カテゴリ: Final Guidance for Industry and FDA Staff

管理番号: Bpp-Lib-020

改訂番号: 4

名称: CDRH
General Principles of Software Validation

ﾍﾟｰｼﾞ数: 全 80ページ

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

Document issued on: January 11, 2002

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

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

カテゴリ:	Final Guidance for Industry and FDA Staff
管理番号:	Bpp-Lib-020
改訂番号:	4
名称:	CDRH

General Principles of Software Validation

ﾍﾟｰｼﾞ数:	全 80ページ

【注記】

本書は、FDA が発行した英語原文をアズビル株式会社にて和文翻訳したものです。

翻訳文はできるだけ英語原文に忠実になるよう努めましたが、あくまでも英語原文を正とするものです。本書は規制の理解を補助する目的で作成したものであり、アズビル株式会社は、翻訳文に誤りがないことについて保証いたしません。

原文の内容をご自身で必ず確認してください。アズビル株式会社は、本書を利用したことに起因して、お客様に何らかの損害が生じたとしても、これについては一切の責任を負いません。

本書に含まれる内容は、予告なしに変更されることがあります。

【本書の表記について】

- 文脈に応じ説明を補足した場合、〔 〕内にそれを記述しています。
- 英語原文の Italics(斜体字)を日本語訳文では下線としています。但し、書籍名は原文のまま斜体字にしています。
- 1 段落中に 2 箇所以上の訳注挿入があった場合、〔訳注 [1]〕のように段落内のみの通し番号を付け、段落末尾に【訳注 [1]:】と番号を対応させて記述しています。

--- 目 次 ---

Preface

総則

Public Comments

一般向けコメント

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.	コメントと提案は FDA で検討するために、常時受け付ける。送付先は、以下の通りである。 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 コメント送付時には、本ガイダンス文書の正確なタイトルを付すること。本文書の次の改訂または更新時まで、コメントは反映されない場合がある。 本文書の利用または解釈に関する質問事項で、Center for Devices and Radiological Health (CDRH)関連のものについては、下記担当者に連絡のこと。 John F. Murray 電話番号:(301) 594-4659 または Ｅメール:jfm@cdrh.fda.gov 本文書の利用または解釈に関する質問事項で、Center for Biologics Evaluation and Research (CBER)関連のものについては、下記担当者に連絡のこと。 Jerome Davis 電話番号:(301) 827-6220 または Eメール:davis@cber.fda.gov.

Additional Copies

追加コピーの請求

CDRH Additional copies are available from the Internet at :http://www.fda.gov/cdrh/comp/guidance/938.pdf or via CDRH Facts-On-Demand. In order to receive this document via your fax machine, call the CDRH Facts-On-Demand system at 800-899-0381 or 301-827-0111 from a touch-tone telephone. Press 1 to enter the system. At the second voice prompt, press 1 to order a document. Enter the document number 938 followed by the pound sign (#). Follow the remaining voice prompts to complete your request. CBER Additional copies are available from the Internet at: http://www.fda.gov/cber/guidelines.htm, by writing to CBER, Office of Communication, Training, and Manufacturers' Assistance (HFM-40), 1401 Rockville Pike, Rockville, Maryland 20852-1448, or by telephone request at 1-800-835-5709 or 301-827-1800.	CDRH 追加コピーは、ホームページ http://www.fda.gov/cdrh/comp/guidance/938.pdf または、CDRH の Facts-On-Demand を通じて入手できる。FAX で本文書を受信したい場合には、プッシュ信号の出る電話機から、CDRH の Facts-On-Demand システム(電話番号:800-899-0381 または 301-827-0111)に連絡のこと。プッシュボタンの1を押して、システムにアクセスする。文書番号 938 を押し、その後シャープ(#)を押す。あとは音声の指示に従って、請求を完了する。 CBER 追加コピーの入手は、ホームページ http://www.fda.gov/cber/guidelines.htm、または CBER の Office of Communication, Training, and Manufacturers' Assistance (HFM-40), 1401 Rockville Pike, Rockville, Maryland 20852-1448 に文書で請求のこと。電話での請求は、1-800-835-5709 または 301-827-1800 に連絡のこと。

SECTION 1. PURPOSE

SECTION 1. 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.	本ガイダンスは、医療機器ソフトウェアまたは医療機器の設計、開発、製造に用いるソフトウェア・バリデーションに関して、食品医薬品局(FDA)が適切とする一般原則の概要である。この最終ガイダンス文書 2.0 版は、ドラフト文書であった 1997 年 6 月 9 日付けの General Principles of Software Validation, Version 1.1 に代わるものである。

SECTION 2. SCOPE

SECTION 2. 適用範囲

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.	本ガイダンスでは、医療機器の品質システム(Quality System)規制がソフトウェアに対してどのように適用されるか、また、ソフトウェア・バリデーション・システムを評価する上での FDA の現在の姿勢について示す。例えば本文書では、FDA で容認可能としているソフトウェア・バリデーションの要素を挙げているが、法に適合するために常に必須となる作業およびタスクの全てが挙げられているわけではない。
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.	本ガイダンスの適用範囲は、バリデーションという語を最も厳密に定義した場合の適用範囲よりも、やや広いものとなっている。本ガイダンスで論じた、計画、検証(verification)、テスト、トレーサビリティ、コンフィギュレーション管理や、他のグッド・ソフトウェア・エンジニアリング(good software engineering)の様々な局面は、ソフトウェアがバリデートされたという最終的結論を裏づける、重要な活動である。

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.	本ガイダンスでは、ソフトウェア・ライフサイクルの管理、およびリスク管理作業を一本化することを推奨する。開発するソフトウェアの使用目的および安全性に対するリスクに基づいて、ソフトウェア開発者は、具体的なアプローチ、使用する技術の組み合わせ、および、どの程度の労力をかけるかを判断しなければならない。本ガイダンスでは、特定のライフサイクルのモデルや技術、方法の推奨は行わないが、ソフトウェア・ライフサイクルの全過程において、ソフトウェア・バリデーションおよび検証作業を実施することは推奨する。
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. 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.	医療機器製造業者以外が開発したソフトウェア(市販のソフトウェアなど)については、ソフトウェア開発者が、FDA の規制への適合に関する直接の責任を負わないことがある。このような場合、規制に関して責任を負う側(医療機器製造業者など)は、市販ソフトウェア開発者の作業に関して適切性を評価し、医療機器製造業者側の使用目的に合わせてソフトウェア・バリデーションを確立するために、ほかにどのような作業が必要かを判断すること。

2.1. APPLICABILITY

2.1. 適用性

This guidance applies to: ● Software used as a component, part, or accessory of a medical device; ● Software that is itself a medical device (e.g., blood establishment software); ● Software used in the production of a device	本ガイダンスは、以下のソフトウェアに適用される。 ● 医療機器のコンポーネント、部品、または付属品として用いるソフトウェア ● ソフトウェア自体が医療機器となるもの(血液管理施設(blood establishment)用のソフトウェアなど) ● 医療機器製造に用いるソフトウェア(製造機器におけ

(e.g., programmable logic controllers in manufacturing equipment); and ● Software used in implementation of the device manufacturer's quality system (e.g., software that records and maintains the device history record). 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.	るプログラマブル・ロジック・コントローラ(PLC)など) ● 医療機器製造業者による品質システム実装用のソフトウェア(医療機器・ヒストリ・レコード(device history record)を記録および維持するためのソフトウェア) 本文書は、一般に認められているソフトウェア・バリデーションの原則に基づいており、従って、どのソフトウェアに対しても適用することができる。FDA としては、連邦食品医薬品化粧品法(FFDCA)の Section201(h)、および、ソフトウェアと規制に関する FDA の現行の方針に定められた規制の対象となる医療機器関連の全てのソフトウェアに、本ガイダンスを適用する方針である。本文書では、具体的にどのソフトウェアが規制の対象となるかについては特定していない。

2.2. AUDIENCE

2.2. 対象

This guidance provides useful information and recommendations to the following individuals: ● Persons subject to the medical device Quality System regulation ● Persons responsible for the design, development, or production of medical device software ● 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 ● FDA Investigators ● FDA Compliance Officers	本ガイダンスは、以下に該当する個人に実用的な情報および推奨事項を提供するものである。 ● 医療機器の品質システム規制の対象者 ● 医療機器ソフトウェアの設計、開発または製造の責任者 ● 医療機器、または品質システム自体の実装に用いるソフトウェア・ツールの設計、開発、製造に使用する、自動ツールの設計や開発、製造、または調達の責任者 ● FDA 査察官 ● FDA のコンプライアンス・オフィサー ● FDA の科学的レビュー担当者

● FDA Scientific Reviewers	

2.3. THE LEAST BURDENSOME APPROACH

2.3. 最小負担のアプローチ

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: http://www.fda.gov/cdrh/resolvingdisputes/ombudsman.html	医療機器の規制の全分野について、負担が最小となるアプローチを検討するべきだというのが、FDA の信念である。本ガイダンスには、関連する科学的要件および法的要件をよく検討し、適合にあたっての負担を最も軽くすると考えられるものを記載した。しかし、より負担の軽いアプローチが他に考えられる場合には、当方でも検討できるので連絡されたい。書面によるコメントは、本ガイダンスの序文に示した担当者、または CDRH のオンブズマン宛に送付のこと。CDRH のオンブズマンに関しては、連絡先も含めて、下記のホームページで様々な情報を掲載しているので参照されたい。

2.4. REGULATORY REQUIREMENTS FOR SOFTWARE VALIDATION

2.4 ソフトウェア・バリデーションの規制要件

The FDA's analysis of 3140 medical device recalls conducted between 1992 and 1998 reveals that 242 of them (7.7%) are attributable to software failures. Of those software related recalls, 192 (or 79%) 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.	FDA が 1992 年から 1998 年にかけて行った医療機器のリコールに関する分析では、調査した 3140 件のうち 242 件(7.7%)が、ソフトウェアの不具合によるものであったことが明らかになった。このソフトウェア関連のリコールのうち 192 件(79%)が、最初に製造、出荷された後でソフトウェアを変更したことにより生じたソフトウェアの欠陥が原因であった。本ガイダンスで論じるソフトウェア・バリデーションや他のグッド・ソフトウェア・エンジニアリング・プラクティス（good software engineering practices)は、このような欠陥やその結果生ずるリコールを回避する重要な手段である。

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.	ソフトウェア・バリデーションは、品質システム規制要件である。品質システム規制は、1996 年 10 月 7 日の米国官報で発布され、1997 年 6 月 1 日に発効したものである。(Title21 CFR の Part 820 および 61 米国官報(FR)52602 ページをそれぞれ参照のこと。)バリデーション要件が適用されるのは、医療機器の部品となるソフトウェア、ソフトウェア自体が医療機器となるもの、および、医療機器の製造または医療機器製造業者の品質システムの実装に用いるソフトウェアである。
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 in 21 CFR §820.30(g). Other design controls, such as planning, input, verification, and reviews, are required for medical device software. (See 21 CFR §820.30.) The corresponding documented results from these activities can provide additional support for a conclusion that medical device software is validated.	分類に関する規制(classification regulation)で特に免除対象となっていない限り、1997 年 6 月 1 日以降に開発された医療機器ソフトウェア製品は、医療機器の種類に関わらず設計管理規定の対象となる。(21CFR Section 820.30 を参照のこと。)本要件の対象となるのは、現在進行中の開発プロジェクト、全ての新規開発プロジェクト、および既存の医療機器ソフトウェアに対する全ての変更である。医療機器ソフトウェア・バリデーションに関する具体的な要件は、21 CFR Section 820.30(g)に示されている。医療機器のソフトウェアについては、その他にも計画、入力、検証およびレビューなどの設計管理が必要となる。(21 CFR Section 820.30 を参照のこと。)これらの活動から得られる文書化された結果は、医療機器ソフトウェアがバリデートされたという結論を導く上で、付加的な裏づけとなる。
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	医療機器製造工程または品質システムを自動化するソフトウェアは、それが一部分であっても、使用目的に対してバリデートすることが必須となる。これは、21 CFR Section 820.70(i)で要件とされている。本要件は、医療機器設計、テスト、部品の受入、製造、ラベリング、梱包、出荷、苦情への対応、または、品質システムのその

acceptance, manufacturing, labeling, packaging, distribution, complaint handling, or to automate any other aspect of the quality system.	他の部分を自動化する際に用いる全てのソフトウェアに適用される。
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.	さらに、電子記録の作成、修正、維持、および電子署名の管理に用いるコンピュータ・システムも、バリデーション要件の対象となる。(21 CFR Section 11.10(a)を参照。)このようなコンピュータ・システムについては、正確性、信頼性、常に意図通りの性能かどうか、無効な記録または記録の改竄を識別する性能について、確認するためのバリデートが必須となる。
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.	上記のアプリケーション用ソフトウェアは、社内または(契約により)社外で開発されるが、特定の意図で使用するものについては、市販のソフトウェアを購入するケースも多い。製造システム／品質システムに用いる全てのソフトウェアについては、市販品を購入した場合であっても、ソフトウェアが使用目的に対してバリデートされていることを示すため、使用目的を完全に定義した要件、およびテスト結果や証拠と比較評価できるような情報を文書化すること。
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	自動化された医療機器、および自動化された製造／品質システムのオペレーションにおいては、市販ソフトウェアの使用が増加している。市販ソフトウェアは多くの機能を有していても、医療機器製造業者が必要とする機能はごく一部である。医療機器製造業者は、医療機器で使用されるソフトウェア、および医療機器の製造に使用されるソフトウェアの妥当性に関して医療機器製造業者が責任をもつ。医療機器製造業者が「市販」ソフトウェアを購入する場合は、当該アプリケーション上で、意図どおりの性能が出るかどうかを確認することが必須である。製造／品質システムで用いる市販のソフトウェアについては、本文書の Section 6.3 に追加のガイダンスを掲載した。医

