其他地方产生不可预计和非常重要的问题。软件开发过程应充分计划、控制和记录，以探测和改正出乎预料的软件变更结果。

Given the high demand for software professionals and the highly mobile workforce, the software personnel who make maintenance changes to software may not have been involved in the original software development. Therefore, accurate and thorough documentation is essential.

考虑到对软件专家的高度需求和高度的流动劳动力，负责软件变更维护的软件人员可能不参与最初软件的开发活动。因此，准确和全面的文件是必不可少的。

Historically, software components have not been as frequently standardized and interchangeable as hardware components. However, medical device software developers are beginning to use component-based development tools and techniques. Object-oriented methodologies and the use of off-the-shelf software components hold promise for faster and less expensive software development. However, component-based approaches require very careful attention during integration. Prior to integration, time is needed to fully define and develop reusable software code and to fully understand the behavior of off-the-shelf components.

For these and other reasons, software engineering needs an even greater level of managerial scrutiny and control than does hardware engineering.

历史资料显示，一直认为软件组件不能经常被标化，并同硬件组件互换。然而，医疗器械软件开发者正开始使用基于组件的开发工具和技术。面向对象的方法和成品软件组件的应用为快捷和价廉的软件开发带来希望。然而，在集成时，基于组件的方法要求极其小心的处理。在集成前，需要时间完全地定义和开发可重用软件代码，并完全地理解成品组件的行为。

针对这些和其他原因，软件工程需要一个比硬件工程更高水平的安全管理和控制。

3.4. BENEFITS OF SOFTWARE VALIDATION 软件验证的益处

Software validation is a critical tool used to assure the quality of device software and software automated operations. Software validation can 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. Software maintenance can represent a very large percentage of the total cost of software over its entire life cycle. An established comprehensive software validation process helps to reduce the long-term cost of software by reducing the cost of validation for each subsequent release of the software.

软件验证是一个关键工具，可保证器械软件的质量和软件自动化运行。软件验证能够增加器械的可用性和可靠性，减少故障率，更少的召回和改正行动，患者和用户的低风险，以及减少器械生产商的责任。软件验证也可减少长期的投入，更加便捷和较少投入地进行可靠软件修复和重新生效变更软件。

3.5 DESIGN REVIEW 设计评审

Design reviews are documented, comprehensive, and systematic examinations of a design to evaluate the adequacy of the design requirements, to evaluate the capability of the design to meet these requirements, and to identify problems. While there may be many informal technical reviews that occur within the development team during a software project, a formal design review is more structured and includes participation from others outside the development team. Formal design reviews may reference or include results from other formal and informal reviews. Design reviews may be conducted separately for the software, after the software is integrated with the hardware into the system, or both. 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.

设计评审是文件化的、广泛的和系统审核的一个设计，用以评估设计需求的充分性，设计是否满足这些需求，以及识别问题。虽然一个软件项目的开发团队会出现许多非正式技术评审，一个正式设计评审应有良好的组织，并包括团队外的其他人员的参与。

正式设计评审可能引用或包括其他正式和非正式评审的结果。在软件在硬件系统平台上集成，或者两个系统分别集成后，可对软件单独进行设计评审。设计评审应包括开发计划的审查，需求规约，设计标准，测试计划和程序，所有其他与项目相关的文件和活动，定义的生命周期的每阶段的确认结果，以及全部器械的验证结果。

Design review is a primary tool for managing and evaluating development projects. For example, formal design reviews allow management to confirm that all goals defined in the software validation plan have been achieved. 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 review is especially important at or near the end of the requirements activity, before major resources have been committed to specific design solutions. Problems found at this point can be resolved more easily, save time and money, and reduce the likelihood of missing a critical issue.

设计评审是管理和评估开发项目的一个主要工具。例如，正式设计评审容许管理人员确认所有软件验证计划中的目标已完成。质量系统法规要求在器械设计期间至少进行一个正式设计评审。然而，推荐进行多重设计评审（例如，在每个软件生命周期活动末期，制定下个活动）。在重要资源已用作特定设计解决方案以前，在需求活动末期或邻近末期时正式设计评审尤为重要。此时发现的问题会更加容易解决，节省时间和金钱，并减少漏掉一个关键问题的可能性。

Answers to some key questions should be documented during formal design reviews. These include:

在正式设计评审期间，应记录某些关键问题的回答，包括：

Have the appropriate tasks and expected results, outputs, or products been established for each software life cycle activity?

已经为每个软件生命周期活动建立适当任务和预期结果、输出或产品？

Do the tasks and expected results, outputs, or products of each software life cycle activity:
每个软件生命周期活动的任务和预期结果、输出或产品：
Comply with the requirements of other software life cycle activities in terms of correctness, completeness, consistency, and accuracy?
依据正确性、完整性、持续性和准确性，是否遵照其他软件生命周期活动的需求？
Satisfy the standards, practices, and conventions of that activity?
是否满足活动的标准、实践和实施惯例？
Establish a proper basis for initiating tasks for the next software life cycle activity?
是否为启动下次软件生命周期活动的任务建立一个合适的基础？

SECTION 4. PRINCIPLES OF SOFTWARE VALIDATION 软件验证的法则

This section lists the general principles that should be considered for the validation of software.
本部分列举了软件验证的一般法则

4.1. REQUIREMENTS 法规要求

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 (Ref: 21 CFR 820.3(z) and (aa) and 820.30(f) and (g)). 软件需求规约为验证和检验提供了一个基准线。若没有它，则不能完成软件验证过程（参见21 CFR 820.3(z) and (aa) and 820.30(f) and (g)）。

4.2. DEFECT PREVENTION 缺陷预防

Software quality assurance needs to focus on preventing the introduction of defects into the software development process and not on trying to “test quality into” the software code after it is written. Software testing is very limited in its ability to surface all latent defects in software code. For example, the complexity of most software prevents it from being exhaustively tested. **Software testing is a necessary activity. However, in most cases software testing by itself is not sufficient to establish confidence that the software is fit for its intended use.** 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. The “best mix” of methods depends on many factors including the development environment, application, size of project, language, and risk.
软件质量保证是必要的，以杜绝软件开发过程中引入的缺陷，但不是在软件编码完成后尝试测试编码的质量。软件测试对检测软件编码中存在的所有潜在缺陷的能力是极其有限的。例如，大多数软件的复杂度妨碍了它被详尽彻底地测试。软件测试是一项必要活动。然而，在多数情况下，单独的软件测试不足以说明软件适合其预期适用。为此，软件开发人员应运用方法和技术的组合体来预防软件错误，探测出现的软件错误。“最佳组合”中的方法依赖许多因素，包括开发环境、应用程序、项目大小、语言和风险。

4.3. TIME AND EFFORT 时间和尝试

To build a case that the software is validated requires 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.
创建一个软件被验证的案例需要时间和尝试。软件验证的制备应尽早开始，也就是，在设计和开发计划及设计输入时。软件验证的最终结论应以软件整个生命周期中执行的计划着的尝试中收集的证据未基础。

4.4. SOFTWARE LIFE CYCLE 软件生命周期

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. This guidance does not recommend any particular life cycle models – only that they should be selected and used for a software development project.

软件验证发生在一个已制定的软件生命周期的环境内。这个软件生命周期包括软件开发任务和支持软件验证尝试的必要文件。另外，这个软件生命周期包括特殊核实和适合于这个软件预期用途的验证任务。

4.5. PLANS 计划

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.

软件验证过程通过使用一个计划进行确定和控制。软件验证计划确定凭借软件验证尝试要完成什么。软件验证计划是一个重要的质量系统工具。软件验证计划详细说明如下方面：适用范围、方法、资源、安排及活动、任务和工作项目的类型和范围。

4.6. PROCEDURES 规程

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.

软件验证过程凭借规程进行。这些规程建立怎样执行软件验证尝试。这个规程应确定特殊行动或完成单个验证活动、任务和工作项目的行动顺序。

4.7. SOFTWARE VALIDATION AFTER A CHANGE 变更后的软件验证

Due to the complexity of software, a seemingly small local change may have a significant global system impact. When any change (even a small change) is made to the software, the validation status of the software needs to be re-established. **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.** Based on this analysis, 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. Design controls and appropriate regression testing provide the confidence that the software is validated after a software change.

由于软件的复杂性，一个看似小的局部变更或许会对全球系统产生一个重要的影响。当任何变化（即使一个小变化）发生时，软件的验证状态需要被再次验证。**无论何时软件变更，应进行一个验证分析，不仅为了这个单独变更的验证，而且也为了确定整个软件系统发生变更的程度和影响**。基于此次分析，软件开发者应执行一个适当水平的软件回归分析测试，以表明未变更，但易受攻击的系统组份没受到不利影响。设计控制和适当的软件回归分析测试为软件变更后验证提供了信心。

4.8. VALIDATION COVERAGE 验证范围

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. For lower risk devices, only baseline validation activities may be conducted. As the risk increases additional validation activities should be added to cover the additional risk. Validation documentation should be sufficient to demonstrate that all software validation plans and procedures have

been completed successfully.
验证范围应基于软件的复杂性和安全风险，而不是公司规模或资源约束。验证活动、任务和工作项目的选择应与软件设计的和软件特殊用途相关的风险的复杂性相适应。对于较低风险的设备，或许仅执行基准验证活动。随着风险增加，额外验证活动也相应增加，以覆盖增加的风险。验证文件应足够证明所有的软件验证计划和规程顺利地完成。

4.9. INDEPENDENCE OF REVIEW 独立评审

Validation activities should be conducted using the basic quality assurance precept of “independence of review.” Self-validation is extremely difficult. When possible, an independent evaluation is always better, especially for higher risk applications. Some firms contract out for a third-party independent verification and validation, but this solution may not always be feasible. 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. Smaller firms may need to be creative in how tasks are organized and assigned in order to maintain internal independence of review.
验证活动的执行应使用基本质量保证规则“独立评审”。自我验证是极其困难的。若可能，一个独立评估总会更好些，特别对于存在更高风险的应用程序。某些公司外包给第三方进行独立验证和确认，但是这种解决方案并不可取。另外的方式，是指定未参与一个特别设计或未参与实施的内部人员，但他们却有着足够的评估项目和执行验证和确认活动的知识。小一点的公司可能需要在一些问题上进行创新，如组织和指派什么任务，以维护内部独立评审。

4.10. FLEXIBILITY AND RESPONSIBILITY 灵活性和责任

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.
这些软件验证法则的特殊实施从一个到另一个应用程序或许有很大不同。设备生产商可灵活选择怎样应用这些验证法则，但仍有责任证明软件已被验证。
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. FDA regulated medical device applications include software that:
软件应在各种运行环境中，并为多种带有不同风险水平的器械进行设计、开发、验证及监管。FDA监管的医疗器械应用程序如：

Is a component, part, or accessory of a medical device; 一个医疗器械的组件、部件或配件

Is itself a medical device; or 本身是一个医疗器械；或

Is used in manufacturing, design and development, or other parts of the quality system.
用于生产、设计和开发，或质量系统的其他部件。