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.*	療機器ソフトウェアに関しては、FDA の Guidance for Industry、FDA Reviewers、および Compliance on Off-The-Shelf Software Use in Medical Devices に、有益と思われる情報が掲載されている。

2.4. QUALITY SYSTEM REGULATION VS PRE-MARKET SUBMISSIONS

2.4. 品質システムの規制と市販前の提出物

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.	本文書は、ソフトウェア・バリデーションの実施などに関する品質システム規制の課題について述べるものであり、ソフトウェア・バリデーションのプロセスの管理および統制に関するガイダンスとなるものである。ソフトウェア・バリデーションのプロセスの管理および統制は、自動製造プロセスのプロセス・バリデーションなどの、他のバリデーション要件と混同すべきものではない。
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	医療機器製造業者は、品質システム要件への適合、および設計管理要件への適合に別々の手順と記録を用いる必要はない。また、市販前の FDA への提出についても同様である。本文書では、ソフトウェア・バリデーションに関する具体的な安全性または有効性に関する問題は取り扱わない。規制対象となるソフトウェアの設計課題および市販前の提出物に関する文書要件については、本文書では述べていない。安全性と有効性、および市販前の提出物に必要な文書に関する具体的な課題については、Center for Devices and Radiological Health (CDRH)の Office of Device Evaluation (ODE)、または Center for Biologics Evaluation and Research (CBER)の Office of Blood Research and Review 宛に送付すること。市販前の提出物に適用される FDA のガイダンス文書については、付録 A の参考資料を参照のこと。

references in Appendix A for applicable FDA guidance documents for pre-market submissions.	

SECTION 3. CONTEXT FOR SOFTWARE VALIDATION

SECTION 3. ソフトウェア・バリデーションの背景

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.	ソフトウェア・バリデーションに関して、品質システムの規制に確実に適合するために、FDA が何を求めているかを示した具体的なガイダンスを求める声が多かった。本文書に示したソフトウェア・バリデーションに関する情報は、決して新しいものではない。Section 4 と 5 の原則およびタスクを用いたソフトウェア・バリデーションは、20 年以上にも渡って、ソフトウェア業界の各所で行われてきたものである。
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.	医療機器、プロセス、製造設備は多様であるため、具体的なバリデーションの全要素を、一つの文書で述べることは不可能である。しかし、いくつかの広範な概念を、ソフトウェア・バリデーションのガイダンスとして、一般適用することも可能であろう。このような広範な概念からは、ソフトウェア・バリデーションに対する包括的アプローチを構築するための枠組みが得られる。具体的な情報については、さらに付録 A の参考資料を参照のこと。

3.1. DEFINITIONS AND TERMINOLOGY

3.1. 定義と用語

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*.	本文書で用いる用語で、品質システムの規制または下記で定義した以外のものについては、全て FDA の Glossary of Computerized System and Software Development Terminology の現行版で定義した通りである。

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. 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.	医療機器の品質システムの要件(21 CFR 820.3(k))では、**「確立(establish)」**を「定義し、文書化して実行すること」と定義している。本ガイダンスでも、「確立(establish)」や「確立された(established)」は、同様に解釈されるものとする。 医療機器の品質システムの規制にある定義には、ソフトウェア業界で一般的に用いられている用語と比較した場合に、混同されそうなものがある。この例としては、要件、仕様、検証、バリデーションなどが挙げられる。

3.1.1 Requirements and Specifications

3.1.1. 要件と仕様

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	品質システム規制では、設計の入力となる要件を文書化し、定められた要件を検証することを必須事項と定めているが、規制の中では「要件」と「仕様」という用語を、それ以上明確に区別していない。**「要件」**とは、システムまたはシステム上のソフトウェアに要求または期待されるもの全てを指す。要件は、明確に規定された、あるいは状況から推測される顧客のニーズを示す。また、社内の要件のみならず、市場ベースの要件、契約上の要件、または法律上の要件を指す場合も多い。要件には、多くの異なる種類が考えられる(設計、機能、実装、インターフェース、性能、または物理的要件など)。ソフトウェア要件は一般的に、システム要件から派生したもので、システム機能の中でソフトウェアに割り当てた部分に対するものである。ソフトウェア要件は機能の側面から記述したもので、開発プロジェクトの進行に沿って定義、詳細化し、更新される。ソフトウェア要件を正確かつ完全に文書化することは、製作されるソフトウェア・バリデーションの成否を左右する極めて重要な要因である。

software requirements is a crucial factor in successful validation of the resulting software. 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.	**「仕様」**は、「要件を記した文書」と定義されている。(CFR 21 Section 820.3(y)を参照。)これには、図、パターンや他の関連文書等があり、また通常、要件への適合をチェックする方法および基準を示すものである。仕様書には、システム要求仕様書、ソフトウェア要求仕様書、ソフトウェア設計仕様書、ソフトウェア・テスト仕様書、ソフトウェア統合仕様書など、多くの種類がある。このような文書は全て、「定められた要件」を確立するものであり、また、設計のアウトプットとして種々の検証が必要となるる。

3.1.2 Verification and Validation

3.1.2. 検証／バリデーション

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. **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,	品質システム規制は、1994 年版 ISO 8402 と調和(harmonize)している。ISO8402 では「検証(verification)」と「バリデーション(validation)」を別の用語として、区別して取り扱っている。一方、エンジニアリング専門誌の記事やテキストでは、「検証(verification)」と「バリデーション(validation)」を、同じような意味で用いていることも多い。また、ソフトウェアの「検証、バリデーション、テスト(verification, validation and testing; VV&T)」を一つの概念として、三つの用語を区別していない場合もある。 **「ソフトウェアの検証」**によって、ソフトウェア開発ライフサイクルにおける、あるフェーズでの設計のアウトプットが、あらかじめ定められていた全ての要件(specified requirements)を満たす客観的な証拠が得られる。ソフトウェアの検証は、開発中のソフトウェアの一貫性、完全

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.	性、正確性、およびその裏づけのための文書を調べるものであり、ソフトウェアがバリデートされたという結論を裏づけるものである。検証には多くのアクティビティがあるが、ソフトウェアのテストはそのうちの一つであり、ソフトウェア開発の出力が入力要件を満たしているということを確認するものである。その他の検証アクティビティには、様々な静的・動的な解析、コードおよび文書の検査、ウォーク・スルーなどの技術がある。
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	**「ソフトウェア・バリデーション」**は、最終医療機器に対する設計バリデーションの一部であるが、品質システム規制の中では、個別に定義されてはいない。本ガイダンスにおいて FDA は、ソフトウェア・バリデーションを以下のように定義している。**「ソフトウェアの仕様がユーザのニーズおよび使用目的に適合するということ、また、ソフトウェアで実装された要件が一貫して満たされているということを、検査(examination)と客観的証拠によって確認すること。」**実際のソフトウェア・バリデーション活動は、全要件を満たしたということを確認するために、ソフトウェア開発ライフサイクルの途中と終わりの両方で行われることが多い。ソフトウェアは通常、より大きなハードウェア・システムの一部であるため、ソフトウェア・バリデーションには一般的に、ソフトウェアの全要件が正しくかつ完全に実装されたという証拠が含まれ、システム要件と関連づけられるようになっている。ソフトウェアがバリデートされたと結論づけられるかどうかは、ソフトウェア開発ライフサイクルの各段階で行われる包括的なソフトウェアのテスト・点検・分析やその他の検証タスクに大きく左右される。一般的に、ソフトウェアで自動化された医療機器の設計バリデーション・プログラムには、模擬的な使用環境での医療機器・ソフトウェアの機能テスト、およびユーザ・サイトにおけるテストが含まれる。

are typically included as components of an overall design validation program for a software automated device. 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.	ソフトウェアの検証／バリデーションは、開発者が永久にテストを継続することもできず、どの程度の証拠があれば十分であるかを判断しにくいところが難しい。ソフトウェア・バリデーションとは、その医療機器において、ソフトウェアで自動化した機能および特徴が、全ての要件とユーザの期待を満足するという「ある一定のレベルの信頼性」を得るものである。仕様文書中の誤り〔の発見〕、残存欠陥の推定、テスト・カバレージなどの方策はいずれも、出荷前に一定の信頼性のレベルを得るために実施されるものである。信頼性のレベル、すなわちそのために必要なソフトウェア・バリデーション、検証、テストに必要な労力のレベルは、医療機器を自動化した機能から生ずる安全性に対するリスク(危険)に応じたものとなる。ソフトウェアの安全性に対する危機の管理ついては、付録 A に挙げた FDA の Guidance for the Content of Pre-market Submissions for Software Contained in Medical Devices、および ISO/IEC 14971-1 and IEC 60601-1-4 の国際基準に、付加的なガイダンスが記載されている。

3.1.3 IQ/OQ/PQ

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	FDA および規制対象の業界の双方は、長い間ソフトウェア・バリデーションをプロセス・バリデーションの用語で理解し、定義しようとしてきた。例えば、業界用文書やその他の FDA のバリデーション・ガイダンスでは、ユーザ・サイトのソフトウェア・バリデーションを、据付時適格性評価(IQ)、運転時適格性評価(OQ)、稼動性能適格性評価

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.	(PQ)と記述している場合がある。これらの用語の定義および IQ/OQ/PQ に関する付加情報は、FDA が 1987 年 5 月 11 日に発行した Guideline on General Principles of Process Validation、および 1995 年 8 月発行の Glossary of Computerized System and Software Development Terminology に記載されている。
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.	IQ/OQ/PQ という用語は、その目的を十分に果たしてきたし、ユーザ・サイトのソフトウェア・バリデーションのタスクを整理する適正な方法の一つであるが、ソフトウェアの専門家に十分な理解が得られていない可能性もあり、本文書の他の箇所でも、この用語を使用していない。しかし、FDA 職員および医療機器製造業者は、ソフトウェア・バリデーションに関する情報を請求、提供する立場にあるので、これらの用語の違いを知っておく必要がある。

3.2. SOFTWARE DEVELOPMENT AS PART OF SYSTEM DESIGN

3.2. システム設計の一部としてのソフトウェア開発

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.	システム機能を、ソフトウェアを用いて実装するという判断は、概してシステムの設計中に行われるものである。一般的にソフトウェア要件は、システム要件／設計のうち、ソフトウェアを用いて実装する部分を切り出したものである。一般的に最終的に製造される医療機器（以下、最終医療機器）に関するユーザのニーズおよび使用目的では、ユーザはハードウェア、ソフトウェア、またどの組み合わせで実現するかということまでは指定しない。従って、ソフトウェア・バリデーションは、システムの設計バリデーション全般の中で考慮することが必須となる。

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.	文書化された要求仕様書は、製品開発の背景となったユーザのニーズおよび使用目的を表している。ソフトウェア・バリデーションの第一の目的は、完成した全ソフトウェア製品が、その文書化された全てのソフトウェアおよびシステム要件に適合しているということを示すことである。医療機器システム要件およびソフトウェア要件の両方に関する正確性および完全性は、医療機器の設計バリデーション・プロセスの一環として取り扱うべきである。ソフトウェア・バリデーションでは、全てのソフトウェアの仕様に対する適合の確認、および、ソフトウェアの全ての要件がシステム仕様に関連づけられることの確認も行う。確認作業は設計バリデーション全体において重要な部分であり、その医療機器の全ての側面をユーザのニーズおよび使用目的に確実に適合させるものである。