In each environment, software components from many sources may be used to create the application (e.g., in-house developed software, off-the-shelf software, contract software, shareware). In addition, software components come in many different forms (e.g., application software, operating systems, compilers, debuggers, configuration management tools, and many more). The validation of software in these environments can be a complex undertaking; **therefore, it is appropriate that all of these software validation principles be considered when designing the software validation process. The resultant software validation process should be commensurate with the safety risk associated with the system, device, or process.**
每个运行环境，来自许多资源的软件构件可用于创建应用程序（例如，内部开发的软件、成品软件、合同软件、共享软件）。另外，软件构件以多种不同形式起作用（如应用程序软件、操作系统、编译程序、调试器、配置管理工具及许多其他的）。这些运行环境中的软件验证是一项复杂任务；因此，当设计软件验证过程时，考虑所有这些软件验证原则是适当的。设计成的软件验证过程应与系统、器械或过程有关的安全风险相称。

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.
软件验证活动和任务也许是分散的，出现在不同的地方，并由不同组织执行。然而，不管任务如何分配，合同关系、组件来源或开发环境，器械生产商或规范制定者始终有责任保证软件是经验证的。

SECTION 5. ACTIVITIES AND TASKS 活动和任务

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 or may be iterated many times, depending on the life cycle model used and the scope of changes made as the software project progresses.
软件验证借助软件开发生命周期的不同阶段计划的和执行的一系列活动和任务来完成。这些任务或许一次出现或重复出现多次，取决于随着项目进度使用的生命周期模型及所做变更的范围。

5.1. SOFTWARE LIFE CYCLE ACTIVITIES 软件生命周期活动

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:
本指导并不建议使用任何特殊软件生命周期模型。软件开发者应建立一个适合他们产品和组织的软件生命周期模型。选出的这个模型应覆盖软件整个周期，即从产生到结束。经典的软件生命周期模型活动包括：

Quality Planning 质量计划
System Requirements Definition 系统需求定义
Detailed Software Requirements Specification 详细的软件需求规约
Software Design Specification 软件设计规格说明规范
Construction or Coding 组建或编码
Testing 测试
Installation 安装
Operation and Support 运行和支持
Maintenance 维护
Retirement 退役

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. Several software life cycle models (e.g., waterfall, spiral, rapid prototyping, incremental development, etc.) are defined in FDA's *Glossary of Computerized System and Software Development Terminology*, dated August 1995. These and many other life cycle models are described in various references listed in Appendix A.
确认、测试及其他支持软件验证的任务出现在每个活动中。一个生命周期模型以多种方式组建这些软件开发活动，并为监控和控制软件开发项目提供一个框架。几个软件生命周期模型（例如，瀑布型、螺旋型、快速原型法、增量模型等）在1995年8月FDA出版的“计算机系统和软件开发术语”中被定义。附件A中所列出的各种参考是对这些模型以及其他生命周期模型的描述。

5.2. TYPICAL TASKS SUPPORTING VALIDATION 典型的任务辅助性验证

For each of the software life cycle activities, there are certain "typical" tasks that support a conclusion that the software is validated. However, the specific tasks to be performed, their order of performance, and the

iteration and timing of their performance will be dictated by the specific software life cycle model that is selected and the safety risk associated with the software application. For very low risk applications, certain tasks may not be needed at all. However, the software developer should at least consider each of these tasks and should define and document which tasks are or are not appropriate for their specific application. The following discussion is generic and is not intended to prescribe any particular software life cycle model or any particular order in which tasks are to be performed.
对于每个软件生命周期活动，都有某些“典型”任务来支持软件验证的一个结论。然而，执行的这个特殊任务，他们执行的顺序及他们执行反复和时间的选择将受制于选择的这个特殊软件生命周期模型和与软件应用相关的安全风险。对于极低风险的应用软件，某些任务可能根本不需要。然而，软件开发者至少应考虑每一个任务，定义和记录哪些任务对于他们的具体应用是适当的或是不适当的。

5.2.1. Quality Planning 质量计划

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. The plan should include:
设计和开发计划应集中在对必要任务、异常报告和解决的规程、必要资源及包含一个正式设计评审的管理评审进行鉴别的一项计划上。应识别一个软件生命周期模型及相关活动，还有每个软件生命周期活动所必须的那些任务。计划应包括：

The specific tasks for each life cycle activity; 每个软件生命周期活动的特殊任务
Enumeration of important quality factors (e.g., reliability, maintainability, and usability);
重要质量因素的枚举（如，可靠性、易维护性和可用性）；
Methods and procedures for each task; 每项任务的方法和规程；
Task acceptance criteria;
任务验收准则；
Criteria for defining and documenting outputs in terms that will allow evaluation of their conformance to input requirements;
明确地定义和记录输出的准则，将对输出对输入要求的符合程度进行评估；
Inputs for each task; 每项任务的输入；
Outputs from each task; 每项任务的输出；
Roles, resources, and responsibilities for each task; 每项任务的角色、资源和责任；
Risks and assumptions; and 风险和假设；和
Documentation of user needs. 用户需求文件。

Management must identify and provide the appropriate software development environment and resources. (See 21 CFR §820.20(b)(1) and (2).) Typically, each task requires personnel as well as physical resources. 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.
管理必须鉴别和提供适当的软件开发环境和资源（21 CFR §820.20(b)(1) and (2).）。具有代表性的是，每个任务需要人员和物理资源。计划应鉴别每项任务所需的人员、设施和设备资源，以及风险（危险）管理扮演的角色。应开发一个配置管理计划，指导和控制并行开发活动和确保适当的信息交流和文件定制。控制是必要的，以确保在所有规范文件的批准版本、源代码、目标代码和包含一个软件系统测试集中有正确而积极的对应性。控制也应确保对当前批准版本的正确辨别和有权使用。
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.
应创建规程，通过验证或其它活动来报告和解决软件异常问题。管理应能识别报告并详细说明内容、格式及每个报告的可靠组织要素。规程对于软件开发结果的评审和批准也是必要的，包括这些评审和批准的可靠组织要素。

Typical Tasks – Quality Planning 典型任务-质量计划

- Risk (Hazard) Management Plan 风险（危害）管理计划
- Configuration Management Plan 配置管理计划
- Software Quality Assurance Plan软件质量保证计划
  –Software Verification and Validation Plan -软件验证和确认计划
  - Verification and Validation Tasks, and Acceptance Criteria验证和确认任务及 验收标准
  - Schedule and Resource Allocation (for software verification and validation activities) 计划安排和资源配置（对于软件确认和验证活动）
  - Reporting Requirements报告要求
    –Formal Design Review Requirements正式设计评审要求
    –Other Technical Review Requirements其它技术评审要求
- Problem Reporting and Resolution Procedures 问题报告和解决规程
- Other Support Activities 其它支持活动

5.2.2. Requirements 需求

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.
需求开发包括器械及其预期用途等信息的鉴别、分析及记录。特别重要的部分包括硬件/软件的系统功能配置、运行条件、用户特点、潜在危害及预期任务。另外，需求应清晰阐述软件的预期用途。
The software requirements specification document should contain a written definition of the software functions. It is not possible to validate software without predetermined and documented software requirements. Typical software requirements specify the following:
软件需求规约文件应包括一个软件功能的书面定义。没有预先确定的和记录的软件需求，是不可能进行软件验证的。具有代表性的软件需求如下：

All software system inputs; 所有软件系统的输入；

All software system outputs; 所有软件系统的输出；

All functions that the software system will perform;将执行软件系统的所有功能；

All performance requirements that the software will meet, (e.g., data throughput, reliability, and timing); 软件将满足的所有性能需求（例如，数据吞吐量，可靠性和时间选择）

The definition of all external and user interfaces, as well as any internal software-to-system interfaces;
所有外部和用户接口的定义，以及所有内部软件系统接口；

How users will interact with the system; 用户将怎样与系统相互作用；

What constitutes an error and how errors should be handled; 是什么构成了一个错误， 以及错误怎样被处理；

Required response times; 要求的响应时间；

The intended operating environment for the software, if this is a design constraint (e.g., hardware platform, operating system);
软件预期的操作运行环境，假如这是一个设计约束（例如，硬件平台，操作系统）

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. 软件接受的所有范围、限度、默认值和比值；和
软件实施的所有安全相关的需求、规范、特征、或功能。

Software safety requirements are derived from a technical risk management process that is closely integrated with the system requirements development process. 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.). From this analysis, it should be possible to identify the most appropriate measures necessary to prevent harm.

软件安全需求来自一个技术风险管理过程，其紧密地与系统需求开发过程相结合。软件需求规约应清楚地鉴别一个系统软件故障产生的潜在危害，以及软件实施的所有安全需求。

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:

质量系统法规需要一个机制来说明不完整的、模棱两可的或相矛盾的需求（见21 CFR 820.30(c).）应评估软件需求规约中确定的每个需求（如，硬件、软件、用户、操作员接口和安全）的准确性、完整性、一致性、可检测性、正确性和清晰度。例如，软件需求应被评估以核实：

There are no internal inconsistencies among requirements; 需求间没有内部矛盾；

All of the performance requirements for the system have been spelled out; 已经清楚说明系统所有的性能需求；

Fault tolerance, safety, and security requirements are complete and correct;
故障容差、安全和安全性要求是完整和正确的；

Allocation of software functions is accurate and complete; 软件功能配置是正确的和完整的；

Software requirements are appropriate for the system hazards; and 软件需求考虑到系统危害；

All requirements are expressed in terms that are measurable or objectively verifiable.
所有需求均明确表达，是可测量或客观检验的。

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.

应执行一个软件需求的可追溯分析，以便对来自系统需求的软件需求和风险分析结果进行追溯。除了其它用于核实软件需求的分析和文件记录，建议用一个正式设计评审来确定在广泛的软件设计尝试开始前，详细地说明需求及其适当性。需求可被批准并逐步放行，但要注意对多个软件（和硬件）需求间的交互接口进行适当审核、分析和控制。

Typical Tasks – Requirements 典型任务-需求

Preliminary Risk Analysis 初步任务分析

Traceability Analysis –Software Requirements to System Requirements (and vice versa) –Software Requirements to Risk Analysis
可追溯分析-系统需求到软件需求（软件需求到系统需求）-软件需求到风险分析

Description of User Characteristics 用户特征的描述

Listing of Characteristics and Limitations of Primary and Secondary Memory
首要和辅助记忆体的局限性和特征列表

Software Requirements Evaluation 软件需求评估

Software User Interface Requirements Analysis 软件用户接口需求分析

System Test Plan Generation 系统测试计划的生成

Acceptance Test Plan Generation 验收测试计划的生成

Ambiguity Review or Analysis 含糊的审核或分析

5.2.3. Design 设计

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. Due to complexity of the project or to enable persons with varying levels of technical responsibilities to clearly understand design information, the design specification may contain both a high level summary of the design and detailed design information. The completed software design specification constrains the programmer/coder to stay within the intent of the agreed upon requirements and design. A complete software design specification will relieve the programmer from the need to make ad hoc design decisions.