3.3. SOFTWARE IS DIFFERENT FROM HARDWARE

3.3. ソフトウェアとハードウェアの違い

While software shares many of the same engineering tasks as hardware, it has some very important differences. For example:	ソフトウェアのエンジニアリング・タスクの多くは、ハードウェアと共通であるが、以下のような非常に重要な違いもある。
● 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	● ソフトウェアの問題の大部分は、設計および開発プロセスのエラーに起因している。ハードウェア製品の品質は、設計、開発および製造に左右されるところが大きいが、ソフトウェア製品の品質は、主に設計および開発に左右され、ソフトウェア製造とはほとんど関わりがない。ソフトウェア製造とは、複製することであり、これは容易に検証することができる。オリジナルと全く同じ機能を持つプログラムのコピーを、何千も作ることは困難ではない。オリジナルのプログラムを全ての仕様に適合させることが難しいのである。

program copies that function exactly the same as the original; the difficulty comes in getting the original program to meet all specifications.	
● 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.	● ソフトウェアの最も重要な特色は、分岐、すなわち、異なる入力により、別の一連の命令を実行できる能力である。この特徴は、ソフトウェアのもう一つの特質である複雑性に、大きく寄与する要因となっている。短いプログラムであっても、非常に複雑で、完全に理解することは困難な場合が多い。
● 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.	● 一般的に、テストだけでは、ソフトウェアが完全かつ正確であるということを検証することはできない。テストのほかに、他の検証技術および構造的かつ文書化された開発プロセスを組み合わせること。これは、包括的なバリデーションのアプローチを確実なものとするためである。
● 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.	● ハードウェアと違ってソフトウェアは、物理的な実体はなく、経年変化するものでもない。実際にソフトウェアは、時と共に潜在的な欠陥が発見されて取り除かれるため、改良されることが多い。しかし、ソフトウェアは常に更新、変更されるため、変更中にソフトウェアに新たな欠陥が作りこまれることもある。
● 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	● ある種のハードウェアの欠陥と違い、ソフトウェアの欠陥は、事前の警告なしに発生する。ソフトウェアの潜在的な欠陥は、ソフトウェア実行中に異なるパスを通るようにする分岐が存在するため、市場に流通してからかなり後になるまで明らかにならない場合も多い。

marketplace.	
● 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.**	● さらにこれと関連するソフトウェアの特質は、ソフトウェアの変更の速さおよび容易さである。この要因によって、ソフトウェアの専門家もそうでない者も、ソフトウェアの問題点は簡単に修正できるものだと信じてしまう。ソフトウェアに関する理解不足と相まって、管理者は、ソフトウェアにはハードウェアのような厳重に管理されたエンジニアリングは必要ないと思ってしまいかねない。実際にはその反対である。**ソフトウェアの開発プロセスは複雑であるため、後になって検出しにくい問題を残さないようにするためには、ハードウェア以上に厳重に管理する必要がある。**
● 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.	● ソフトウェアのコード上の変更は、一見小さな変更と思えるものであっても、そのソフトウェア・プログラムのほかの箇所に予想外かつ非常に重大な問題を引き起こすことが多い。ソフトウェア開発プロセスには、ソフトウェアの変更による予想外の結果を検出し、修正できるようにするために、十分な計画および管理を行い、文書化しておくこと。
● 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.	● ソフトウェアの専門家の需要は高く、転職も多いため、ソフトウェアの維持変更を行う人は、オリジナルのソフトウェア開発には関わっていないことが多い。従って、正確かつ完全な文書が非常に重要となる。
● Historically, software components have not	● これまで、ソフトウェア部品は、ハードウェア部品ほ

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.	ど頻繁に標準化されておらず、互換性も存在しなかった。しかし、医療機器ソフトウェア開発者は、部品を前提とした開発ツールおよび技術を利用し始めている。オブジェクト指向メソドロジーと市販のソフトウェア部品の利用により、より速くて安価なソフトウェアの開発が期待されている。しかし、部品に基づく方法は、統合の際に、非常に慎重な配慮が必要となる。統合の前には、再利用可能なソフトウェアのコードを定義し開発する十分な時間、および市販の部品の動作を理解する十分な時間が必要となる。
For these and other reasons, software engineering needs an even greater level of managerial scrutiny and control than does hardware engineering.	**上記のような理由から、ソフトウェア・エンジニアリングは、ハードウェア・エンジニアリング以上に処理上の精密な検査と管理が必要なのである。**

3.4. BENEFITS OF SOFTWARE VALIDATION

3.4. ソフトウェア・バリデーションによる利点

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	ソフトウェア・バリデーションは、医療機器ソフトウェア、およびソフトウェアで自動化したオペレーションの品質保証のための重要な手段である。ソフトウェア・バリデーションによって医療機器の有用性と信頼性は高まり、その結果、欠陥の起こる割合やリコール、修正措置が減り、患者およびユーザへのリスク、医療機器製造業者の法的負担が少なくなる。また、ソフトウェア・バリデーションによって、ソフトウェアの確実な修正、およびソフトウェアの変更に伴う再バリデートが容易かつ安価に行えるようになり、長期的にコストが削減される。ソフトウェアのメンテナンスにかかる費用は、ライフサイクル全体から見ると、全コストの中で非常に大きな割合を占めることが多い。確立された包括的なソフトウェア・バリデーションのプロセスは、その後のソフトウェア・レリース毎のバリデーション・

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.	コストを減少させるため、長期的にはソフトウェアのコストを減らすことができる。

3.5 DESIGN REVIEW

3.5　設計レビュー

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.	設計レビューとは、文書化された設計の包括的、体系的な検査であり、その目的は以下の通りである。 ● 設計要件の妥当性の評価 ● 設計が要件を満たせるかどうかの評価 ● 問題点を明らかにすること ソフトウェアの開発期間中に、開発チームは非公式のレビューを何度も行うことが多いが、公式の設計レビューはより体系的で、開発チーム外の人員が参加するものである。公式の設計レビューは、他の公式および非公式のレビューの結果を参照しても、含めてもよい。設計レビューは、ソフトウェアについて独立して行っても、ソフトウェアをハードウェアに統合した後にシステム全体として行っても、またはこの両方を行ってもよい。設計レビューでは、開発計画、要求仕様書、設計仕様書、テスト計画／手順、プロジェクト関連のその他の文書および作業、ライフサイクルの各段階における検証結果、医療機器全体のバリデーションの結果、などを検査すべきである。
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	設計レビューは、開発プロジェクトの管理および評価のための、最も重要な手段である。例えば、公式の設計レビューによって、管理者はソフトウェア・バリデーション計画で定義した目標が、全て達成されたということを確認できる。品質システム規制では、医療機器の設計プロセスの中で、公式の設計レビューを少なくとも１回は実施する

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.	よう定めている。しかし、設計レビューは、複数回（すなわち、ソフトウェア・ライフサイクル活動の各段階の終了時、次の活動に移る前の準備として）行うことが推奨される。要件定義活動の終了時または終了直前に、多くのリソースを特定の設計課題解決のために充てる前に、公式の設計レビューを実施することが特に重要である。この時点で発見された問題点は、より容易に解決でき、時間的、経済的節約になり、重大な課題を見逃す可能性を減らすことにつながる。
Answers to some key questions should be documented during formal design reviews. These include:	以下のようなキーとなる質問への答を、設計レビューの際に文書化すること。
● Have the appropriate tasks and expected results, outputs, or products been established for each software life cycle activity?	● ソフトウェア・ライフサイクルの各活動について、適切なタスクおよび期待される結果、成果物、または製品が確立できたか？
● Do the tasks and expected results, outputs, or products of each software life cycle activity: ✓ Comply with the requirements of other software life cycle activities in terms of correctness, completeness, consistency, and accuracy? ✓ Satisfy the standards, practices, and conventions of that activity? ✓ Establish a proper basis for initiating tasks for the next software life cycle activity?	● ソフトウェア・ライフサイクルの各活動のタスクおよび期待される結果、成果物、または製品は、下記に当てはまるか？ ✓ 他のソフトウェア・ライフサイクル活動の要件に、正確性、完全性、一貫性、精密性という点で適合しているか。 ✓ その活動の基準、規範(practices)、決まりごと(conventions)を満たしているか。 ✓ 次のソフトウェア・ライフサイクルの活動のタスクを開始するための適切な基盤が確立されたか。

SECTION 4. PRINCIPLES OF SOFTWARE VALIDATION

SECTION 4. ソフトウェア・バリデーションの原則

This section lists the general principles that should be considered for the validation of software.	本セクションでは、ソフトウェア・バリデーションを実施する上で、考慮すべき一般原則を挙げる。

4.1. REQUIREMENTS

4.1. 要件

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)).	文書化されたソフトウェア要求仕様は、バリデーションおよび検証の両方に対する基盤となる。ソフトウェア・バリデーションのプロセスは、ソフトウェア要求仕様書(21 CFR 820.3(z)、(aa)、820.30(f)および(g)を参照)を確立しなければ、完了しない。

4.2. DEFECT PREVENTION

4.2. 欠陥の防止

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.	ソフトウェアの品質保証では、ソフトウェア開発プロセスで欠陥を作り込まないようにし、ソフトウェアをコーディングしてからテストによって品質を作り込むということのないようにすることが重要である。ソフトウェアのテストでは、ソフトウェアのコードの中の潜在的欠陥を表面化させる能力は、非常に限定されている。例えば、ほとんどのソフトウェアは複雑すぎて、徹底したテストを実施することができない。**ソフトウェアのテストは、必要な作業である。しかし、ほとんどの場合、ソフトウェアのテストだけでは、そのソフトウェアが意図した使用目的に合っているかどうかという信頼性を確立するには、不十分である。**この信頼性を確立するには、ソフトウェア開発者は、ソフトウェアのエラー予防および発生するエラーの検出のための方法と技術を組み合わせて用いなければならない。何が「最良の組み合わせ」かは、開発環境、アプリケーション、プロジェクトの規模、言語、リスクなどの、多くの要因によって異なってくる。

4.3. TIME AND EFFORT

4.3. 時間と労力

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.	ソフトウェアがバリデートされたという十分な論拠を築くには、時間と労力を要する。ソフトウェア・バリデーションの準備は、設計や開発の計画段階、また設計インプット中など、早期に開始しなければならない。ソフトウェアがバリデートされたという最終結論は、ソフトウェア・ライフサイクルを通じて、計画に基づいて実施された作業から収集した証拠に基づくものでなければならない。

4.4. SOFTWARE LIFE CYCLE

4.4. ソフトウェアのライフサイクル

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.	ソフトウェア・バリデーションは、確立されたソフトウェア・ライフサイクル環境の中で行うものである。ソフトウェア・ライフサイクルには、ソフトウェア・バリデーションの裏づけに必要な、ソフトウェア・エンジニアリング・タスクおよび文書が含まれる。さらに、ソフトウェア・ライフサイクルには、ソフトウェアの使用目的に合った、検証およびバリデーションのタスクも含まれる。本ガイダンスでは、特定のライフサイクル・モデルを推奨していないが、ソフトウェアの開発プロジェクトでは、何らかのライフサイクルのモデルを選定および使用すべきである。

4.5. PLANS

4.5. 計画

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	ソフトウェア・バリデーションのプロセスは、計画により定義され、管理される。ソフトウェア・バリデーション計画では、ソフトウェア・バリデーション業務を通じて、「何を」達成するかを明らかにする。ソフトウェア・バリデーション計画は、品質システムにおける重要なツールである。ソフトウェア・バリデーション計画書には、適用範囲、アプロー

specify areas such as scope, approach, resources, schedules and the types and extent of activities, tasks, and work items.	チ、リソース、実施スケジュール、および作業、タスク、作業項目の種類と範囲などを明記する。

4.6. PROCEDURES

4.6. 手順

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.	ソフトウェアのバリデーション・プロセスは、手順に従って実行する。この手順では、ソフトウェア・バリデーション業務を実施する「方法」を確立する。手順書では、それぞれのバリデーション活動、タスクおよび作業項目を完了する上で必須となる具体的なアクションまたはアクションの順序を明らかにすべきである。

4.7. SOFTWARE VALIDATION AFTER A CHANGE

4.7. 変更後のソフトウェア・バリデーション

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.	ソフトウェアは複雑であるため、一見小さく部分的と思える変更であっても、システム全体に大きな影響を及ぼすことが多い。ソフトウェアに変更を行った場合は(小さな変更であっても)、そのソフトウェアがバリデートされている状態を再確立(re-established)する必要がある。**ソフトウェアを変更した場合には必ず、バリデーション分析を実施しなければならない。これは、各変更部分のバリデーションだけではなく、ソフトウェア・システム全体に対する変更範囲および影響度を判断するために実施する。**その後この分析に基づいて、ソフトウェア開発者は、未変更ではあるが脆弱な部分に悪影響が出ていないことを示すため、適切なレベルのソフトウェア・リグレッション・テストを実施すること。設計管理および適切なリグレッション・テストにより、ソフトウェアが変更後にバリデートされたという信頼が得られる。

4.8. VALIDATION COVERAGE

4.8. バリデーションのカバレージ

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.	バリデーションのカバレージは、企業規模やリソースによる制約ではなくて、ソフトウェアの複雑性、および安全性に対するリスクに基づいたものとすべきである。バリデーション活動、タスク、作業項目の選択は、ソフトウェアの設計上の複雑さ、および仕様書に記載した使用目的での利用に伴うリスクに見合うものとすること。リスクの少ない医療機器では、基本的なバリデーション活動のみを実施する場合が多い。リスクの増加に伴い、そのリスクをカバーするようなバリデーション活動を追加すること。バリデーション文書は、ソフトウェア・バリデーションの全ての計画および手順が無事完了したことを示す上で、十分なものとすること。

4.9. INDEPENDENCE OF REVIEW

4.9. レビューの独立性

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.	バリデーション作業は、「レビューの独立性(independence of review)」を基本的な品質保証指針として実施すべきである。ソフトウェア開発者自身で行うバリデーションは、極めて難しいものである。可能であれば、独立した評価を実施する方がよい。リスクの高いアプリケーションの場合、特に独立した評価が必要である。独立した検証およびバリデーションを第三者に委託する企業もあるが、この方法が常に可能であるとは限らない。特定の設計または設計の実装に関わっていないもののプロジェクトの評価、検証およびバリデーション活動を行えるだけの十分な知識を備えた、内部のスタッフを選任するという方法もある。規模の小さな企業が社内でレビューの独立性を維持するには、タスクの編成および割り当てを行う上で工夫が必要かもしれない。