在设计过程中，软件需求规约可转换为实施软件的一个逻辑的和物理图形的表示。软件设计规约描述了软件做什么和怎样做。由于项目的复杂性或责任人对清晰理解设计信息的技术水平的差异，这个设计规约可能应包括一个高水平的设计总结和详尽的设计信息。完整的软件设计规约把程序员/编码员限制在已定的需求和设计的框架中。一个完整软件设计规约使程序员不需要制定特别设计方案。

The software design needs to address human factors. Use error caused by designs that are either overly complex or contrary to users' intuitive expectations for operation is one of the most persistent and critical problems encountered by FDA. Frequently, the design of the software is a factor in such use errors. Human factors engineering should be woven into the entire design and development process, including 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.

软件设计需要说明人的因素。使用设计产生的错误，错误或来自过度复杂或违反用户的预期操作，是FDA遇到的最顽固和危险的问题之一。经常，软件设计是这些使用错误的一个因素。人因工程应贯穿整个设计和开发过程，包括器械设计需求、分析和测试。当开发流程图、状态转换图、样机研究工具和测试计划时，也应考虑到器械安全和可用性问题。另外，应进行任务和功能分析、风险分析、样机测试和评审及功能性试验。当应用这些方法时，应包括使用者人群的参与。

The software design specification should include:

软件设计分析应包括：

Software requirements specification, including predetermined criteria for acceptance of the software;

软件需求规约，包括软件验收标准的预定准则；

Software risk analysis; 软件风险分析；

Development procedures and coding guidelines (or other programming procedures);

开发过程和编程指南（或其他程序设计过程）

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;

系统文件编制（例如，一个叙述或环境图）描述了程序运行的系统的环境，包括硬件、软件和物理环境之间的关系；

Hardware to be used; 使用的硬件

Parameters to be measured or recorded; 测量和记录的参数；

Logical structure (including control logic) and logical processing steps (e.g., algorithms);

逻辑结构（包括逻辑控制单元）和逻辑过程步骤（例如，运算法则）；

Data structures and data flow diagrams; 数据结构和数据流程图；

Definitions of variables (control and data) and description of where they are used;

变量定义（控制和数据）和在哪使用的描述；

Error, alarm, and warning messages;

Supporting software (e.g., operating systems, drivers, other application software);
支持软件（例如，操作系统、驱动、其他应用程序软件）

Communication links (links among internal modules of the software, links with the supporting software, links with the hardware, and links with the user); 通信链接装置（软件内部模块链接，支持软件链接，硬件链接及用户链接）

Security measures (both physical and logical security); and 安全措施（物理和逻辑安全）及

Any additional constraints not identified in the above elements.
上述元素中确定的任何附加约束。

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. Written development procedures serve as a guide to the organization, and written programming procedures serve as a guide to individual programmers. As software cannot be validated without knowledge of the context in which it is intended to function, systems documentation is referenced. If some of the above elements are not included in the software, it may be helpful to future reviewers and maintainers of the software if that is clearly stated (e.g., There are no error messages in this program).

上面提到的前四个元素通常是独立的已存在文件，包含在软件设计分析的引用中。之前讨论的软件需求分析也是软件风险分析。书面开发规程可作为组织团体的一个指导，并且书面程序规程也可作为单个程序员的一个指导。当没有预期功能的背景知识，软件不能被验证时，可参考系统文件。若上面提到的几个元素不包括在软件中，这可能对未来审核人员和软件的维护人员是有帮助的，如果被清楚地规定（例如在这个程序里不存在错误信息）

The activities that occur during software design have several purposes. Software design evaluations are conducted to determine if the design is complete, correct, consistent, unambiguous, feasible, and maintainable. Appropriate consideration of software architecture (e.g., modular structure) during design can reduce the magnitude of future validation efforts when software changes are needed. Software design evaluations may include analyses of control flow, data flow, complexity, timing, sizing, memory allocation, criticality analysis, and many other aspects of the design. 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. The software risk analysis should be re-examined to determine whether any additional hazards have been identified and whether any new hazards have been introduced by the design.

在软件设计期间发生的活动基于几个目的。执行软件设计评估以确定设计是否完整、正确、一致、不含糊、可行和可维护。当需要软件变更时，设计期间对软件架构的适当考虑可减少未来验证尝试的规模幅度。软件设计评估可包括控制流程分析、数据分析、复杂性、时序、规模估计、内存分配、危害性分析及设计其他方面。应进行一个追溯性分析以证实软件设计实现了所有软件需求。作为鉴别需求不够充分的一个技术，追溯分析也应证实设计的方方面面可追溯软件需求。应对通信链接进行分析以评估关于硬件、用户和关联的软件需求相关的提出的设计。软件风险分析应被再检验以确定是否所有附加危害已被识别，是否所有新的危害被引入到设计中。

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. Portions of the design can be approved and released incrementally for implementation; but care should be taken that interactions and communication links among various elements are properly reviewed, analyzed, and controlled.

在软件设计活动结束时，在实施设计前，应进行一个正式设计评审以证实设计是正确、一致、完整、准确和可检测的。部分设计可经批准然后逐渐放行实施；但是应注意恰当地评审、分析和控制不同元素间的交互通信链接。

Most software development models will be iterative. This is likely to result in several versions of both the

software requirement specification and the software design specification. All approved versions should be archived and controlled in accordance with established configuration management procedures.
大多数的软件开发模型是迭代式的。这将产生软件需求规约和软件设计分析的几个版本。
所有批准版本应按照已建立的配置管理规程进行存档和控制。

Typical Tasks – Design 典型任务-设计

Updated Software Risk Analysis 更新的软件风险分析
Traceability Analysis - Design Specification to Software Requirements (and vice versa)
可追溯分析-统需求到软件需求（软件需求到系统需求）
Software Design Evaluation 软件设计评估
Design Communication Link Analysis 设计交互链接分析
Module Test Plan Generation 模型测试计划的生成
Integration Test Plan Generation 集成测试的生成
Test Design Generation (module, integration, system, and acceptance)
测试设计生成（模型、集成、系统和验收）

5.2.4. Construction or Coding

构建或编码

Software may be constructed either by coding (i.e., programming) or by assembling together previously coded software components (e.g., from code libraries, off-the-shelf software, etc.) for use in a new application. Coding is the software activity where the detailed design specification is implemented as source code. Coding is the lowest level of abstraction for the software development process. It is the last stage in decomposition of the software requirements where module specifications are translated into a programming language.

软件的构建或通过编码（即编程）或通过一起装配一个新应用程序使用的先前编码的软件构件（如，来自编码图书馆，成品软件等）来完成。编码是详细的设计分析作为源代码实施的软件活动。编码是软件开发过程最低级的抽象 。它是软件需求分解的最后阶段，模块分析被转换成一个程序语言。

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. The source code may be either compiled or interpreted for use on a target hardware platform. Decisions on the selection of programming languages and software build tools (assemblers, linkers, and compilers) should include consideration of the impact on subsequent quality evaluation tasks (e.g., availability of debugging and testing tools for the chosen language). Some compilers offer optional levels and commands for error checking to assist in debugging the code. Different levels of error checking may be used throughout the coding process, and warnings or other messages from the compiler may or may not be recorded. However, at the end of the coding and debugging process, the most rigorous level of error checking is normally used to document what compilation errors still remain in the software. If the most rigorous level of error checking is not used for final translation of the source code, then justification for use of the less rigorous translation error checking should be documented. Also, for the final compilation, there should be documentation of the compilation process and its outcome, including any warnings or other messages from the compiler and their resolution, or justification for the decision to leave issues unresolved.

编码通常包括一种高级编程语言，但也可能针对时序要求严格的操作使用汇编语言（或微码/微程序）。源代码或被编译或被解释在一个目标硬件平台上的使用。确定选择程序设计语言和软件构建工具（组译器、链接器和编译器）应包括对随后的质量评估任务（如调试器的可用性和选定语言的测试工具）产生的影响的考虑。

某些编译器可为错误校验提供可选级别和指令，以协助调试编码。编码的整个过程可使用错误校验的不同级别，而且来自编译器的警告或其他信息可能或不可能被记录。然而，在编码和调试过程结束时，最严格级别的错误检查通常用来记录仍存在软件中的编译错误

如果这个最严格级别的错误检查并没有用于源代码的最终转化，那么应记录使用不太严格的错误检查的理由。同样，对于最终编译，应有记录编译过程及其结果的文件，包括来自编译器及其分辨率的所有警告或其他信息，或离开未解决问题的决定的理由。

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. Code comments should provide useful and descriptive information for a module, including expected inputs and outputs, variables referenced, expected data types, and operations to be performed. Source code should also be evaluated to verify its compliance with the corresponding detailed design specification. Modules ready for integration and test should have documentation of compliance with coding guidelines and any other applicable quality policies and procedures.

公司经常采用特殊编码指南建立与软件编码过程相关的质量方针和规程。源代码应接受评估以证实与指定编码指南的符合性。这些指南应包括编码约定，如清楚明白、风格、复杂性管理和评论。编码评论应为一个模块提供有用的和可描述的信息，如预期的输入和输出，参考的变量，预期的数据类型及执行的操作。源代码也应经评估以核实符合相应详细的设计分析规范。已准备集成和测试的模块应有符合编码指南和所有其他可应用的质量方针和规程的文件。

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. They allow for examination of each error in isolation and can also help in focusing later dynamic testing of the software. Firms may use manual (desk) checking with appropriate controls to ensure consistency and independence. Source code evaluations should be extended to verification of internal linkages between modules and layers (horizontal and vertical interfaces), and compliance with their design specifications. Documentation of the procedures used and the results of source code evaluations should be maintained as part of design verification.

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:

源代码评价经常实现为代码审查和代码走查。这些静态分析在代码执行前提供了一个非常有效的方式探测错误。它们在封闭状态下检查每个错误，也可以为后面的软件动态分析提供帮助。公司可使用带有适当控制的桌面检查，以确保代码的设计是一致和独立的。源代码评价应延伸到核实内部模块和图层集合的链接（水平和垂直接口），以及与设计分析的符合性。使用的规程文件 和源代码评估的结果应作为设计核实的一部分进行维护。一个源代码追踪分析是一个重要的工具，可核实链接到已建立的规范说明和测试规程的所有代码。应执行兵记录一个源代码追踪分析，以证实：

Each element of the software design specification has been implemented in code;

软件设计规约的每个元素已用代码实施；

Modules and functions implemented in code can be traced back to an element in the software design specification and to the risk analysis;

模块和代码实施的功能可被追溯到软件设计规约的一个元素，以及风险分析；

Tests for modules and functions can be traced back to an element in the software design specification and to the risk analysis; and

模块和功能的测试可追溯到软件设计规约的一个元素，以及风险分析；以及

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

模块和功能的测试可追溯到相同模块和功能的源代码。

Typical Tasks – Construction or Coding 典型任务-构建或编码

Traceability Analyses –Source Code to Design Specification (and vice versa) –Test Cases to Source Code and to Design Specification

追溯分析-源代码到设计规约（反之亦然）-源代码和设计规约的测试用例

Source Code and Source Code Documentation Evaluation

源代码和源代码文件评估

Source Code Interface Analysis 源代码接口分析

Test Procedure and Test Case Generation (module, integration, system, and acceptance)

测试程序和测试用例生成（模块、集成、系统和验收）

5.2.5. Testing by the Software Developer 软件开发者执行的测试

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.

软件测试是在已知条件下运行软件产品，并有确定的输入和的与预先定义的预期结果相比较的记录输出结果。它是一项耗时、困难和存在瑕疵的活动。就其本身而言，它需要较早计划以实现有效和高效。

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. The magnitude of effort to be applied throughout the testing process can be linked to complexity, criticality, reliability, and/or safety issues (e.g., requiring functions or modules that produce critical outcomes to be challenged with intensive testing of their fault tolerance features). Descriptions of categories of software and software testing effort appear in the literature, for example:

在软件开发阶段， 应及早制定测试计划和测试用例是可行的。它们应确定计划、运行环境、资源（人员、工具等）、方法、用例（输入、程序、输出、预期结果）、文件和报告限度。整个测试阶段进行尝试的量级可与复杂性、临界程度、可靠性和/或安全问题（例如，产生关键结果的需求功能或模块，它们受到增强容错功能技术的挑战）联系起来。文献中有关于软件和软件测试尝试的分类的描述，例如：

NIST Special Publication 500-235, *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*;

美国国家标准技术研究所，结构化测试：圈复杂度测试方法；

NUREG/CR-6293, *Verification and Validation Guidelines for High Integrity Systems*;

高完善系统的验证和确认；

IEEE Computer Society Press, *Handbook of Software Reliability Engineering*.

电气与电子工程协会，软件可靠性手册。

Software test plans should identify the particular tasks to be conducted at each stage of development and include justification of the level of effort represented by their corresponding completion criteria.

软件测试计划应确定每个开发阶段执行的特别任务，应包括与任务相应完成标准相适应的尝试水平的判断。

Software testing has limitations that must be recognized and considered when planning the testing of a particular software product. Except for the simplest of programs, software cannot be exhaustively tested. Generally it is not feasible to test a software product with all possible inputs, nor is it possible to test all possible data processing paths that can occur during program execution. There is no one type of testing or testing methodology that can ensure a particular software product has been thoroughly tested. 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.

当计划测试一个特别软件产品时，必须意识和考虑到软件测试的局限性。除了最简单的程序，软件不能被严格地测试。通常以所有可能输入测试一个软件产品是不可行的，同样，也是不可能测试程序运行时产生的所有可能数据处理路径。没有一个测试或测试方法的类型可以保证一个特别软件产品被彻底地测试。所有程序功能的测试并不是所有已测试的程序。一个程序的所有代码不是指所有必要功能均存在程序中。所有程序功能和所有程序代码的测试并不意味着这个程序百分之百正确。未发现错误的软件测试，不应认为没有错误存在；或许可认为测试是浅显的。

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 specification document must identify what, when, how, why, etc., is to be achieved with an engineering (i.e., measurable or objectively verifiable) level of detail in order for it to be confirmed through testing. The real effort of effective software testing lies in the definition of what is to be tested rather than in the performance of the test.

软件测试用例的一个本质元素是预期结果。允许对实际测试结果进行客观评价是重要的细节。从相应、预定的定义或标准获得必需测试信息。一个软件标准文件必须用通过借助测试进行确认的设计细节标准（即可测量或客观地证实的）确定将要完成什么，何时，怎样做，为什么等。 有效软件测试的真实尝试在于定义测试什么，而不在于测试的执行。

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

一个软件测试过程应以培养软件产品的有效检查为原则。适用的软件测试原则包括：

The expected test outcome is predefined;

预先确定预期测试输出结果；

A good test case has a high probability of exposing an error;

具有暴露错误可能性的良好测试用例；

A successful test is one that finds an error;

发现一个错误的一个成功的测试；

There is independence from coding;

有独立于代码的自主性；

Both application (user) and software (programming) expertise are employed;

雇用应用程序（用户）和软件（程序设计）专家

Testers use different tools from coders;

测试人员使用不同于编码器的工具

Examining only the usual case is insufficient;

仅有检查通用用例是不充足的；

Test documentation permits its reuse and an independent confirmation of the pass/fail status of a test outcome during subsequent review.

测试文件记录允许其本身再利用及对一个随后评审过程中的一个测试结果的通过/失败状态的独立确认。

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. There may be a distinct integration level of testing. A software product should be challenged with test cases based on its internal structure and with test cases based on its external specification. These tests should provide a thorough and rigorous examination of the software product's compliance with its functional, performance, and interface definitions and requirements.

一旦成功完成首要任务（例如，代码走查），即可启动软件测试。从单元级测试开始，以系统级测试结束。或许有一个独特的集成度测试。一个软件产品应根据其内部结构挑战测试用例，并根据其内部标准挑战测试用例。这些测试应提供一个周密和精确的检查，以判断软件产品是否符合其功能、性能和接口的定义和要求。

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. These test cases challenge the control decisions made by the program; and the program's data structures including configuration tables. 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.

基于代码的测试被认为是结构测试或“白箱/盒” 测试。它根据从源代码、细节设计标准和其他开发文件中获得的知识确定测试用例。这些测试用例挑战程序做出的控制决策；及包括配置表的程序的数据结构。结构测试可识别程序运行时从未执行的“死”/无作用代码。结构化测试最初借助单元（模块）

级测试来完成，但可延伸到软件测试的其他级。

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. For example, if a testing program has achieved “statement coverage,” it means that 100% of the statements in the software have been executed at least once. Common structural coverage metrics include:

结构化测试等级可使用度量进行评估，设计度量来证明在结构测试期间软件结构已被评估的百分率是多少。这些度量通常称作“覆盖”和一个关于测试挑选标准的完成估量。结构覆盖的量应与软件引发的风险级相匹配。“覆盖”这个术语的使用通常是指100%覆盖。例如，如果一个测试程序已经完成“语句覆盖”，那么软件中100%的语句至少执行一次。一般结构覆盖度量包括：

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.

语句覆盖-要求有足够的保证每个可执行语句至少被执行一次的测试用例；然而，此覆盖的完成不足以为软件产品的行为提供信任。

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.

分支覆盖/判定覆盖-要求为每个可执行程序判定或分支提供有足够的测试用例，以便每个可能的结果至少出现一次。对于多数软件产品，它被认为是一个最小的覆盖级别，但单独的判定覆盖对于高集成应用程序是不充分的。

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.

条件覆盖-要求为一个针对所有可能结果至少执行一次程序判定的每个条件提供足够的测试用例。仅当多重条件必须经评估做出决定时，它不同于分支覆盖。

Multi-Condition Coverage – This criteria requires sufficient test cases to exercise all possible combinations of conditions in a program decision.

多条件覆盖—针对可执行程序所有可能组合的条件提供足够的测试用例。

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.

循环覆盖-要求为所有可执行程序环路执行0次，1次，2次循环提供足够的测试用例，并包含了许多迭代覆盖初始化、典型运行和终止（界限）条件。

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.

路径覆盖-为每个可行的路径、基础路径等提供足够的测试用例，从一个定义的程序段（函数）的入口开始至少执行一次。因为有大量的通过一个软件程序的可能路径，路径覆盖通常不能完成。路径覆盖的数量通常根据测试软件的风险或紧要程度确定。

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.

数据流覆盖-要求为每个可行的数据流至少执行一次所有的足够测试用例。有许多可用的数据流测试策略。

Definition-based or specification-based testing is also known as functional testing or "black-box" testing. It identifies test cases based on the definition of what the software product (whether it be a unit (module) or a complete program) is intended to do. These test cases challenge the intended use or functionality of a program, and the program's internal and external interfaces. Functional testing can be applied at all levels of software testing, from unit to system level testing.

基于判定的或标准的测试也被认作功能测试或“黑箱测试”。根据什么是软件产品的预期用途的判定确定测试用例。这些测试用例挑战一个程序的预期用途或功能，以及程序的内部和外部接口。功能测试可用于软件测试的所有级，从单元到系统级测试。

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

下面功能软件测试的类型通常包括越来愈多的尝试等级：

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.

常规用例测试-以常规输入进行的测试是必要的。然而，对于仅以预期结果进行的软件产品测试，有效性输入并不能彻底测试这个软件产品。单独借助常规用例测试，不能为软件产品的可靠性提供足够的保证。

Output Forcing – Choosing test inputs to ensure that selected (or all) software outputs are generated by testing.

强制输出-选择测试输入以确保测试产生选定（或全部）的软件输出。

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.

稳健性-当出现意想不到结果和无效输入时，软件测试应证明一个软件产品可正确地运行。识别一套这样的测试用例是否充分（即覆盖更加全面）的方法（黑盒测试）包括等价类划分、边界值分区和特殊用例鉴别（错误猜测）。这些技术并不能保证测试能识别出一个软件产品面对的所有的最适当的挑战。

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.

组合测试-这个识别功能测试方法，最重要的是强调个别或单一的测试输入。根据使用条件，多数软件产品使用多重输入方式运行。周密的软件测试应考虑一个软件单元或系统运行期间可能遇见的多个输入的组合。错误猜测可扩展到识别多个输入的组合，但是这是一项特别的技术。因果图分析是一项功能软件测试技术，它可以系统地识别一个软件产品的测试用例所包含的输入组合。

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.

功能和结构软件测试用例识别技术为测试提供了特定输入，而不是随机测试输入。这些技术的一个缺陷是很难将结构和功能测试的完成标准与软件产品的可靠性联系起来。统计测试使用的测试数据（测试用例）可依据一个软件运行剖面生成的确定的分布中随意产生（例如软件产品的预期使用、冒险使

用或恶意使用）。假如出现一个更高的几率，即鉴别既不是软件产品设计者也不是测试者预期的单独和多重罕见的运行条件，可产生大量的测试数据并可覆盖特别区域或关注点。统计测试也提供更高的结构覆盖。其并不需要一个稳固的软件产品。因此，结构和功能测试是一个软件产品统计测试的先决条件。

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. Not only should it demonstrate that the change was implemented correctly, testing should also demonstrate that the change did not adversely impact other parts of the software product. Regression analysis and testing are employed to provide assurance that a change has not created problems elsewhere in the software product. Regression analysis is the determination of the impact of a change based on review of the relevant documentation (e.g., software requirements specification, software design specification, source code, test plans, test cases, test scripts, etc.) 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.