4.10. FLEXIBILITY AND RESPONSIBILITY

4.10. 柔軟性と責任

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.	このようなソフトウェア･バリデーションの原則をどのように実施するかは、アプリケーションによってかなり異なってくるであろう。医療機器製造業者は、バリデーション原則をどのように適用させるかを選択できるが、ソフトウェアがバリデートされたことを証明する最終的な責任も有することになる。
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: ● Is a component, part, or accessory of a medical device; ● Is itself a medical device; or ● Is used in manufacturing, design and development, or other parts of the quality system.	ソフトウェアは、多様な環境の中で、また様々なリスク･レベルを持つ多様な医療機器に対して、設計され、開発され、バリデートされ、規制を受ける。FDA は下記のソフトウェアを含む、医療機器アプリケーションを規制してきた。 ● 医療機器の部品、一部、または付属品となるソフトウェア ● それ自体が医療機器となるもの ● 製造、設計、および開発、または品質システムのその他の部分で用いられるソフトウェア
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	それぞれの環境で、アプリケーションの作成に用いられるソフトウェア部品は、様々な所から入手されたものであることが多い(社内開発のソフトウェア、市販のソフトウェア、外注のソフトウェア、シェアウェアなど。)さらに、ソフトウェア部品にはさまざまな形式がある(アプリケーション･ソフトウェア、オペレーティング･システム、コンパイラ、デバッガ、コンフィギュレーション管理ツール　等々。)このような環境下では、ソフトウェア･バリデーションは複雑になりうるため、ソフトウェア･バリデーション･プロセスを設計するときに、ソフトウェア･バリデーションの原則を考慮しておくことが適切である。その結果得られたソフトウェア･バリデーション･プロセスは、システム、医療機器、または

designing the software validation process. The resultant software validation process should be commensurate with the safety risk associated with the system, device, or process.	プロセスの安全性へのリスクに応じたものとなる。
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.	ソフトウェア・バリデーション活動およびタスクは、分散し、別の場所で発生し、異なる組織で処理されるかもしれない。しかし、医療機器製造業者、すなわち仕様を作る者は、タスクや契約関係、コンポーネントの入手先、開発環境の区分に関わらず、ソフトウェアをバリデートする上での最終的な責任を持つことになる。

SECTION 5. ACTIVITIES AND TASKS

SECTION 5. 作業とタスク

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.	ソフトウェア・バリデーションは、ソフトウェア開発ライフサイクルの様々な段階で、計画、実行される一連の活動とタスクによって達成される。これらのタスクは、採用したライフサイクル・モデル、およびソフトウェア・プロジェクトが進む中で行われる変更の範囲に応じて、一度または繰り返し何度も行われる。

5.1. SOFTWARE LIFE CYCLE ACTIVITIES

5.1. ソフトウェア・ライフサイクル活動

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	本ガイダンスでは、特定のソフトウェア・ライフサイクル・モデルは推奨していない。ソフトウェアの開発者は、製品および組織に適合したソフトウェア・ライフサイクル・モデルを確立すべきである。選択するソフトウェア・ライフサイクル・モデルは、ソフトウェアの生成から廃棄までを対象とするものであること。典型的なソフトウェア・ライフサイクル・モデルの活動には、下記のようなものがある。

life cycle model include the following: ● Quality Planning ● System Requirements Definition ● Detailed Software Requirements Specification ● Software Design Specification ● Construction or Coding ● Testing ● Installation ● Operation and Support ● Maintenance ● Retirement Verification, testing, and other tasks that support software validation occur during each of these activities. A life cycle model organizes these software development activities in various ways and provides a framework for monitoring and controlling the software development project. 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.	 ● 品質計画 ● システム要件の定義 ● 詳細なソフトウェア要求仕様書 ● ソフトウェア設計仕様書 ● 構築／コーディング ● テスト ● インストレーション ● 操作およびサポート ● メンテナンス ● 廃棄 これらの各活動の間に、ソフトウェア・バリデーションを裏づける検証、テスト等のタスクを行う。ライフサイクル・モデルは、これらソフトウェア開発活動を様々な方法で体系づけ、またソフトウェア開発プロジェクトを監視・管理するためのフレームワークを提供する。ソフトウェア・ライフサイクル・モデル(ウォーターフォール、スパイラル、ラピッド・プロトタイピング、インクリメンタル開発など)は、FDA が 1995 年 8 月に発行した、Glossary of Computerized System and Software Development Terminology に定義されている。このようなライフサイクル・モデルについては、付録 A に挙げた様々な参考資料に記載されている。

5.2. TYPICAL TASKS SUPPORTING VALIDATION

5.2. バリデーションを裏づける典型的なタスク

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	ソフトウェア・ライフサイクルにおける各活動には、ソフトウェアがバリデートされたという結論を裏づけるための「典型的な」タスクがある。しかし、実行されるタスク、実行の順序、反復およびタイミングは、選択したソフトウェア・ライフサイクル・モデル、およびそのソフトウェア・アプリケーションに関する安全性へのリスクにより決まる。リ

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.	スクの非常に低いアプリケーションについては、ある種のタスクは全く必要ないことも多い。しかし、ソフトウェア開発者は、少なくともこのような各タスクを考慮した上で、そのアプリケーションに、どのタスクが適切または不適切であるかを定義し、文書化する。下記に一般的な検討項目を示したが、特定のソフトウェア・ライフサイクル・モデルや、タスクを行う際の特定の順序を定めるものではない。

5.2.1. Quality Planning

5.2.1. 品質計画

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:	設計および開発計画が最終的に、以下を確認する計画となっていること。 ● 必要なタスク ● 異常に関する報告と解決の手順 ● 必要なリソース ● 公式な設計レビューを含めた管理レビュー要件 ソフトウェア・ライフサイクル・モデル、および関連する活動を、ソフトウェア・ライフサイクルにおける各活動に必要なタスクと併せて確認する。計画には、以下のものを全て含めること。
● The specific tasks for each life cycle activity; ● Enumeration of important quality factors (e.g., reliability, maintainability, and usability); ● Methods and procedures for each task; ● Task acceptance criteria; ● Criteria for defining and documenting outputs in terms that will allow evaluation of their conformance to input requirements;	● ライフサイクルの各活動における具体的なタスク ● 重要な品質要因のリスト(信頼性、保守性、利便性など)。 ● 各タスクの方法および手順 ● タスクの合格基準 ● 入力となる要件を満足しているか評価できる出力を定義し、文書化するための基準 ● 各タスクへの入力 ● 各タスクからの出力

● Inputs for each task; ● Outputs from each task; ● Roles, resources, and responsibilities for each task; ● Risks and assumptions; and ● Documentation of user needs.	● 各タスクにおける役割、リソースおよび責任 ● リスクおよび前提条件 ● ユーザのニーズに関する文書
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.	管理者は、適切なソフトウェア開発環境とリソースを明確にして提供することが必須である。(See 21 CFR Section 820.20(b)(1)および(2)を参照のこと。)一般的に各タスクでは、物理的リソースと共に人が必要となる。計画では、各タスクに必要な人、施設および設備のリソース、また、リスク(ハザード)管理の果たす役割を明確にすべきである。複数の並行開発活動を導き、管理するために、また適切なコミュニケーションおよび文書化を確実にするために、コンフィギュレーション管理計画を設けるべきである。ソフトウェア・システムを構成する仕様文書、ソース・コード、オブジェクト・コードおよび様々なテストの承認された全てのバージョンの間には、明確で間違いのない対応を確実にするような管理が必要である。また、承認された現行のバージョンを正確に特定し、アクセスできるように管理すること。
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.	バリデーションや他の活動で発見されたソフトウェア異常を報告および解決するための手順を設けること。管理者は必要な報告書を明らかにし、その内容、形式、および各報告書の責任部署を定めること。また、ソフトウェア開発の結果のレビューおよび承認を行うための手順(このようなレビューおよび承認を行う責任部署の特定も含む)が必要である。

Typical Tasks – Quality Planning	典型的なタスクー品質計画
● Risk (Hazard) Management Plan ● Configuration Management Plan ● Software Quality Assurance Plan – Software Verification and Validation Plan ▫ Verification and Validation Tasks, and Acceptance Criteria ▫ Schedule and Resource Allocation (for software verification and validation activities) ▫ Reporting Requirements – Formal Design Review Requirements – Other Technical Review Requirements ● Problem Reporting and Resolution Procedures ● Other Support Activities	● リスク(ハザード)管理計画 ● コンフィギュレーション管理計画 ● ソフトウェア品質保証計画 – ソフトウェアの検証／バリデーション計画 ▫ 検証／バリデーションの作業、および合格基準 ▫ (ソフトウェアの検証／バリデーション作業の)スケジュールとリソースの割付 ▫ 要件の報告 – 公式な設計レビュー要件 – その他の技術的レビュー要件 ● 問題の報告および解決の手順 ● その他のサポート活動

5.2.2. Requirements

5.2.2. 要件

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.	要件の作成に含まれるのは、医療機器およびその使用目的に関する情報の明確化、分析、文書化である。システム機能のハードウェア／ソフトウェアへの割り当て、動作条件、ユーザの特徴、潜在的ハザード、および予想されるタスク等が特に重要である。また、要件では、そのソフトウェアの使用目的を明確に述べること。
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:	ソフトウェア要求仕様書には、ソフトウェア機能の定義の記述を含めること。ソフトウェア要件を事前に決定して文書化しなければ、ソフトウェアをバリデートすることは不可能である。典型的なソフトウェア要件では、以下の点を明記する。

● All software system inputs; ● All software system outputs; ● All functions that the software system will perform; ● All performance requirements that the software will meet, (e.g., data throughput, reliability, and timing); ● The definition of all external and user interfaces, as well as any internal software-to-system interfaces; ● How users will interact with the system; ● What constitutes an error and how errors should be handled; ● Required response times; ● The intended operating environment for the software, if this is a design constraint (e.g., hardware platform, operating system); ● All ranges, limits, defaults, and specific values that the software will accept; and ● All safety related requirements, specifications, features, or functions that will be implemented in software.	● ソフトウェア・システムの全入力 ● ソフトウェア・システムの全出力 ● ソフトウェア・システムで実行される全機能 ● ソフトウェアが満たす全てのパフォーマンス要件(データ処理能力、信頼性、タイミングなど) ● 外部インターフェースおよびユーザ・インターフェースの定義。また、ソフトウェア~システム間の内部インターフェースの定義 ● ユーザとシステムとのインタラクション方法 ● エラーの内容および対処方法 ● 必要な応答時間 ● ソフトウェアの動作環境が、設計上の制約となる場合(ハードウェアのプラットフォーム、オペレーティング・システムなど)。 ● ソフトウェアが受け付けるデータ範囲、限界、規定値、および特定の値 ● ソフトウェアで実装される安全性に関する全ての要件、仕様、特徴、または機能
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	ソフトウェアの安全性の要件は、システム要件作成プロセスに密接に統合された技術的リスク管理プロセスから得られる。ソフトウェア要求仕様書では、ソフトウェアで実装される安全性の要件と共に、システムでソフトウェア・フェイルが引き起こす潜在的なハザードについて、明らかにすること。ソフトウェア・フェイルの引き起こす結果について、ハザードを緩和する手段(ハードウェアによる緩和、防御的プログラミングなど)と共に評価しておくこと。この分析を行えば、被害を防止するために必要な、最も適切な手段を明らかにすることができる。

harm. The Quality System regulation requires a mechanism for addressing incomplete, ambiguous, or conflicting requirements. (See 21 CFR 820.30(c).) Each requirement (e.g., hardware, software, user, operator interface, and safety) identified in the software requirements specification should be evaluated for accuracy, completeness, consistency, testability, correctness, and clarity. For example, software requirements should be evaluated to verify that: ● There are no internal inconsistencies among requirements; ● All of the performance requirements for the system have been spelled out; ● Fault tolerance, safety, and security requirements are complete and correct; ● Allocation of software functions is accurate and complete; ● Software requirements are appropriate for the system hazards; and ● All requirements are expressed in terms that are measurable or objectively verifiable. A software requirements traceability analysis should be conducted to trace software requirements to (and from) system requirements and to risk analysis results. In addition to any other analyses and documentation used to verify software requirements, a formal design review is recommended to confirm that requirements are fully specified and appropriate before extensive software design efforts begin. Requirements can be approved and released incrementally, but care	品質システム規制では、不完全、不明確、または矛盾した要件に対応する仕組が要求される。(21 CFR 820.30(c)を参照のこと。)ソフトウェア要求仕様書で確認された各要件(ハードウェア、ソフトウェア、ユーザ、オペレータ・インターフェース、および安全性など)を、精密性、完全性、一貫性、テスト性、正確性および明瞭性について、評価すること。例えば、ソフトウェア要件は下記を検証することを目的とし、評価すること。 ● 要件同士が内部で矛盾しない ● システムのパフォーマンス要件が全て記述されている ● 耐障害性、安全性、およびセキュリティの要件が完全かつ正確である ● ソフトウェア機能の割付が、正確かつ完全である ● ソフトウェア要件が、システムのハザードに対して適切である ● 全ての要件が、計測可能であるか、または客観的に検証できるように明確に表現されている システム要件やリスク分析結果に対して(および、システム要件から)ソフトウェア要件を辿れるように、ソフトウェア要件をトレーサビリティ分析すること。さらに、ソフトウェア要件の検証のために行われる他の分析および文書化に加えて、公式な設計レビューを実施し、ソフトウェア設計に大きな労力をかける前に、要件が全て仕様書に記され、適切であるということを確認しておくことを推奨する。要件は逐次追加的に承認してレリースできるが、ソフトウェア(およびハードウェア)要件間の相互作用およびインターフェースのレビュー、分析、管理を適切に行うよう、注

should be taken that interactions and interfaces among software (and hardware) requirements are properly reviewed, analyzed, and controlled. Typical Tasks – Requirements ● Preliminary Risk Analysis ● Traceability Analysis – Software Requirements to System Requirements (and vice versa) – Software Requirements to Risk Analysis ● Description of User Characteristics ● Listing of Characteristics and Limitations of Primary and Secondary Memory ● Software Requirements Evaluation ● Software User Interface Requirements Analysis ● System Test Plan Generation ● Acceptance Test Plan Generation ● Ambiguity Review or Analysis	意を要する。 典型的なタスク-要件 ● 事前のリスク分析 ● トレーサビリティ分析 – システム要件とソフトウェア要件 – リスク分析とソフトウェア要件 ● ユーザの特徴に関する記述 ● 主メモリおよび補助メモリの特徴および制限事項のリスト ● ソフトウェア要件の評価 ● ソフトウェアのユーザ・インターフェースの要件分析 ● システムのテスト計画立案 ● 受入テスト計画立案 ● 曖昧な点のレビューまたは分析

5.2.3. Design

5.2.3. 設計

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	ソフトウェア要求仕様は設計プロセスにおいて、実装されるソフトウェアの論理的表現および物理的表現に翻訳される。ソフトウェアの設計仕様には、そのソフトウェアが何をどのように実行するかを記述してある。プロジェクトの複雑性に応じるため、または様々なレベルの技術的な責任者が、設計情報を明確に理解できるようにするため、設計仕様に、設計のハイレベルな概要と詳細な設計情報の両方を含めてもよい。完成されたソフトウェア設計仕様は、プログラマを、合意された要件および設計の目的から外れないようにする。ソフトウェア設計仕様が完全なものであれば、プログラマは、設計上の判断をその場その場で行う必要がなくなる。

agreed upon requirements and design. A complete software design specification will relieve the programmer from the need to make ad hoc design decisions.	
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.	ソフトウェアの設計は、人的要因に対応したものとする必要がある。過度に複雑な設計、または操作がユーザの直感的な予想に反するために引き起こされる利用上のエラーは、FDA が見てきた問題の中で最も絶え間なく起こり、かつ重大なものの一つである。ソフトウェアの設計が、このような利用上のエラーの要因となっていることがよく見られる。医療機器の設計要件、分析、テストを含めた設計および開発のプロセス全体で、人間工学的な考え方を採用すること。フローチャートや状態図、プロトタイピング・ツール、テスト計画を作成する際には、医療機器の安全性および利便性の問題を考慮すること。また、タスクや機能分析、リスク分析、プロトタイプ・テスト／レビュー、および十分な使用性テストを行うこと。ユーザ側の参加者を含め、この方法を採用すること。
The software design specification should include: ● Software requirements specification, including predetermined criteria for acceptance of the software; ● Software risk analysis; ● Development procedures and coding guidelines (or other programming procedures); ● Systems documentation (e.g., a narrative or a context diagram) that describes the systems context in which the program is intended to	ソフトウェアの設計仕様には、下記の事項を含めること。 ● ソフトウェア要求仕様（予め定義された合格基準を含む） ● ソフトウェアのリスク分析 ● 開発手順およびコーディングのガイドライン(または、その他のプログラミング手順) ● （文章またはコンテキスト図など)で、プログラムが動作するシステムのコンテキストを、ハードウェア、ソフトウェア、および物理的環境間の関連を含めて記述したシステム文書 ● 使用するハードウェア

function, including the relationship of hardware, software, and the physical environment; ● Hardware to be used; ● Parameters to be measured or recorded; ● Logical structure (including control logic) and logical processing steps (e.g., algorithms); ● Data structures and data flow diagrams; ● Definitions of variables (control and data) and description of where they are used; ● Error, alarm, and warning messages; ● Supporting software (e.g., operating systems, drivers, other application software); ● Communication links (links among internal modules of the software, links with the supporting software, links with the hardware, and links with the user); ● Security measures (both physical and logical security); and ● Any additional constraints not identified in the above elements.	● 測定または記録するパラメータ ● (制御ロジックを含む)論理構造および論理的プロセス・ステップ(アルゴリズムなど) ● データ構造およびデータ・フロー・ダイアグラム ● 変数(制御とデータ)の定義およびそれらの使用箇所に関する記述 ● エラー、アラームおよび警告メッセージ ● サポート・ソフトウェア(オペレーティング・システム、ドライバ、その他のアプリケーション・ソフトウェアなど) ● 通信リンク(ソフトウェアの内部モジュール間のリンク、対応ソフトウェアとのリンク、ハードウェアとのリンク、ユーザとのリンク) ● セキュリティ手段(物理的および論理的セキュリティ) ● 上記以外の付加的な制限事項
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	上記に示したうち最初の四つの要素は、別途作成済みの文書に記載されていることが多く、ソフトウェア設計仕様書からはこれらの文書を参照する。ソフトウェア要求仕様およびソフトウェアのリスク分析については、前のセクションで述べた。開発手順書は、企業がガイドとして用いることができるものであり、プログラミング手順書は、個々のプログラマのガイドとなる。ソフトウェアがどのようなコンテキストで動作するかわからないとバリデートできないため、システム文書を参照する。上記の要素のうち、ソフトウェアに含まれないものがある場合は、その旨(例:本プログラムにはエラー・メッセージを出さない)を記載しておけば、ソフトウェアのレビュー／メンテナンスを行う担当者の役に立つであろう。

and maintainers of the software if that is clearly stated (e.g., There are no error messages in this program).	
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.	ソフトウェア設計で実施する活動には、いくつかの目的がある。ソフトウェアの設計評価は、設計に関する以下の判断が目的である。 ● 完全な設計であるか ● 正確であるか ● 一貫性があるか ● 曖昧さはないか ● 実行可能かどうか ● メンテナンスが可能か 設計中に、ソフトウェアのアーキテクチャ(モジュール構造など)を適切に考慮しておけば、将来ソフトウェアの変更が必要になった際、バリデーションの労力を軽減することができる。ソフトウェア設計の評価には、制御フロー、データ・フロー、複雑性、タイミング、サイジング、メモリ・アロケーション、致命度解析や、他にも多くの設計の側面の分析が含まれることが多い。ソフトウェアの設計で、ソフトウェア要件が全て実装されていることを検証するため、トレーサビリティ分析を行うこと。ソフトウェア要件が不十分な箇所を特定するための手段としてトレーサビリティ分析を実施すると、設計の全ての側面がソフトウェア要件に対してトレースできるかどうかを検証できる。コミュニケーション・リンクの分析を行うことにより、ハードウェア、ユーザおよび関連のソフトウェア要件に対して、提案された設計を評価すること。設計上、さらに危険が発見されたかどうか、また、新たなハザードが作り込まれたかどうかを判断するため、ソフトウェアのリスク分析を再検討すること。
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	ソフトウェアの設計活動の最後で、設計の実装に移る前に、設計が正確で一貫しており、テスト可能であることを検証するため、公式な設計レビューを行うこと。ソフトウェア設計は部分的に、完成の都度、承認してレリースする

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.	こともできるが、様々な要素間での相互作用およびコミュニケーション・リンクのレビュー・分析・制御が適切になるよう注意を要する。
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.	ほとんどのソフトウェア開発モデルは、反復して行うものである。このため、ソフトウェア要求仕様書および設計仕様書には、いくつかのバージョンができることが多い。承認された全てのバージョンは、確立されたコンフィギュレーション管理手順に従って、アーカイブおよび管理すること。
Typical Tasks – Design ● Updated Software Risk Analysis ● Traceability Analysis - Design Specification to Software Requirements (and vice versa) ● Software Design Evaluation ● Design Communication Link Analysis ● Module Test Plan Generation ● Integration Test Plan Generation ● Test Design Generation (module, integration, system, and acceptance)	典型的なタスク-設計 ● ソフトウェアのリスク分析の更新 ● ソフトウェア要件と設計仕様書の間のトレーサビリティ分析 ● ソフトウェアの設計評価 ● 設計のコミュニケーション・リンク分析 ● モジュールのテスト計画立案 ● 統合テスト計画の立案 ● テスト計画の立案(モジュール、統合、システム、および受入テスト)

5.2.4. Construction or Coding

5.2.4. 構築／コーディング

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	ソフトウェアの構築は、コーディング(プログラミングなど)によるもの、または、すでにコード化されたソフトウェア部品(コード・ライブラリや市販のソフトウェアなど)を、新しいアプリケーションで使用するために組み立てたものが多い。コーディングとは、設計仕様の詳細をソース・コードとして実装する、ソフトウェア製作活動である。コーディング

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.	は、ソフトウェア開発プロセスにおいて最も抽象度が低い。これは、ソフトウェア要件の分解の最終段階であり、ここでモジュール仕様をプログラミング言語に翻訳する。
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	コーディングでは通常、高度のプログラミング言語を使用するが、スピードを要する操作には、アセンブリ言語(またはマイクロコード)の使用が必要となる場合もある。ソース・コードとは、対象のハードウェア・プラットフォームで使用するために、コンパイルまたはインタプリトされたものである。プログラム言語およびソフトウェア・ビルド・ツール(アセンブラ、リンカ、およびコンパイラ)を選択する際は、後の品質評価タスクに対する影響を考慮すること(選択した言語でデバッギングおよびテスト・ツールが利用できるかどうか、など)。コンパイラの中には、コードのデバッギングを支援するエラー・チェックのレベルおよびコマンドを選択できるものがある。コーディングのプロセスを通じて異なるレベルのエラー・チェックを用いるかもしれないし、コンパイラの警告などのメッセージを記録に残す場合も、残さない場合もある。しかし、コーディングおよびデバッグのプロセス終了時には、通常、最も厳密なレベルのエラー・チェックを行い、ソフトウェアに残ったコンパイルのエラーを記録する。最終的なソース・コードの翻訳に、最も厳密なレベルのエラー・チェックを用いない場合には、その理由を文書化しておくこと。また、最終コンパイルとして、コンパイラの警告や他のメッセージ、その解決方法、またはイッシュを未解決にしておくという判断理由も含めて、コンパイル・プロセスおよびその結果を文書として残すこと。

issues unresolved.	
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.	ソフトウェアのコーディング・プロセスに関する品質方針および手順を確立するコーディング・ガイドラインを採用する企業が多い。ソース・コードが、そのコーディング・ガイドラインに適合しているかを検証するための評価を行うこと。このようなガイドラインには、分かりやすさ、スタイル、複雑性の管理、コメントに関するコーディングの決まりごと(conventions)を含めること。コードのコメントには、予想される入力および出力、参照される変数、データ型、操作などを含めたモジュールに関する実用的な説明的な情報を記載すること。またソース・コードについては、対応する詳細設計仕様への適合を検証するための評価を行うこと。統合、テストの準備ができたモジュールについては、コーディングのガイドライン等の、適用される品質方針および手順への適合に関する文書を作成しておくこと。
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	ソース・コードの評価は、コード検査(inspection)、およびコードのウォーク・スルーで行う場合が多い。このような静的解析は、コード実行前にエラーを検出する上で、非常に効果的な手段である。この方法を採用すると、各エラーを個別に検討することができる。また、後のソフトウェアの動的テストの焦点が絞りやすくなる。企業では、一貫性および独立性を確実にするための適切な管理のもとで、手動(机上)チェックを実施する場合もある。ソース・コード評価をさらに広げて、モジュール間、レイヤー間の内部リンケージ(水平および垂直インターフェース)、また設計仕様に対する適合についても検証すること。ソース・コードの評価に用いた手順および結果の文書は、設計検証の一部として保存しておくこと。

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: ● Each element of the software design specification has been implemented in code; ● Modules and functions implemented in code can be traced back to an element in the software design specification and to the risk analysis; ● Tests for modules and functions can be traced back to an element in the software design specification and to the risk analysis; and ● Tests for modules and functions can be traced to source code for the same modules and functions. <u>Typical Tasks – Construction or Coding</u> ● Traceability Analyses – Source Code to Design Specification (and vice versa) – Test Cases to Source Code and to Design Specification ● Source Code and Source Code Documentation Evaluation ● Source Code Interface Analysis ● Test Procedure and Test Case Generation (module, integration, system, and acceptance)	ソース・コードのトレーサビリティ分析は、全てのコードが、確立された仕様書およびテストの手順に関連付けられているかどうかを検証する、重要なツールである。ソース・コードのトレーサビリティ分析は、下記を検証し文書化する目的で行うこと。 ● ソフトウェア設計仕様の各要素がコード化されている ● コード化したモジュールおよび機能が、ソフトウェアの設計仕様およびリスク分析の要素にトレースできる ● モジュールおよび機能に対するテストが、ソフトウェアの設計仕様書およびリスク分析の要素にトレースできる ● モジュールおよび機能に対するテストが、そのモジュールと機能のソース・コードにトレースできる <u>典型的なタスクー構築／コーディング</u> ● トレーサビリティ分析 – ソース・コードと設計仕様との関連 – テスト・ケースの、ソース・コードおよび設計仕様に対する関連 ● ソース・コードおよびソース・コードの文書評価 ● ソース・コードのインターフェース分析 ● テスト手順およびテスト・ケースの作成(モジュール、統合、システムおよび受入テスト)