软件测试的其他方面是软件变更的测试。变更经常出现在软件开发期间。这些变更发生的情况如1）发现错误后的调试以排除故障，并改正；2）新的或改变的要求（“需求蠕变”）；3）当发现更加有效或高效的软件实施情况，修改设计。对于一个基线软件产品（认可的），产品的任何变更应有其自身的“迷你型生命周期”，包括测试。一个改变的软件测试要求额外的尝试。测试不仅应证明变更已被正确实施，也应证实变更并不会对软件产品的其他方面带来不良影响。借助回归分析和测试保证变更并不会在软件产品的其他地方产生问题。为鉴别即将进行的回归测试的必要性，回归分析应基于对相关文件（例如，软件需求规约、软件设计标准、源代码、测试计划、测试用例、测试脚本等）的评审来确定变更带来的影响。回归测试可再次运行测试用例，判断一个程序是否被正确执行，并将先前的与现在的结果进行比较以检测一个软件变更产生的非计划性的影响。当使用集成方法构建一个软件产品时，也应采用回归分析和回归测试以确保最新集成的模块并不会对先前集成的模块的处理带来不良影响。

In order to provide a thorough and rigorous examination of a software product, development testing is typically organized into levels. As an example, a software product's testing can be organized into unit, integration, and system levels of testing.

为了提供一个彻底和严格的软件产品检查，开发测试通常按级别进行组织。例如，一个软件产品的测试可被组织成单元、集成和系统测试级别。

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.

   单元（模块或组件）级别测试集中于子程序功能的早期检查，以确保在系统级别上不明显的功能可被检查出来。单元测试确保质量软件单元被装配集合成最终的软件产品。

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

   集成级别测试集中于数据的转移和程序内外接口的控制。外部接口是指其他软件（包括操作系统软件）、系统硬件及用户，可描述为通讯链路。

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):
系统级别测试证明所有规定的功能均存在，软件产品是制得信赖的。这个级别的测试可证实在特殊操作平台表现出来的与软件产品需求相关的内建软件的功能和性能，系统级别测试可说明功能关注点和与预期用途相关的器械软件的元素 ：

Performance issues (e.g., response times, reliability measurements);
性能问题（例如，响应时间，可靠性测定）;
Responses to stress conditions, e.g., behavior under maximum load, continuous use;
应力状态反应，如，最大负载条件下的行为，持续使用；
Operation of internal and external security features;
内外部安全功能的运行；
Effectiveness of recovery procedures, including disaster recovery;
恢复过程的有效性，包括灾难复原；
Usability; 可用性；
Compatibility with other software products; 与其他软件产品的兼容性；
Behavior in each of the defined hardware configurations; and 每个确定的硬件配置的 行为；及
Accuracy of documentation.
文件的准确度。

Control measures (e.g., a traceability analysis) should be used to ensure that the intended coverage is achieved.
应通过控制措施（例如，一个可追溯分析）确保完成预期覆盖范围。

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. Also, for a software product that is a medical device or a component of a medical device that is to be used on humans prior to FDA clearance, testing involving human subjects may require an Investigational Device Exemption (IDE) or Institutional Review Board (IRB) approval.
系统级别测试也展现了软件产品的预期操作环境行为。这种测试的定位依赖软件开发者生成目标运行环境的能力。可能会依赖在（潜在）客户位置上的环境、模拟和/或测试。当现场执行软件开发人员不能直接控制的计划的系统级别测试时，测试计划应鉴别需要的控制，以确保实现预期覆盖，制定适当的文件。同样，对于一个软件产品，它是一个医疗器械或是一个人用医疗器械的部件，在获得FDA的安检证明前，包含受试对象的测试可能需要一个调研性器械豁免或伦理审查委员会的批准。

Test procedures, test data, and test results should be documented in a manner permitting objective pass/fail decisions to be reached. They should also be suitable for review and objective decision making subsequent to running the test, and they 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.
测试程序、测试数据及测试结果应以目标完成/失败的结论方式来证明。他们应适合评审和在运行测试后做出目标决定，也应适合在任何后续回归测试中使用。测试过程中检测到的错误在软件放行之前应被记录、分级、评审和解决。可根据开发生命周期收集和分析到的软件错误数据确定软件产品上市放行的适宜性。测试报告应符合相应测试计划的要求。

Software products that perform useful functions in medical devices or their production are often complex. 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. These tools may include supporting software built in-house to facilitate unit (module) testing and subsequent integration testing (e.g., drivers and stubs) as well as commercial software testing tools. Such tools should have a degree of quality no less than the software product they are used to develop. Appropriate documentation providing evidence of the validation of these software tools for their intended use should be maintained (see section 6 of this guidance).

执行有效医疗器械或其生产功能的软件产品通常是复杂的。经常用软件测试工具来确保这样的软件产品测试的一致性、彻底性和效能。这些工具可包括支持软件内置以便于单元（模块）测试和后续集成测试（例如，驱动和短线），还有商业软件测试工具。这些工具应具有一定程度的质量特质，程度应不少于用于开发的软件产品。应维护提供这些软件测试工具预期用途验证的适当的证明文件（参见本指导第6部分）。

Typical Tasks – Testing by the Software Developer

通常任务-软件研发人员执行的测试

Test Planning 测试计划

Structural Test Case Identification 结构测试用例鉴别

Functional Test Case Identification 功能测试用例鉴别

Traceability Analysis - Testing –Unit (Module) Tests to Detailed Design –Integration Tests to High Level Design –System Tests to Software Requirements

可追溯性分析-测试-详细设计单元（模块）测试-高级别设计集成测试-软件需求系统 测试

Unit (Module) Test Execution 执行单元（模块）测试

Integration Test Execution 执行集成测试

Functional Test Execution执行功能测试

System Test Execution 执行系统测试

Acceptance Test Execution 执行验收测试

Test Results Evaluation 测试结果评估

Error Evaluation/Resolution 错误评估/解决

Final Test Report 最终测试报告

5.2.6. User Site Testing 用户现场测试

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.)

用户现场测试是软件验证的主要部分。质量系统法规要求安装和检查规程（包括检测，在合适的情况），以及检查文件和测试证明是否安装适当（见21C FR §820.170.）。同样地，生产设备必须满足指定需求，并且自动设备必须被验证以符合预期用途（见21 CFR §820.70(g) and 21 CFR §820.70(i)）。

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 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.

用户现场测试的相关术语可能令人困惑。术语，如beta测试、现场安装、用户验收测试、安装确认和安装检测都可用来描述用户现场测试。就本指导的目的而言，“用户现场测试”这个名词包含所有这些及在开发人员的受控环境之外发生的其他检测。用户现场测试应是一个带有真实硬件和软件的用户现场，而且这个现场将是这个安装系统配置的一部分。用户现场测试或通过实际使用或通过模拟使用

软件来完成，软件将在预期功能范畴内通过测试。

Guidance contained here is general in nature and is applicable to any user site testing. However, in some areas (e.g., blood establishment systems) there may be specific site validation issues that need to be considered in the planning of user site testing. 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.

这里的指导文件本质上是通用的，并且适合任何用户现场测试。然而，在计划用户现场测试中，某些领域（如，血液机构系统） 可能有特殊现场验证问题需要考虑。测试计划人员应与ＦＤＡ下属的所有相应产品管辖中心联系，以确定是否有其他关于用户现场测试的额外法规要求。

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.

用户现场测试应遵循一个预定的书面计划，计划带有一份正式的测试总结和一份正式验收的报告。应保留所有测试程序、测试输入数据和测试结果的文件记录。

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.

应有证据证明按照说明安装和配置了硬件和软件。采取措施确保测试期间运用了所有的系统组件，并确保这些组件的版本是指定的。测试计划应是详细说明整个运行条件全程的测试，也应详细说明延续情况，即容许系统与一个广泛的条件和事件邂逅的足够的时间，以试图探测到在更为正常活动期间出现的不明显的任何后续错误。

Some of the evaluations 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. The developer may be able to furnish the user with some of the test data sets to be used for this purpose.

在开发地点经软件开发人员早先执行的一些评估应在实际使用时再次进行。这些可包括对大量数据、重负载或压力、安全、错误测试（规避、探测、容错和恢复）、错误信息及安全要求实施。研发人员可以提供给使用者一些针对目的使用的测试数据集。

In addition to an evaluation of the system's ability to properly perform its intended functions, 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.

During user site testing, 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.

除了评估系统恰当执行其预期功能的能力，也应评估用户理解系统及正确与系统连接的能力。操作员应能执行预期功能，并以适当及时的方式对所有警报、警告和错误信息做出回应。在现场用户测试期间，应记录和维护已出现的适当系统性能和任何系统故障。应有系统修正来弥补测试期间探测到的故障，针对任何其他软件变更，系统应遵循相同程序和控制。

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.

软件开发人员可能参与，也可能不参与用户现场测试。如果开发人员参与，他们可一直坚守到现场测

试的最后部分，即设计层次系统测试。如果开发人员未参与，更加重要的是，应使所有人理解仔细测试计划的重要性，预期的测试结果的确定及所有测试输出结果的记录。

Typical Tasks – User Site Testing 经典任务-用户现场测试

Acceptance Test Execution 验收测试执行
Test Results Evaluation 测试结果评估
Error Evaluation/Resolution 错误评估/解决
Final Test Report终测试报告

5.2.7. Maintenance and Software Changes 维护和软件变更

As applied to software, the term maintenance does not mean the same as when applied to hardware. The operational maintenance of hardware and software are different because their failure/error mechanisms are different. 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.

当应用软件时，维护这个词并不意味着同硬件的维护一样。硬件和软件的操作性维护是不同的，因为他们故障／错误机制不相同。硬件维护典型包括预防性硬件维护行动、组件更换和改正变化。软件维护包括纠正、完善、适应性维护，但不包括预防性维护行动或软件组件的更换。

Changes made to correct errors and faults in the software are corrective maintenance. Changes made to the software to improve the performance, maintainability, or other attributes of the software system are perfective maintenance. Software changes to make the software system usable in a changed environment are adaptive maintenance.

纠正软件错误和故障进行的变更是改进维护。改善软件系统的性能、可维护性或其他属性的变更属于预防性维护。对在一个变更的环境中可用的软件系统进行的软件变更是适应性维护。

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).

当对一个软件系统进行变更时，或者在初始开发期间，或者在后放行维护期间，应进行充分的回归分析和测试以证明变更未涉及的软件部分没受到不良影响。这个测试不包括评估实施变更的正确性。

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. Careful and complete documentation of the design structure and interrelationships of various modules, interfaces, etc., can limit the validation effort needed when a change is made. The level of effort needed to fully validate a change is also dependent upon the degree to which validation of the original software was documented and archived. For example, test documentation, test cases, and results of previous verification and validation testing need to be archived if they are to be available for performing subsequent regression testing. Failure to archive this information for later use can significantly increase the level of effort and expense of revalidating the software after a change is made.