5.2.5. Testing by the Software Developer

5.2.5. ソフトウェア開発者による試験

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.	ソフトウェアのテストでは、予め定めた予想結果と比較ができるように、定義された入力と文書化された出力を定めた既知の条件下で、ソフトウェア製品を動作させてみることが必要である。これは時間がかかり、難しく、また不完全なものである。従って、効果的かつ能率的に行うためには、早期に計画することが必要になる。
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: ● NIST Special Publication 500-235, *Structured Testing: A Testing Methodology Using the Cyclomatic Complexity Metric*; ● NUREG/CR-6293, *Verification and Validation Guidelines for High Integrity Systems*; and ● IEEE Computer Society Press, *Handbook of Software Reliability Engineering*.	テスト計画およびテスト・ケースは、ソフトウェアの開発プロセスの中で、可能な限り早期の段階に作成し、下記の項目を明らかにすること。 ● スケジュール ● 環境、リソース(人、ツールなど) ● 方法論、ケース(入力、手順、出力、予想結果) ● 文書報告基準 テスト・プロセス全体にかかる労力の大きさは、複雑性、重大性、信頼性、および安全性の課題に関連する(例:重大な結果をもたらす機能またはモジュールについて、フォールト・トレランス機能を徹底的にテストして検証することを要求する)ソフトウェア・カテゴリの説明、およびソフトウェアのテストにかかる労力については、下記のような文献に記載されている。 (文献名:左記参照)
Software test plans should identify the particular tasks to be conducted at each stage of	ソフトウェアのテスト計画では、開発の各段階に特有のタスクを明確にし、また各タスクの完了基準に応じた労力

development and include justification of the level of effort represented by their corresponding completion criteria.	のレベルの根拠を記載すること。
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.	ソフトウェア製品のテストを計画する際には、ソフトウェア・テストには、限界があることを認識し、考慮に入れなければならない。よほど単純なプログラムでない限り、ソフトウェアを余す所なくテストし尽くすということはできない。一般的に、考えられる全ての入力を行ってソフトウェア製品をテストすることは現実的ではなく、プログラムの実行中に生じる可能なデータ処理のパスを全てテストすることも不可能である。ソフトウェア製品を確実に徹底してテストできるようなテストの種類やテスト方法はない。プログラムの機能を全てテストしても、プログラムが全てテストされたということにはならない。あるプログラム・コードを全てテストしても、必要な機能が全てプログラムに組み込まれているとは限らない。プログラムの全ての機能、および全てのプログラム・コードをテストしても、プログラムが 100%正確であるとは限らないのである！ソフトウェアのテストでエラーが発見されなかった場合でも、そのソフトウェアにエラーがないと考えるべきではない。テストが表面的だったという場合もあるからである。
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	ソフトウェアのテスト・ケースでは、結果を予想することが重要な要素となる。実際のテスト結果の客観的評価を可能にするのは、キーとなる細部である。この必要なテスト情報は、対応する、事前に定められた定義または仕様から得られるものである。ソフトウェア仕様文書では、テストを通じて確認できるよう、エンジニアリング(例：測定可能、または客観的検証が可能)レベルの詳細さで、何を、いつ、どのように、何のために達成することになっているのか、などを明確にしなければならない。ソフトウェア・テストを有効なものとする上での実際の労力がかかるの

effective software testing lies in the definition of what is to be tested rather than in the performance of the test.	は、テストの実施よりも、何をテストするかという定義するときである。
A software testing process should be based on principles that foster effective examinations of a software product. Applicable software testing tenets include: ● The expected test outcome is predefined; ● A good test case has a high probability of exposing an error; ● A successful test is one that finds an error; ● There is independence from coding; ● Both application (user) and software (programming) expertise are employed; ● Testers use different tools from coders; ● Examining only the usual case is insufficient; ● Test documentation permits its reuse and an independent confirmation of the pass/fail status of a test outcome during subsequent review.	ソフトウェアのテスト・プロセスは、ソフトウェア製品の検査を効果的にするような原則に基づいたものにすること。ソフトウェア・テストに適用できる原則には、次のようなものがある。 ● テスト結果の予想をあらかじめ立てておく ● よいテスト・ケースは、エラーを発見する可能性が高い ● エラーを発見できれば、テストは成功である ● コーディングから独立している ● アプリケーション(ユーザ)とソフトウェア(プログラミング)の両方の専門技術を用いている ● テスト担当者は、プログラマとは違うツールを使用する ● 通常のケースを検討するだけでは不十分である ● テスト文書が再利用でき、その後のレビューでテスト結果の合否状況が第三者でも確認できるようになっている
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.	事前に必要なタスク(コード検証など)が無事終了した後、ソフトウェアのテストを開始する。テストはユニット・レベルで開始し、システム・レベルのテストで終了する。統合レベルのテストを実施する場合もある。ソフトウェア製品は、内部構造に基づくテスト・ケース、および外部仕様に基づくテスト・ケースで検査すること。これらのテストにより、ソフトウェア製品が機能、性能、インターフェースの定義および要件に適合しているかどうかを、徹底的かつ厳密に検査することができる。

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.	コードに基づくテストは構造テスト、または「ホワイト・ボックス」テストと呼ばれる。これは、ソース・コードや設計仕様の詳細や他の開発文書から得た知識を基に、テスト・ケースを明確にするものである。これらのテスト・ケースにより、プログラムによる制御の判断、およびコンフィギュレーション・テーブルを含むプログラムのデータ構造がテストされる。構造テストでは、プログラムを実行した時に全く実行されないデッド・コードを見つけることができる。構造テストは、主にユニット(モジュール)レベルのテストで完了するが、拡大して他のレベルのソフトウェアのテストを行うこともできる。
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:	構造テストのレベルは、ソフトウェア構造の何パーセントが構造テスト中に評価されたかを示すような測定基準を用いて評価できる。この測定基準は、一般的に「カバレージ」と呼ばれるものであり、テストの選択基準に関する完成度の尺度である。構造テスト・カバレージの量は、ソフトウェアが持つリスクのレベルに相応のものとすること。「カバレージ」という用語を使用する場合は、通常、カバレージが 100%である場合を意味する。例えば、テスト・プログラムが「ステートメント・カバレージ」を完了したという場合には、ソフトウェアのステートメントを少なくとも 1 回は、100%実行したということになる。一般的な構造カバレージの測定基準には、以下のようなものがある。
● **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.	● **ステートメント・カバレージ** – この基準では、各プログラムのステートメントを少なくとも一度実行するための、十分なテスト・ケースが必要となる。しかしこれだけでは、ソフトウェア製品の動作の信頼性を証明するのに不十分である。

● **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.	● **判定(ブランチ)カバレージ** – この基準では、各プログラムの判定またはブランチを実行して、起こりうる結果を少なくとも 1 回発生させるための十分なテスト・ケースが必要となる。これは、大半のソフトウェア製品において最低限のレベルとされるカバレージだが、判定カバレージだけでは、高い品質を要求されるアプリケーションには不十分である。
● **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.	● **コンディション・カバレージ** – この基準では、プログラム判定の各条件について、起こりうる結果全てを少なくとも 1 回通るための十分なテスト・ケースが必要となる。ブランチ・カバレージと異なるのは、判定に至る前に複数の条件の評価が必須となっている場合のみである。
● **Multi-Condition Coverage** – This criteria requires sufficient test cases to exercise all possible combinations of conditions in a program decision.	● **マルチ・コンディション・カバレージ** – この基準では、プログラム判定条件について、可能な組み合わせ全てを実行するための十分なテスト・ケースが必要となる。
● **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.	● **ループ・カバレージ** – この基準では、プログラムの全ループを、0 回、1 回、2 回、複数回反復し、初期化、典型的な実行、および終了(境界)条件を網羅するための十分なテスト・ケースが必要となる。
● **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	● **パス・カバレージ** – この基準では、実行可能なそれぞれのパスや基本パスなどを、定義されたプログラム・セグメントの開始から終了まで、少なくとも 1 回実行するための十分なテスト・ケースが必要となる。ソフトウェア・プログラムで可能なパスは非常に数多くあるため、パス・カバレージを完璧に行うことは、一般的には不可能である。パス・カバレージの量は通

amount of path coverage is normally established based on the risk or criticality of the software under test.	常、テスト中のソフトウェアのリスクまたは危険度に基づいて決定される。
● **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.	● **データ・フロー・カバレージ** － この基準では、実行可能な各データ・フローを少なくとも一度実行するための十分なテスト・ケースが必要となる。データ・フローのテスト方法は、数多くある。
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.	定義または仕様書に基づくテストは、機能テストまたは「ブラック・ボックス」テストと呼ばれる。これは、ソフトウェア製品(それがユニット(モジュール)であっても、完成したプログラムであっても)の使用目的の定義に基づいて、テスト・ケースを明らかにするものである。これらのテスト・ケースは、プログラムの使用目的または機能、およびプログラムの内部／外部インターフェースをテストするものである。機能テストはユニット・レベルからシステム・レベルのテストまで、全てのレベルのソフトウェア・テストで用いることができる。
The following types of functional software testing involve generally increasing levels of effort:	下記に挙げる種類のソフトウェアの機能テストでは、一般的に、下にいくほど労力レベルが上がる。
● **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.	● **正常な場合** － 通常の入力によるテストが必要である。しかし、ソフトウェア製品を予想される妥当な入力だけでテストしても、そのソフトウェア製品を完全にテストしたことにはならない。正常な場合のテストのみでは、ソフトウェア製品の信頼性についての十分な確信を得ることはできない。
● **Output Forcing** – Choosing test inputs to ensure that selected (or all) software outputs are generated by testing.	● **アウトプット・フォーシング** － 選択した(または全ての)ソフトウェア出力をテストで生成できるかどうかを確認するために、テストの入力を選ぶ。
● **Robustness** – Software testing should	● **ロバストネス** － ソフトウェアのテストは、ソフトウェ

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.	ア製品が予想外の無効な入力を与えた際にも、正しく動作するということを立証するものでなければならない。このような十分なテスト・ケースを特定する方法には、境界値分析(Equivalence Class Partitioning)、Boundary Value Analysis、Special Case Identification(Error guessing)などがある。これらの方法は重要かつ必要なものであるが、ソフトウェア製品に最適なテスト項目が洗い出されたことを保障するものではない。
● **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.	● **入力の組み合わせ** － 上記に挙げた機能テストの方法は、個別または単独のテスト入力を重視したものである。大半のソフトウェア製品は、その製品の使用条件下において、複数の入力で動作する。ソフトウェア製品を徹底的にテストする場合、ソフトウェアのユニットまたはシステムの動作中に遭遇しうる、入力の組み合わせを考慮すべきである。Error guessing を拡大して、入力の組み合わせを挙げることもできるが、これは暫定的な技法である。原因と結果のグラフ化はソフトウェアの機能テストの一つであり、ソフトウェア製品への入力の組み合わせを、テスト・ケースに含めるために、体系的に明確にするものである。
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	ソフトウェアの機能テスト・ケースおよび構造テスト・ケースを洗い出すための技法により、でたらめなテスト入力ではなく、テストに用いる入力を得ることができる。このような技法における一つの弱点は、構造テストと機能テストの完了基準をソフトウェア製品の信頼度に結びつけることが困難だということである。ソフトウェア製品の信頼性をさらに確実にすべく、統計テストのような高度なソフトウェア・テスト技法が用いられることもある。統計テストでは、動作上の特徴(ソフトウェア製品の期待される使用、

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.	危険な使用、または故意の悪意を持った使用など)に基づいて定義された分布に従って、無作為に生成されるテスト・データを用いる。膨大な量のテスト・データが生成され、特定の領域または懸念事項をカバーすることができる。このようなデータから、ソフトウェア製品の設計担当者にもテスト担当者も予測していなかった個別または複数の、稀な動作条件を明らかにすることができる可能性がある。統計テストにより、構造カバレージも高くなる。これには、安定したソフトウェア製品が必要となる。従って、ソフトウェア製品の統計テストを行う場合には、構造テストおよび機能テストを実施しておくことが必須となる。
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	ソフトウェア・テストのもう一つの側面とは、ソフトウェアの変更をテストすることである。ソフトウェアの開発中には、変更が頻繁に行われる。このような変更は、次のような事柄の結果として発生するものである。 1) エラーの発見と修正のデバッグ 2) 新規の要件または要件の変更(「要件のクリープ」) 3) より効果的または効率的な方法を発見したことによる設計の修正 ソフトウェア製品のベースラインが決定されると(承認されると)、その製品に対する変更はいかなるものであっても、テストを含む独自の「ミニ・ライフサイクル」を持つようになる。変更したソフトウェア製品のテストは、追加作業となる。変更が正確に実装されていることを立証するだけではなく、その変更によってソフトウェア製品の他の部分に悪影響が出ていないということを、テストでも証明すること。リグレッション分析およびリグレッション・テストは、変更によってソフトウェア製品のほかの場所に問題が生じていないということを確認することが目的である。リグレッション分析では、実施すべきリグレッション・テストを明確にするため、関連する文書(ソフトウェア要求仕

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.	様、ソフトウェア設計仕様、ソース・コード、テスト計画、テスト・ケース、テスト・スクリプトなど)のレビューに基づいて、変更の影響度を判断する。リグレッション・テストでは、プログラムが以前に正確に動作したテスト・ケースを再実行して現在の結果を前の結果と比較し、ソフトウェアの変更によって生じた影響のうち意図されていなかったものを検出する。ソフトウェア製品を統合して作成する際にも、リグレッション分析およびリグレッション・テストを行うこと。これは、新規に統合したモジュールが、既に統合したモジュールに悪影響を及ぼしていないかを確認することが目的である。
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.	ソフトウェア製品を徹底して厳密に検査するためには、通常、開発テストを複数レベルに分ける。例えば、ソフトウェア製品のテストをユニット、統合、およびシステム・レベルのテストに分けることができる。
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.	1) ユニット(モジュールまたはコンポーネント)レベルのテストでは、サブ・プログラム機能を早期に検査し、また、システム・レベルでは見えない機能をテストで検査することに焦点を当てる。ユニット・テストでは、高品質のソフトウェア・ユニットが確実に、最終ソフトウェア製品に統合されるできるようにする。。
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	2) 統合レベルのテストでは、プログラムの内部と外部のインターフェース間の、データおよびコントロールのやりとりに焦点を当てる。外部インターフェースとは、(オペレーティング・システム・ソフトウェアなどの)

(including operating system software), system hardware, and the users and can be described as communications links.	他のソフトウェア、システムのハードウェア、およびユーザとのインターフェースであり、コミュニケーション・リンクとも呼ばれるものである。
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):	3) システム・レベルのテストでは、仕様書にある機能が全て存在し、ソフトウェア製品が信頼できるものであることを立証する。このテストでは、指定のオペレーティング・プラットフォーム上で、ソフトウェア製品の要件について、製作されたプログラムの機能および性能を検証する。システム・レベルのソフトウェアのテストでは、機能的な懸案事項、および医療機器・ソフトウェアの使用目的に関する以下の要素に対応する。
● Performance issues (e.g., response times, reliability measurements); ● Responses to stress conditions, e.g., behavior under maximum load, continuous use; ● Operation of internal and external security features; ● Effectiveness of recovery procedures, including disaster recovery; ● Usability; ● Compatibility with other software products; ● Behavior in each of the defined hardware configurations; and ● Accuracy of documentation.	● 性能のイッシュ(応答時間、信頼性測定など) ● ストレス状態での応答。最大負荷での動作、連続使用など ● 内部および外部セキュリティ機能の運用 ● 障害リカバリを含む、リカバリ手順の有効性 ● 操作性 ● 他のソフトウェア製品との互換性 ● 定義された各ハードウェア・コンフィギュレーションの動作 ● 文書の正確性
Control measures (e.g., a traceability analysis) should be used to ensure that the intended coverage is achieved.	管理方策(トレーサビリティ分析など)を用いて、意図したカバレージを確実に達成すること。
System level testing also exhibits the software product's behavior in the intended operating	システム・レベルのテストでは、意図した操作環境におけるソフトウェア製品の動作も明らかになる。このようなテ

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.	ストをどの場所で実施するかは、ソフトウェア開発者がターゲットとなる操作環境を作り出せるかに左右される。環境によっては、シミュレーションおよび／または(潜在的な)顧客の場所でのテストを利用してもよい。計画したシステム・レベルのテストをソフトウェア開発者が直接管理しない現場で実施する際に、テスト計画で、確実に意図したカバレージを達成し、適切な文書を用意するために必要な管理を明らかにすること。また、医療機器または医療機器のコンポーネントであるソフトウェア製品で、FDAの認可以前にヒトに使用されるものについては、ヒトを対象とするテストには、Investigational Device Exemption (IDE)または Institutional Review Board (IRB)の承認が必要となるかもしれない。
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.	テスト手順、テスト・データおよびテスト結果は、客観的な合否判断を下すことが可能となる方法で文書化すること。また、テスト実施後のレビューおよび客観的判断、また後のリグレッション・テストでの使用にも適したものとすること。テスト中にエラーを発見したときはログを作成し、分類、レビューを行って、ソフトウェアのレリース前に解決すること。開発ライフサイクル中に収集および分析したソフトウェアのエラー・データは、ソフトウェアが市場での発売に適しているかどうかの判断に用いてもよい。テスト・レポートは、対応するテスト計画の要件に適合させること。
Software products that perform useful functions in medical devices or their production are often	医療機器、または医療機器の製造に役立つ機能を持つソフトウェア製品は、複雑である場合が多い。このような

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).	ソフトウェア製品のテストを実施する際にテストが一貫性があり、徹底的に実施され、かつ有効であることを確実にし、また、計画したテスト作業の要件を満たすために、ソフトウェアのテスト・ツールを用いてもよい。これらのツールには、市販のソフトウェア・テスト・ツールのほかに、ユニット(モジュール)・テストおよびその後の統合テストをスムーズに動作させるために、社内で作成したサポート・ソフトウェア(ドライバやスタブなど)が含まれる。このようなツールの品質は、テスト対象となるソフトウェア製品に劣らないものとすること。これらのソフトウェア・ツールを使用目的に対してバリデートしたという証拠を示す適切な文書を維持しておくこと(本ガイダンスの Section 6を参照のこと)。
Typical Tasks – Testing by the Software Developer	典型的なタスク ー ソフトウェア開発者によるテスト
● Test Planning ● Structural Test Case Identification ● Functional Test Case Identification ● Traceability Analysis - Testing – Unit (Module) Tests to Detailed Design – Integration Tests to High Level Design – System Tests to Software Requirements ● Unit (Module) Test Execution ● Integration Test Execution ● Functional Test Execution ● System Test Execution ● Acceptance Test Execution ● Test Results Evaluation ● Error Evaluation/Resolution ● Final Test Report	● テスト計画 ● 構造テスト・ケース洗い出し ● 機能テスト・ケース洗い出し ● トレーサビリティ分析およびトレーサビリティ・テスト – ユニット(モジュール)テスト→詳細設計 – 統合テスト→概要レベルの設計 – システム・テスト→ソフトウェア要件 ● ユニット(モジュール)テストの実行 ● 統合テストの実行 ● 機能テストの実行 ● システム・テストの実行 ● 受入テストの実行 ● テスト結果の評価 ● エラーの評価と解決 ● テストの最終報告書

5.2.6. User Site Testing

5.2.6. ユーザ・サイトのテスト

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.)	ユーザ・サイト・テストは、ソフトウェア・バリデーションの重要な部分である。品質システム規制では、(しかるべきテストを含めた)インストール手順および検査手順を求めている。また、インストールが正しく行われたことを立証するための、検査およびテストの文書も要求される。(21 CFR Section 820.170 を参照のこと。)同様に、製造機器においては指定された要件(specified requirements)へ適合することが必須となり、自動化システムでは使用目的に合わせたバリデーションを実施することが必須となる。(21 CFR Section 820.70(g)および 21 CFR Section 820.70(i)をそれぞれ参照のこと。)
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.	ユーザ・サイト・テストに関する用語は、紛らわしいものである。ユーザ・サイト・テストを表す語としては、ベータ・テストやサイト・バリデーション、ユーザ受入テスト、インストール検証、インストール・テストなどの用語が使われてきている。本ガイダンスでは、「ユーザ・サイト・テスト」という語は、上記、およびその他の開発者が管理する環境以外で行われる全てのテストを含むものとする。このテストは、インストールしたシステム構成の一部となる実際のハードウェアおよびソフトウェアを用いて、ユーザ・サイトで行う。このテストを行う際には、テスト対象のソフトウェアが機能する予定の環境で、実際にソフトウェアを使用するか、または模擬的に使用してみる。
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	本ガイダンスは一般的なものであり、どのようなユーザ・サイトのテストにも適用できる。しかし、ある領域(血液管理施設(blood establishment)用のシステムなど)では、ユーザ・サイト・テスト計画時に考慮すべき特別なサイト・バリデーションの課題がある。テスト計画の作成者は、ユーザ・サイト・テストを行う製品について、追加の規制要

Center(s) with the corresponding product jurisdiction to determine whether there are any additional regulatory requirements for user site testing.	件があるかどうかを FDA のセンターに確認すること。
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.	ユーザ・サイト・テストは、正式なテスト概要および正式な受入記録のある、事前に作成された計画書に従って行うこと。テスト手順、テスト入力データ、およびテスト結果の文書(テスト記録)は、全て保持しておくこと。
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.	ハードウェアおよびソフトウェアが、指定どおりにインストールされコンフィギュレーションされているという証拠がなければならない。テストを通じて全てのシステム部品が確実に実行され、また部品のバージョンが指定されたものであることを確実にする方策を講じること。テスト計画書には、動作環境全体をテストするよう明記すること。また、通常の活動では明らかにならない潜在的な欠陥を発見するため、様々な条件やイベントを発生させるためにテストを十分な時間継続することについても明記しておくこと。
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.	ソフトウェア開発者が開発サイトで行った評価のうちのいくつかは、ソフトウェアを実際に使用する現場で、再度行うこと。以下のテストなどが、これに該当する。 ● 高容量のデータ ● 高負荷やストレス ● セキュリティ ● 欠陥テスト(異常回避、異常検出、許容値、リカバリ) ● エラー・メッセージ ● セキュリティ要件の実行 この目的で、ソフトウェア開発者がユーザにテスト・データ・セットを提供する場合がある。

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.	意図した機能を適切に動作させられるかどうかというシステムの能力評価に加えて、ユーザがシステムを理解し、正しくシステムを運転できるかどうかという、ユーザの能力についても評価すること。オペレータは、意図された機能を実行できなければならない。また、アラームや警告、エラー・メッセージに対し、適切かつタイミングよく反応できなければならない。
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.	ユーザ・サイト・テスト中には、正常なシステム動作、および発生したシステムの異常についての記録を全て維持すること。ユーザ・サイト・テスト中に発見された欠陥を修復するようなシステムの改訂は、他のソフトウェア変更と同じ手順および管理法に従って行うこと。
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.	ソフトウェア開発者は、ユーザ・サイト・テストに参加する場合も、参加しない場合もある。開発者が参加する場合は、設計レベルのシステム・テストの最終部分を、継続してユーザ・サイト上で続行する。開発者がテストに参加しない場合に特に重要となるのは、入念なテスト計画やテストの予想結果の定義、全てのテスト出力を記録することの重要性を理解する者がユーザ側にいることである。
Typical Tasks – User Site Testing ● Acceptance Test Execution ● Test Results Evaluation ● Error Evaluation/Resolution ● Final Test Report	典型的なタスク ― ユーザ・サイト・テスト ● 受入テストの実行 ● テスト結果の評価 ● エラーの評価と解決 ● テストの最終レポート

5.2.7. Maintenance and Software Changes

5.2.7. メンテナンスとソフトウェアの変更

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.	メンテナンスという語をソフトウェアに用いる場合は、ハードウェアの場合とは、同じ意味にならない。ハードウェアとソフトウェアのメンテナンスは、フェイルやエラーの構造が違うため、異なったものとなる。ハードウェアのメンテナンスには、通常、予防的なハードウェア・メンテナンスのアクション、部品の交換、是正のための変更が含まれる。ソフトウェアのメンテナンスには、是正保守、完全化保守、および適応保守が含まれるが、予防保守活動(preventive maintenance actions)やソフトウェア部品の交換は含まれない。
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.	ソフトウェアのエラーおよび欠陥修正のための変更は、是正保守(corrective maintenance)である。ソフトウェアの性能や保守性、または他のソフトウェア・システムの特性を改良するための変更は、完全化保守(perfective maintenance)である。ソフトウェア・システムを変化した環境で使用できるようにするための変更は、適応保守(adaptive maintenance)である。
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).	最初の開発期間、またはレリース後のメンテナンス中にソフトウェア・システムを変更した場合には、十分なリグレッション分析およびリグレッション・テストを実施し、変更部分以外に悪影響が出ていないことを立証すること。これは、変更部分が正しいかを評価するテストに追加して行うものである。
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	ソフトウェア変更時に必要なバリデーションの労力は、変更の種類や影響の出る開発製品の種類、および、その製品がソフトウェアの操作に与える影響に従って決定される。設計構造、および様々なモジュールやインターフェ

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.	ースなどの相互関連に関する、注意深く書かれた完璧な文書があれば、変更が行われる際のバリデーションの労力を減らすことができる。変更を十分バリデートするために必要な労力は、最初のソフトウェア・バリデーションがどの程度文書化され、アーカイブされたかにも左右される。例えば、テスト文書やテスト・ケース、また以前に行った検証／バリデーション・テストの結果を後のリグレッション・テスト実施時に使用できるようにするためには、アーカイブしておく必要がある。この情報を後のためにアーカイブしておかなければ、変更後にソフトウェアの再バリデーションを行う際の労力と出費は、著しく増大することになる。
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:	標準的なソフトウェア開発プロセスにおけるソフトウェアの検証／バリデーションのタスクのほかに、以下に挙げる追加のメンテナンス・タスクを行うこと。
● **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.	● **ソフトウェア・バリデーション計画の改訂** － 以前にバリデートしたソフトウェアであれば、既存のソフトウェア・バリデーション計画を改訂し、修正したソフトウェアに対応させること。既存のソフトウェア・バリデーション計画が存在しない場合は、修正したソフトウェアのバリデーションに対応した計画を確立すること。
● **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	● **異常に関する評価** － ソフトウェア担当組織では、発見したソフトウェアの異常および各異常部に対する具体的な修正措置を記したソフトウェア・トラブル・レポートなどの文書を維持することが多い。しかし、ソフトウェア開発者が、問題の根本原因を判断し、問題の再発を回避するために必要なプロセスと手