每个软件变更所必需的特殊验证尝试由变更的类型、受影响的开发产品和那些产品对软件操作的影响决定。当发生变更时，各种模块、接口的相互关系和设计结构的仔细完整的文件等限制了必需的验证尝试。需要进行完整验证一个变更的尝试程度也取决于原先软件验证是否有文件记录及存档。例如，测试文件、测试用例及需要存档的先前验证与确认的结果，如果它们对随后进行的回归分析有用。没有存档这个资料备用，可能严重增加一个变更后的进行尝试的程度和再验证软件的花费。

In addition to software verification and validation tasks that are part of the standard software development process, the following additional maintenance tasks should be addressed:

除了软件验证和确认工作外，其是标准软件开发过程的组成部分，下列其他维护任务也应说明：

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.
软件验证计划修订-对于先前验证过的软件，这个目前的软件验证计划应被修订以支持这个修订的软件验证。若没有先前的软件验证计划，应建立一个这样的计划以支持这个修订的软件验证。

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.)

异常评估-软件组织定期维护文件记录，如描述软件异常发现的软件问题报告和确定每个异常采取的特殊行动。然而，太频繁的错误被重复，因为软件开发人员并没有采取下个步骤以确定问题的根源和对需要避免再次发生问题的过程和规程进行变更。软件异常问题的评估应根据它们的严重性及对系统操作和安全的影响，但是它们也应作为质量系统过程缺陷症侯来对待。一个对异常的根源分析能鉴别特殊的质量系统缺陷。进行趋势识别（例如相同软件缺陷的再次出现），应执行并记录适当的CAPA以避免相同软件缺陷的再次出现（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.

问题鉴别和解决跟踪-所有软件维护期间发现的问题应被记录。每个问题的解决应被跟踪，以保证从历史参考和趋势中看出其是被修正的。

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.

建议的变更评估-所有建议的修改、提高或其他应被评估，以确定对系统产生影响的每个变更。这个信息应确定验证和/或确认需要重复的程度。

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.

任务迭代-对于批准的软件变更，应进行所有必须的验证和缺认工作，以确保正确执行计划的变更，所有文件是完整的和更新的，没有不可验收的变更出现在软件执行中。

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.

文件更新-应仔细评审文件以确定受变更影响的文件。所有受影响的批准的文件（例如，标准、测试程序、用户手册等）均应按照配置管理程序进行更新。在任何进行任何维护和软件变更前应更新标准。

SECTION 6. VALIDATION OF AUTOMATED PROCESS EQUIPMENT AND QUALITY SYSTEM SOFTWARE 自动化过程验证设备和质量系统软件

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.

质量系统法规要求“当计算机或自动数据处理系统用作生产或质量系统的组成部分时，器械生产商应按照已建立的方案验证计算机软件的预期用途”（见21 CFR §820.70(i)））。这是自1978年以来，FDA的医疗器械良好生产规范法规的监管要求。

In addition to the above validation requirement, 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.

除了上述验证要求，实现器械生产商的生产工序或质量系统的一部分的计算机系统（或按照FDA其他法规的要求用于建立和维护记录的计算机系统）应用电子记录（电子签名法规）(见 21 CFR Part 11.)；这个法规建立了在创建或维护电子记录时额外的安全、数据完整性及验证需求。 对于任何自动化记录保持系统，这些额外的第11部分要求应被仔细考虑并涵盖在系统需求和软件需求中。系统验证和软件验证应证明已满足所有的第11部分要求。

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:

计算机和自动设备应被广泛应用在以下所有方面，如医疗器械设计、实验室检验和分析、产品检查和验收、生产和过程控制、环境控制 、包装、标签、可追溯性、文件控制、投诉管理及质量系统的其他方面。自动厂房楼面运行操作越来越多地包含了嵌入系统在下列方面的广泛应用：

programmable logic controllers; 可编程逻辑控制器
digital function controllers; 数据功能控制器
statistical process control; 统计过程控制
supervisory control and data acquisition; 监督控制和数据采集
robotics; 机器人
human-machine interfaces; 人机接口
input/output devices; and 输入/输出器械
computer operating systems. 计算机操作系统

Software tools are frequently used to design, build, and test the software that goes into an automated medical device. Many other commercial software applications, such as word processors, spreadsheets, databases, and flowcharting software are used to implement the quality system. All of these applications are subject to the requirement for software validation, but the validation approach used for each application can vary widely.

软件工具通常用来设计、构造和检测医疗器械软件。其他许多商业软件应用程序常用于质量管理，如文字处理器、电子表格、数据库及流程图绘制软件。所有这些应用程序均需要进行验证，但每个软件的验证方法可能会有很大不同。

Whether production or quality system software is developed in-house by the device manufacturer, developed by a contractor, or purchased off-the-shelf, it should be developed using the basic principles outlined elsewhere in this guidance. The device manufacturer has latitude and flexibility in defining how validation of that software will be accomplished, but validation should be a key consideration in deciding how and by whom the software will be developed or from whom it will be purchased. The software developer defines a life cycle model. Validation is typically supported by:

不管生产或质量系统软件是由器械生产商内部开发，合同生产商开发或者购买成品软件开发，均应按照本指导概述的基本原则进行。器械生产商在确定软件验证如何完成上存在自主性和灵活性，但是软件验证的一个关键的考虑要素是在确定软件如何并将由谁开发，或从哪购买。软件开发者确定一个生

命周期模型。验证通常需要：

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.

核对软件开发生命周期的每个阶段的输出；和检查成品软件在器械生产商预期使用环境中的正常运行。

6.1. HOW MUCH VALIDATION EVIDENCE IS NEEDED? 需要多少验证依据

The level of validation effort should be commensurate with the risk posed by the automated operation. In addition to risk other factors, such as the complexity of the process software and the degree to which the device manufacturer is dependent upon that automated process to produce a safe and effective device, determine the nature and extent of testing needed as part of the validation effort. 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. For example, an automated milling machine may require very little testing if the device manufacturer can show that the output of the operation is subsequently fully verified against the specification before release. On the other hand, extensive testing may be needed for:

验证尝试的程度应与自动运行引起的风险相称。除了其他风险因素，如处理软件过的复杂性和器械生产商对生产一个安全、有效器械的自动化流程所依赖的程度，确定了验证尝试所需的测试的性质和程度。自动化流程的写在文档中的要求和风险分析有助于明确显示验证软件预期使用的证据范围。例如，自动铣床可能需要微小的测试，如果器械生产商能证明在放行前运行输出结果可随后被证实符合标准。另一方面，大量的测试可能会需要用于：

a plant-wide electronic record and electronic signature system;

一个全厂范围的电子记录和电子签名系统；

an automated controller for a sterilization cycle; or

一个灭菌周期的自动控制器；或

automated test equipment used for inspection and acceptance of finished circuit boards in a life-sustaining / life-supporting device.

用于验收一个生命支持/维持器械使用的成品线路板的自动测试设备。

Numerous commercial software applications may be used as part of the quality system (e.g., a spreadsheet or statistical package used for quality system calculations, a graphics package used for trend analysis, or a commercial database used for recording device history records or for complaint management). The extent of validation evidence needed for such software depends on the device manufacturer's documented intended use of that software. For example, a device manufacturer who chooses not to use all the vendor-supplied capabilities of the software only needs to validate those functions that will be used and for which the device manufacturer is dependent upon the software results as part of production or the quality system. However, 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).)

众多的商业软件应用程序或许可作质量系统的一部分（例如 ，用于质量系统计算的一个电子表格或统计包，用于趋势分析的一个图形软件包，或用作记录历史记录或投诉管理的一个商务数据库） 。这些软件需要的验证依据范围取决于器械生产商记录的软件预期用途。例如，一个没有使用供应商提供的所有软件能力的器械生产商仅仅需要验证将要使用的功能，且器械生产商依赖于生产或质量系统的软件结果。然而，高风险应用程序不应在未经验证软件功能的这个相同的操作环境中运行，尽管那些软件功能并未使用。当高风险应用程序和低风险应用程序用于同一运行环境时，可能需要考虑风险消减技术，如记忆分割或其他资源保护的方法。当软件需要更新或需要进行变更时，器械生产商应考

虑这些变更可能会影响软件的“使用部分”，并必须对它们进行再验证。(见 21 CFR §820.70(i).)

6.2. DEFINED USER REQUIREMENTS 定义用户需求

A very important key to software validation is a documented user requirements specification that defines:
软件验证的一个非常重要方点是一个文件化的用户需求规约，其定义了：

the “intended use” of the software or automated equipment; and
软件或自动化设备的“预期用途”；和

the extent to which the device manufacturer is dependent upon that software or equipment for production of a quality medical device.
器械生产商对软件或用于生产一个合格的医疗器械设备的依赖程度。

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:
器械生产商（用户）需定义预期运行环境，包括所有要求的硬件和软件配置、软件版本、实用程序等。用户也需要：

document requirements for system performance, quality, error handling, startup, shutdown, security, etc.;
记录系统性能、质量、错误处理、启动、关机、安全保障等的需求；

identify any safety related functions or features, such as sensors, alarms, interlocks, logical processing steps, or command sequences; and
识别任何安全相关功能或性能，如传感器、警报器、互锁、逻辑过程步骤，或指令序列；和

define objective criteria for determining acceptable performance.
定义确定合格性能的客观标准。

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 pre-determined 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.
验证必须按照一个文件化的方案执行，并且验证结果也必须被记录(见 21 CFR §820.70(i).)。测试用例应被记录，可使系统在预先确定的标准下，挑战系统本身的性能，特别是系统本身最关键的参数。测试用例应该处理错误和警报情况，启动，关闭，所有适用的用户功能和操作员控制，潜在操作员错误，容许值最大和最小范围，以及对设备预期用途适用的应力状态。应执行这些测试用例并记录和评价结果，以确定结果是否支持这样一个结论-软件是基于其预期用途而验证的。

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:
一个器械生产商或许由其内部人员执行验证，或者由第三方执行，如设备/软件供应商或一个顾问。不管采取何种方式，器械生产商应有最终责任，以确保生产和质量系统软件：

is validated according to a written procedure for the particular intended use; and
will perform as intended in the chosen application.
按照一个规定了特殊预期用途的书面规程进行验证；并将在挑择的应用程序中执行。

The device manufacturer should have documentation including:
器械生产商应有如下文件：

defined user requirements; 确定的用户需求；
validation protocol used; 使用的验证方案；
acceptance criteria; 验收标准；
test cases and results; and 测试用例和结果；及

a validation summary that objectively confirms that the software is validated for its intended use. 一个客观地确定软件基于其预期用途进行验证的验证概述。

6.3. VALIDATION OF OFF-THE-SHELF SOFTWARE AND AUTOMATED EQUIPMENT 成品软件和自动设备的验证

Most of the automated equipment and systems used by device manufacturers are supplied by third-party 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. For OTS software and equipment, the device manufacturer may or may not have access to the vendor's software validation documentation. If the vendor can provide information about their system requirements, software requirements, validation process, and the results of their validation, the medical device manufacturer can use that information as a beginning point for their required validation documentation. The vendor's life cycle documentation, such as testing protocols and results, source code, design specification, and requirements specification, can be useful in establishing that the software has been validated. However, such documentation is frequently not available from commercial equipment vendors, or the vendor may refuse to share their proprietary information.

器械生产商使用的多数自动化设备和系统是由第三方供应商供应的，而且是已购买的成品 (OTS)。器械生产商有责任确保OTS软件者使用的产品开发方法对于器械生产商的OTS软件的预期用途是适当的和有效的。对于OTS软件和设备，器械生产商可能或者不可能接触到这个供应商的软件验证文件。如果这个供应商可以提供有关系统需求、软件需求、验证过程及这些验证结果的信息资料，医疗器械生产商可将此信息资料用作所需验证文件的一个起点。生产商的生命周期文件，如测试方案和结果、源代码，设计标准和需求标准，可有助于确定已被验证的软件。然而吗，这些文件并不能定期从设备生产商获得，或者说生产商可能拒绝提供他们的专有资料。

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.

如有可能，并根据器械存在的风险，器械生产商应考虑审计用于构建OTS软件的供应商的设计和开发方法，并应评估这个OTS软件产生的开发和验证文件。这些审计应由器械生产商或一个合格的第三方执行。审计应证明供应商的规程和执行OTS软件的验证和确认活动的结果是适当和充分的，足以保证使用这个软件生产的医疗器械的安全性和有效性。

Some vendors who are not accustomed to operating in a regulated environment may not have a documented life cycle process that can support the device manufacturer's validation requirement. Other vendors may not permit an audit. Where necessary validation information is not available from the vendor, the device manufacturer will need to perform sufficient system level "black box" testing to establish that the software meets their "user needs and intended uses." For many applications black box testing alone is not sufficient. Depending upon the risk of the device produced, the role of the OTS software in the process, the ability to audit the vendor, and the sufficiency of vendor-supplied information, the use of OTS software or equipment may or may not be appropriate, especially if there are suitable alternatives available. The device manufacturer should also consider the implications (if any) for continued maintenance and support of the OTS software should the vendor terminate their support.

一些不习惯在一个规定的环境中操作的供应商可能没有一个文件记录生命周期过程，以支持器械生产商的验证需求。其他供应商可能不允许审计。必要时，供应商提供的验证信息并不可用，器械生产商将需要执行充分的系统级 "黑箱测试"，来表明软件是符合"用户需要和预期用途的"。对于一些应用程序，单独进行黑箱测试并不充分。根据生产的器械风险，OTS软件的过程角色，审计供应商的能力和供应商信息的充分性，OTS软件或器械的使用可能是适当的也可能是不适当的，特别是如果有

适当的可用备选时。若供应商中止支持，器械生产商也应考虑对OTS软件进行持续的维护和支持（若有的话）。

For some off-the-shelf software development tools, such as software compilers, linkers, editors, and operating systems, exhaustive black-box testing by the device manufacturer may be impractical. Without such testing – a key element of the validation effort – it may not be possible to validate these software tools. However, their proper operation may be satisfactorily inferred by other means. For example, compilers are frequently certified by independent third-party testing, and commercial software products may have “bug lists”, system requirements and other operational information available from the vendor that can be compared to the device manufacturer’s intended use to help focus the “black-box” testing effort. Off-the-shelf operating systems need not be validated as a separate program. However, system-level validation testing of the application software should address all the operating system services used, including maximum loading conditions, file operations, handling of system error conditions, and memory constraints that may be applicable to the intended use of the application program.

对于一些成品软件开发工具，如软件编译器、连接器、编辑器和操作系统，器械生产商进行详尽的黑箱测试或许不现实。没有这些测试-验证尝试的一个重要元素-或许不可能验证这些软件工具。然而，可通过其他方式圆满地推断它们的运行。例如，编译器经常由独立的第三方出具测试证明，商业软件产品可能有“故障表”，及从供应商获取的系统需求和其他运行信息，这些需求和信息可同器械生产商的预期用途相比较，以有助于关注这个黑箱测试的尝试。成品操作系统不需要作为一个单独的程序进行验证。然而，这个应用程序软件的系统级验证测试应说明所有使用的操作系统服务，包括最大负载条件，文件操作技巧，系统错误情况的处理，以及可能适用于应用程序的预期用途的内存限制。

For more detailed information, see the production and process software references in Appendix A. 更多详细信息，参见附件A 生产和过程软件参考。

APPENDIX A - REFERENCES

附录A-引用

Food and Drug Administration References

Design Control Guidance for Medical Device Manufacturers, Center for Devices and Radiological Health, Food and Drug Administration, March 1997.

Do It by Design, An Introduction to Human Factors in Medical Devices, Center for Devices and Radiological Health, Food and Drug Administration, March 1997.

Electronic Records; Electronic Signatures Final Rule, 62 Federal Register 13430 (March 20, 1997).

Glossary of Computerized System and Software Development Terminology, Division of Field Investigations, Office of Regional Operations, Office of Regulatory Affairs, Food and Drug Administration, August 1995.

Guidance for the Content of Pre-market Submissions for Software Contained in Medical Devices, Office of Device Evaluation, Center for Devices and Radiological Health, Food and Drug Administration, May 1998.

Guidance for Industry, FDA Reviewers and Compliance on Off-the-Shelf Software Use in Medical Devices, Office of Device Evaluation, Center for Devices and Radiological Health, Food and Drug Administration, September 1999.

Guideline on General Principles of Process Validation, Center for Drugs and Biologics, & Center For Devices and Radiological Health, Food and Drug Administration, May 1987.

Medical Devices; Current Good Manufacturing Practice (CGMP) Final Rule; Quality System Regulation , 61 Federal Register 52602 (October 7, 1996).

Reviewer Guidance for a Pre-Market Notification Submission for Blood Establishment Computer Software, Center for Biologics Evaluation and Research, Food and Drug Administration, January 1997

Student Manual 1, Course INV545, Computer System Validation, Division of Human Resource Development, Office of Regulatory Affairs, Food and Drug Administration, 1997.

Technical Report, Software Development Activities, Division of Field Investigations, Office of Regional Operations, Office of Regulatory Affairs, Food and Drug Administration, July 1987.

Other Government References

W. Richards Adrion, Martha A. Branstad, John C. Cherniavsky. *NBS Special Publication 500-75, Validation, Verification, and Testing of Computer Software,* Center for Programming Science and Technology, Institute for Computer Sciences and Technology, National Bureau of Standards, U.S. Department of Commerce, February 1981.

Martha A. Branstad, John C Cherniavsky, W. Richards Adrion, *NBS Special Publication 500-56, Validation, Verification, and Testing for the Individual Programmer*, Center for Programming Science and Technology, Institute for Computer Sciences and Technology, National Bureau of Standards, U.S. Department of Commerce, February 1980.

J.L. Bryant, N.P. Wilburn, *Handbook of Software Quality Assurance Techniques Applicable to the Nuclear Industry*, NUREG/CR-4640, U.S. Nuclear Regulatory Commission, 1987.

H. Hecht, et.al., *Verification and Validation Guidelines for High Integrity Systems*. NUREG/CR6293. Prepared for U.S. Nuclear Regulatory Commission, 1995.

H. Hecht, et.al., *Review Guidelines on Software Languages for Use in Nuclear Power Plant Safety Systems, Final Report*. NUREG/CR-6463. Prepared for U.S. Nuclear Regulatory Commission, 1996.

J.D. Lawrence, W.L. Persons, *Survey of Industry Methods for Producing Highly Reliable Software*, NUREG/CR-6278, U.S. Nuclear Regulatory Commission, 1994.

J.D. Lawrence, G.G. Preckshot, *Design Factors for Safety-Critical Software*, NUREG/CR-6294, U.S. Nuclear Regulatory Commission, 1994.

Patricia B. Powell, Editor. *NBS Special Publication 500-98, Planning for Software Validation, Verification, and Testing,* Center for Programming Science and Technology, Institute for Computer Sciences and Technology, National Bureau of Standards, U.S. Department of Commerce, November 1982.

Patricia B. Powell, Editor. *NBS Special Publication 500-93, Software Validation, Verification, and Testing Technique and Tool Reference Guide*, Center for Programming Science and Technology, Institute for Computer Sciences and Technology, National Bureau of Standards, U.S. Department of Commerce, September 1982.

Delores R. Wallace, Roger U. Fujii, *NIST Special Publication 500-165, Software Verification and Validation: Its Role in Computer Assurance and Its Relationship with Software Project Management Standards*, National Computer Systems Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, September 1995.

Delores R. Wallace, Laura M. Ippolito, D. Richard Kuhn, *NIST Special Publication 500-204, High Integrity Software, Standards and Guidelines*, Computer Systems Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, September 1992.

Delores R. Wallace, et.al. *NIST Special Publication 500-234, Reference Information for the Software Verification and Validation Process*. Computer Systems Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, March 1996.

Delores R. Wallace, Editor. *NIST Special Publication 500-235, Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*. Computer Systems Laboratory, National Institute of Standards and Technology, U.S. Department of Commerce, August 1996.

International and National Consensus Standards

ANSI / ANS-10.4-1987, *Guidelines for the Verification and Validation of Scientific and Engineering Computer Programs for the Nuclear Industry*, American National Standards Institute, 1987.

ANSI / ASQC Standard D1160-1995, *Formal Design Reviews,* American Society for Quality Control, 1995.

ANSI / UL 1998:1998, *Standard for Safety for Software in Programmable Components,* Underwriters Laboratories, Inc., 1998.

AS 3563.1-1991, *Software Quality Management System, Part 1: Requirements*. Published by Standards Australia [Standards Association of Australia], 1 The Crescent, Homebush, NSW 2140.

AS 3563.2-1991, *Software Quality Management System, Part 2: Implementation Guide*. Published by Standards Australia [Standards Association of Australia], 1 The Crescent, Homebush, NSW 2140.

IEC 60601-1-4:1996, *Medical electrical equipment, Part 1: General requirements for safety, 4. Collateral Standard: Programmable electrical medical systems*. International Electrotechnical Commission, 1996.

IEC 61506:1997, *Industrial process measurement and control – Documentation of application software.* International Electrotechnical Commission, 1997.

IEC 61508:1998, *Functional safety of electrical/electronic/programmable electronic safety-related systems.* International Electrotechnical Commission, 1998.

IEEE Std 1012-1986, *Software Verification and Validation Plans*, Institute for Electrical and Electronics Engineers, 1986.

IEEE Standards Collection, Software Engineering, Institute of Electrical and Electronics Engineers, Inc., 1994. ISBN 1-55937-442-X.

ISO 8402:1994, *Quality management and quality assurance – Vocabulary.* International Organization for Standardization, 1994.

ISO 9000-3:1997, *Quality management and quality assurance standards - Part 3: Guidelines for the application of ISO 9001:1994 to the development, supply, installation and maintenance of computer software*. International Organization for Standardization, 1997.

ISO 9001:1994, *Quality systems – Model for quality assurance in design, development, production, installation, and servicing.* International Organization for Standardization, 1994.

ISO 13485:1996, *Quality systems – Medical devices – Particular requirements for the application of ISO 9001*. International Organization for Standardization, 1996.

ISO/IEC 12119:1994, *Information technology – Software packages – Quality requirements and testing,* Joint Technical Committee ISO/IEC JTC 1, International Organization for Standardization and International Electrotechnical Commission, 1994.

ISO/IEC 12207:1995, *Information technology – Software life cycle processes*, Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 7, International Organization for Standardization and International Electrotechnical Commission, 1995.

ISO/IEC 14598:1999, *Information technology – Software product evaluation,* Joint Technical Committee ISO/IEC JTC 1, Subcommittee SC 7, International Organization for Standardization and International Electrotechnical Commission, 1999.

ISO 14971-1:1998*, Medical Devices – Risk Management – Part 1: Application of Risk Analysis.* International Organization for Standardization, 1998.

Software Considerations in Airborne Systems and Equipment Certification. Special Committee 167 of RTCA. RTCA Inc., Washington, D.C. Tel: 202-833-9339. Document No. RTCA/DO178B, December 1992.

Production Process Software References

The Application of the Principles of GLP to Computerized Systems, Environmental Monograph #116, Organization for Economic Cooperation and Development (OECD), 1995.

George J. Grigonis, Jr., Edward J. Subak, Jr., and Michael Wyrick, "Validation Key Practices for Computer Systems Used in Regulated Operations*," Pharmaceutical Technology,* June 1997.

Guide to Inspection of Computerized Systems in Drug Processing, Reference Materials and Training Aids for Investigators, Division of Drug Quality Compliance, Associate Director for Compliance, Office of Drugs, National Center for Drugs and Biologics, & Division of Field Investigations, Associate Director for Field Support, Executive Director of Regional Operations, Food and Drug Administration, February 1983.

Daniel P. Olivier, "Validating Process Software", *FDA Investigator Course: Medical Device Process Validation*, Food and Drug Administration.

GAMP Guide For Validation of Automated Systems in Pharmaceutical Manufacture,Version V3.0, Good Automated Manufacturing Practice (GAMP) Forum, March 1998: *Volume 1, Part 1: User Guide Part 2: Supplier Guide Volume 2: Best Practice for User and Suppliers.*

Technical Report No. 18, Validation of Computer-Related Systems. PDA Committee on Validation of Computer-Related Systems. PDA Journal of Pharmaceutical Science and Technology, Volume 49, Number 1, January-February 1995 Supplement.

Validation Compliance Annual 1995, International Validation Forum, Inc.

General Software Quality References

Boris Beizer, *Black Box Testing, Techniques for Functional Testing of Software and Systems*, John Wiley & Sons, 1995. ISBN 0-471-12094-4.

Boris Beizer, *Software System Testing and Quality Assurance*, International Thomson Computer Press, 1996. ISBN 1-85032-821-8.

Boris Beizer, *Software Testing Techniques*, Second Edition, Van Nostrand Reinhold, 1990. ISBN 0-442-20672-0.

Richard Bender, *Writing Testable Requirements, Version 1.0,* Bender & Associates, Inc., Larkspur, CA 94777, 1996.

Frederick P. Brooks, Jr., *The Mythical Man-Month, Essays on Software Engineering,* Addison-Wesley Longman, Anniversary Edition, 1995. ISBN 0-201-83595-9.

Silvana Castano, et.al., *Database Security*, ACM Press, Addison-Wesley Publishing Company, 1995. ISBN 0-201-59375-0.

Computerized Data Systems for Nonclinical Safety Assessment, Current Concepts and Quality Assurance, Drug Information Association, Maple Glen, PA, September 1988.

M. S. Deutsch, *Software Verification and Validation, Realistic Project Approaches*, Prentice Hall, 1982.

Robert H. Dunn and Richard S. Ullman, *TQM for Computer Software*, Second Edition, McGraw-Hill, Inc., 1994. ISBN 0-07-018314-7.

Elfriede Dustin, Jeff Rashka, and John Paul, *Automated Software Testing – Introduction, Management and Performance*, Addison Wesley Longman, Inc., 1999. ISBN 0-201-43287-0.

Robert G. Ebenau and Susan H. Strauss, *Software Inspection Process*, McGraw-Hill, 1994. ISBN 0-07-062166-7.

Richard E. Fairley, *Software Engineering Concepts*, McGraw-Hill Publishing Company, 1985. ISBN 0-07-019902-7.

Michael A. Friedman and Jeffrey M. Voas, *Software Assessment - Reliability, Safety, Testability*, Wiley-Interscience, John Wiley & Sons Inc., 1995. ISBN 0-471-01009-X.

Tom Gilb, Dorothy Graham, *Software Inspection*, Addison-Wesley Publishing Company, 1993. ISBN 0-201-63181-4.

Robert B. Grady, *Practical Software Metrics for Project Management and Process Improvement*, PTR Prentice-Hall Inc., 1992. ISBN 0-13-720384-5.

Les Hatton, *Safer C: Developing Software for High-integrity and Safety-critical Systems,* McGraw-Hill Book Company, 1994. ISBN 0-07-707640-0.

Janis V. Halvorsen, *A Software Requirements Specification Document Model for the Medical Device Industry*, Proceedings IEEE SOUTHEASTCON '93, Banking on Technology, April 4th -7th, 1993, Charlotte, North Carolina.

Debra S. Herrmann, *Software Safety and Reliability: Techniques, Approaches and Standards of Key Industrial Sectors,* IEEE Computer Society, 1999. ISBN 0-7695-0299-7.

Bill Hetzel, *The Complete Guide to Software Testing*, Second Edition, A Wiley-QED Publication, John Wiley & Sons, Inc., 1988. ISBN 0-471-56567-9.

Watts S. Humphrey, *A Discipline for Software Engineering*. Addison-Wesley Longman, 1995. ISBN 0-201-54610-8.

Watts S. Humphrey, *Managing the Software Process*, Addison-Wesley Publishing Company, 1989. ISBN 0-201-18095-2.

Capers Jones, *Software Quality, Analysis and Guidelines for Success*, International Thomson Computer Press, 1997. ISBN 1-85032-867-6.

J.M. Juran, Frank M. Gryna, *Quality Planning and Analysis*, Third Edition, , McGraw-Hill, 1993.
ISBN 0-07-033183-9.Stephen H. Kan, *Metrics and Models in Software Quality Engineering*,
Addison-Wesley PublishingCompany, 1995. ISBN 0-201-63339-6.

Cem Kaner, Jack Falk, Hung Quoc Nguyen, *Testing Computer Software*, Second Edition, Vsn
Nostrand Reinhold, 1993. ISBN 0-442-01361-2.Craig Kaplan, Ralph Clark, Victor Tang, *Secrets of Software Quality, 40 Innovations from IBM*,McGraw-Hill, 1995. ISBN 0-07-911795-3.

Edward Kit, *Software Testing in the Real World*, Addison-Wesley Longman, 1995. ISBN 0-201
87756-2.Alan Kusinitz, “Software Validation*”, Current Issues in Medical Device Quality
Systems*,Association for the Advancement of Medical Instrumentation, 1997. ISBN 1-57020-075-0.

Nancy G. Leveson, *Safeware, System Safety and Computers*, Addison-Wesley Publishing
Company, 1995. ISBN 0-201-11972-2.Michael R. Lyu, Editor, *Handbook of Software Reliability
Engineering*, IEEE Computer SocietyPress, McGraw-Hill, 1996. ISBN 0-07-039400-8.

Steven R. Mallory, *Software Development and Quality Assurance for the HealthcareManufacturing
Industries*, Interpharm Press,Inc., 1994. ISBN 0-935184-58-9.Brian Marick, *The Craft of Software Testing*,
Prentice Hall PTR, 1995. ISBN 0-13-177411-5.

Steve McConnell, *Rapid Development*, Microsoft Press, 1996. ISBN 1-55615-900-5.Glenford J. Myers, *The
Art of Software Testing*, John Wiley & Sons, 1979.ISBN 0-471-04328-1.

Peter G. Neumann, *Computer Related Risks*, ACM Press/Addison-Wesley Publishing Co., 1995.
ISBN 0-201-55805-X.Daniel Olivier, *Conducting Software Audits, Auditing Software for Conformance to
FDARequirements*, Computer Application Specialists, San Diego, CA, 1994.

William Perry, *Effective Methods for Software Testing*, John Wiley & Sons, Inc. 1995. ISBN 0471-06097-6.

William E. Perry, Randall W. Rice, *Surviving the Top Ten Challenges of Software Testing*, Dorset House Publishing, 1997. ISBN 0-932633-38-2.

Roger S. Pressman, *Software Engineering, A Practitioner's Approach*, Third Edition, McGraw-Hill Inc., 1992. ISBN 0-07-050814-3.

Roger S. Pressman, *A Manager’s Guide to Software Engineering*, McGraw-Hill Inc., 1993 ISBN 0-07-050820-8.

A. P. Sage, J. D. Palmer, *Software Systems Engineering*, John Wiley & Sons, 1990.

Joc Sanders, Eugene Curran, *Software Quality*, Addison-Wesley Publishing Co., 1994. ISBN 0-
201-63198-9.Ken Shumate, Marilyn Keller, *Software Specification and Design, A Disciplined Approach for
Real-Time Systems*, John Wiley & Sons, 1992. ISBN 0-471-53296-7.

Dennis D. Smith, *Designing Maintainable Software*, Springer-Verlag, 1999.ISBN 0-387-98783-5.Ian
Sommerville, *Software Engineering*, Third Edition, Addison Wesley Publishing Co., 1989. ISBN
0-201-17568-1.

Karl E. Wiegers, *Creating a Software Engineering Culture*, Dorset House Publishing, 1996. ISBN 0-932633-33-1. Karl E. Wiegers, *Software Inspection, Improving Quality with Software Inspections*, Software
Development, April 1995, pages 55-64.Karl E. Wiegers, *Software Requirements*, Microsoft Press, 1999. ISBN 0-7356-0631-5.

APPENDIX B - DEVELOPMENT TEAM

Center for Devices and Radiological Health

Office of Compliance	Stewart Crumpler
Office of Device Evaluation	James Cheng, Donna-Bea Tillman
Office of Health and Industry Programs	Bryan Benesch, Dick Sawyer
Office of Science and Technology	John Murray
Office of Surveillance and Biometrics	Howard Press

Center for Drug Evaluation and Research Office of Medical Policy Charles Snipes
Center for Biologics Evaluation and Research Office of Compliance and Biologics Quality Alice Godziemski
Office of Regulatory Affairs Office of Regional Operations David Bergeson, Joan Loreng