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.)	順を変更するという「次のステップ」の実行を怠った結果ミスが繰り返されることも非常に多く見られる。ソフトウェアの異常は、その重大性およびシステム操作と安全性に対する影響という観点から評価すべきであるが、品質システムのプロセスにおける問題の兆候としても扱うべきである。異常の根本原因を分析すれば、品質システムの欠陥を明確にできる。(類似したソフトウェアの異常が再発しているなどの)傾向が特定できたら、品質に関する類似の異常がそれ以上再発しないように、適切な是正措置および予防措置を取り、文書化することが必須である。(21 CFR 820.100 を参照のこと。)
● **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.	● **問題点の特定および解決法の追跡記録** ― ソフトウェアのメンテナンス中に発見された問題点は、全て文書化しておくこと。各問題を確実に修正できるように、また参照用に資料化して問題発生の傾向を得ることができるよう、追跡し記録すること。
● **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.	● **提案された変更の評価** ― 全ての提案された修正、拡張、追加については、各変更によるシステム上の影響を判断するための評価を行うこと。このような情報により、検証／バリデーションのタスクを、どの程度繰り返し行うべきかが判断できるようになる。
● **Task Iteration** - For approved software changes, all necessary verification and validation tasks should be performed to	● **タスクの反復** ― ソフトウェアの変更で承認されたものについては、必要な検証／バリデーションのタスクを全て実施すること。これは、計画された変更が

ensure that planned changes are implemented correctly, all documentation is complete and up to date, and no unacceptable changes have occurred in software performance.	正しく実装されたか、全ての文書が完成し、最新版になっているかどうか、およびソフトウェアの動作に容認できない変化が起っていないかどうかを確認することが目的である。
● **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.	● **文書の更新** － 文書は入念にレビューを行い、変更によって、どの文書に影響が発生したかを判断すること。影響のあった文書で、承認されたもの(仕様書、テスト手順書、ユーザ・マニュアルなど)は全て、コンフィギュレーション管理手順に基づいて更新すること。仕様書については、メンテナンスおよびソフトウェアの変更を実施する前に更新すること。

SECTION 6. VALIDATION OF AUTOMATED PROCESS EQUIPMENT AND QUALITY SYSTEM SOFTWARE

SECTION 6. 自動プロセス機器と品質システム・ソフトウェアのバリデーション

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.	品質システム規制では、「コンピュータまたは自動データ処理システムが、製造または品質システムの一部として用いられている場合、(医療機器の)製造業者がコンピュータ・ソフトウェアをバリデートすること。バリデーションは、ソフトウェアの使用目的に関して確立されたプロトコロルに沿って行うこと」が要件とされている。(21 CFR Section 820.70(i)を参照のこと。)これは 1978 年以来、FDA の医療機器に関する Good Manufacturing Practice (GMP)の規制要件となっている。
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	上記のバリデーション要件に加えて、医療機器製造業者の製造プロセスまたは品質システムの一部を実装する(または、FDA の他の規制で要件となっている記録の作成および維持に用いる)コンピュータ・システムは、電子記録と電子署名の規制の対象となる。(21 CFR Part 11

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.	を参照のこと。)この規制は、電子的に記録を作成、維持する場合、セキュリティ、データの完全性、およびバリデーション要件を追加で確立するものである。このようなPart 11 の追加要件は、慎重に考慮した上で、自動化された記録保持システムのシステム要件、およびソフトウェア要件に含めること。システム・バリデーションおよびソフトウェア・バリデーションでは、Part 11 の要件を全て満たしているということを立証すること。
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:	コンピュータおよび自動機器は、医療機器の設計、ラボラトリでのテストおよび分析、製品の検査および受入、製造およびプロセス制御、環境管理、梱包、ラベリング、トレーサビリティ、文書管理、苦情処理など、品質システム全体で広範囲に使用されている。自動化された工場の運転に、下記のような内蔵システムを使用することが、ますます多くなってきている。
● programmable logic controllers; ● digital function controllers; ● statistical process control; ● supervisory control and data acquisition; ● robotics; ● human-machine interfaces; ● input/output devices; and ● computer operating systems.	● プログラマブル・ロジック・コントローラ(PLC) ● デジタル・ファンクション・コントローラ ● 統計的なプロセス制御 ● 監視制御およびデータ収集 ● ロボット ● ヒューマン・マシン・インターフェース ● 入力／出力医療機器 ● コンピュータ・オペレーティング・システム
Software tools are frequently used to design, build, and test the software that goes into an automated medical device. Many other	ソフトウェア・ツールは、自動医療機器内で動作するソフトウェアの設計、作成およびテストに用いられることが多い。ほかにもワード・プロセッサ、スプレッド・シート、デー

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.	タベース、フローチャートなど、多くの市販ソフトウェア・アプリケーションが、品質システムの実装に用いられている。このようなアプリケーションは全て、ソフトウェア・バリデーション要件の対象となるが、各アプリケーションに用いるバリデーションの方法は、多様なものである。
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:	製造または品質システムに用いるソフトウェアは、医療機器製造業者の社内開発品であっても、請負会社が開発したものであっても、また市販品を購入した場合であっても、本ガイダンスで概説した基本原則に沿って、開発されたものであること。医療機器製造業者は、ソフトウェア・バリデーションを行う方法の判断について裁量権と自由度を有するが、ソフトウェアを誰がどのように開発するか、またはどこから購入するかを決定する際には、バリデーションを主要な検討事項とすること。ソフトウェア開発者は、ライフサイクル・モデルの定義を行う。バリデーションを証明する典型的なものには、以下が挙げられる。
● 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.	● ソフトウェア開発ライフサイクルの各段階における出力の検証 ● 医療機器製造業者が指定した使用環境で、最終ソフトウェアが適切に動作するかどうかのチェック

6.1. HOW MUCH VALIDATION EVIDENCE IS NEEDED?

6.1. バリデーションの証拠はどの程度必要か？

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	バリデーションにかける労力のレベルは、自動操作により生ずるリスクに相応するものであること。リスクに加え、その他にも、バリデーション業務の一環として必要なテストの性質および範囲を左右する要素がある。そのような要素とは、プロセス・ソフトウェアの複雑性、および医療機器製造業者が、安全かつ効果的な医療機器を製造す

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:	るにあたって自動プロセスにどの程度依存しているか、などである。自動プロセスの要件およびリスク分析に関する文書は、ソフトウェアを使用目的に沿ってバリデートしたことを示すのに必要な証拠の範囲を明確にするのに役立つ。例えば、自動フライス盤の場合、医療機器製造業者がレリース前に、操業のアウトプットが仕様に対して十分に検証されていることを示すことができれば、テストはほとんど必要ない。一方で、次に挙げるものなどには、広範囲なテストが必要となる。
● a plant-wide electronic record and electronic signature system; ● an automated controller for a sterilization cycle; or ● automated test equipment used for inspection and acceptance of finished circuit boards in a life-sustaining / life-supporting device.	● 製造現場全体の電子記録および電子署名のシステム ● 滅菌サイクルの自動制御装置 ● 延命装置や生命維持装置に組み込まれる回路基板の検査、および受入に用いる自動テスト機器
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	多くの市販のソフトウェア・アプリケーションが、品質システム(例えば、品質システムの演算に用いるスプレッド・シートや統計パッケージ、傾向分析に用いるグラフィック・パッケージ、または医療機器・ヒストリ・レコードや苦情処理に用いる市販データベースなど)の一部として使用されている場合がある。このようなソフトウェアに必要なバリデーションの証拠の範囲は、医療機器製造業者が文書化したソフトウェアの使用目的により異なってくる。例えば、医療機器製造業者が、ベンダーの供給するソフトウェアの機能を全部は使わないという選択をした場合、使用される機能、およびソフトウェアの処理結果が医療機器製造業者に影響を与える機能のみを、製造または品質システムの一部としてバリデートすればよい。しかし、リスクの高いアプリケーションは、バリデートされていないソフトウェア機能(たとえ、使用していなくとも)と同じ運用

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).)	環境で動作させてはならない。リスクの高いアプリケーションと低いアプリケーションを同じ操作環境で用いる際には、リソース保護のため、メモリ・パーティションまたはその他の方法によるリスク緩和の技術を考慮する必要が出てくる。ソフトウェアを更新または変更する場合には、医療機器製造業者は、そのような変更によってソフトウェアの「使用部分」に、どのような影響が出るかを考慮する必要があり、またソフトウェアの使用部分のバリデーションを再確認することが必須となる。(21 CFR Section 820.70(i)を参照のこと。)

6.2. DEFINED USER REQUIREMENTS

6.2. ユーザ要件の定義

A very important key to software validation is a documented user requirements specification that defines: ● the "intended use" of the software or automated equipment; and ● the extent to which the device manufacturer is dependent upon that software or equipment for production of a quality medical device. The device manufacturer (user) needs to define the expected operating environment including any required hardware and software configurations, software versions, utilities, etc. The user also needs to: ● document requirements for system performance, quality, error handling, startup,	ソフトウェア・バリデーションを実施する上で非常に重要な鍵となるのは、文書化されたユーザ要求仕様であり、以下の二つの事項を定義したものである。 ● ソフトウェアまたは自動機器の「使用目的」 ● 医療機器製造業者が高品質の医療機器製造のために、そのソフトウェアまたは機器にどの程度依存しているか 医療機器製造業者(ユーザ)は、必要なハードウェアおよびソフトウェアのコンフィギュレーション、ソフトウェアのバージョン、ユーティリティなどの、想定される動作環境を定義する必要がある。ユーザは、以下のことも必要になる。 ● システムの性能、品質、エラー処理、起動／停止、セキュリティなどの要件を文書化すること

shutdown, security, etc.; ● identify any safety related functions or features, such as sensors, alarms, interlocks, logical processing steps, or command sequences; and ● define objective criteria for determining acceptable performance.	● センサー、アラーム、インターロック、論理処理のステップ、コマンド・シーケンスなどの、安全性に関する機能または特徴を明確にすること ● 容認可能な性能を判断するための客観的基準を定義すること
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.	バリデーションは、文書化されたプロトコルに従って行い、バリデーションの結果についても文書化することが必須である。(21 CFR Section 820.70(i)を参照のこと。)システムが予め定められた基準(特に最重要のパラメータについて)に対するパフォーマンスをテストするため、テスト・ケースを文書化すること。テスト・ケースは、以下に対処するものとすること。 ● エラーおよびアラームの条件 ● 起動 ● 停止 ● 全てのユーザ機能およびオペレータ制御 ● 起こりうるオペレータのエラー ● 最大および最少の許容値範囲 ● 機器の使用目的に合った負荷条件 テスト・ケースを実施して結果を記録し、その結果を評価すること。評価を行うのは、ソフトウェアが使用目的に対してバリデートされたという結論が、テスト結果によって裏づけられているかどうかを判断するためである。
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:	医療機器製造業者は、社員を使ってバリデーションを行うこともあれば、機器やソフトウェアのベンダー、またはコンサルタントなどの第三者に依頼する場合もある。いずれの場合でも医療機器製造業者は、製造システムおよび品質システムのソフトウェアが確実に下記に挙げる事項を満たすことについて最終的な責任を負う。
● is validated according to a written procedure	● ソフトウェアが、特定の使用目的に対する手順の文

for the particular intended use; and ● will perform as intended in the chosen application. The device manufacturer should have documentation including: ● defined user requirements; ● validation protocol used; ● acceptance criteria; ● test cases and results; and ● a validation summary that objectively confirms that the software is validated for its intended use.	書に従ってバリデートされていること ● ソフトウェアが、選択したアプリケーションで意図どおりに動作すること 医療機器製造業者は、次の文書を全て所有していること。 ソフトウェアが使用目的に対してバリデートされたということを、客観的に確認するような、 ● ユーザ要件の定義 ● 使用したバリデーション・プロトコル ● 受入基準 ● テスト・ケースおよび結果 ● バリデーションの概要

6.3. VALIDATION OF OFF-THE-SHELF SOFTWARE AND AUTOMATED EQUIPMENT

6.3. 市販ソフトウェアと自動機器のバリデーション

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	医療機器製造業者が用いる自動機器およびシステムのほとんどは、第三者のベンダーが供給するものであり、市販品(OTS)として購入したものである。OTS ソフトウェアの開発者が用いた製品開発の方法を、医療機器製造業者側の OTS ソフトウェア使用目的に対して、確実に適切かつ十分なものとするのは、医療機器製造業者の責任となる。OTS ソフトウェア／機器については、ベンダーのソフトウェア・バリデーション文書にアクセスできる場合とできない場合がある。ベンダーが、システム要件やソフトウェア要件、バリデーションのプロセスおよび結果に関する情報を提供できる場合、医療機器製造業者は、要件とされるバリデーション文書に、その情報を利用することができる。テスト・プロトコルと結果、ソース・コード、設計仕様書、要求仕様書などのベンダーのライフサイクル文書は、ソフトウェアがバリデートされたことを確立するのに役立つが、、市販機器のベンダーからは入手できない場合も多く、またベンダーが機密情報の提供を拒むかもしれない。

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.	
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.	それが可能であり、かつ医療機器のリスクに応じて、医療機器製造業者は、ベンダーが OTS ソフトウェアの構築に用いた設計および開発方法について監査し、OTS ソフトウェアに関して作成された開発およびバリデーション文書を評価することを考慮すべきである。このような監査は、医療機器製造業者、または条件を満たす第三者が行う。監査により、ベンダーが用いた検証／バリデーション活動の手順と結果が、そのソフトウェアを用いて医療機器を製造するにあたって、医療機器の安全性および有効性の要件に対して適切かつ十分なものであることを立証すること。
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	規制環境での仕事に慣れていないベンダーには、医療機器製造業者のバリデーション要件を満たすような文書化されたライフサイクル・プロセスをもたない場合がある。また、監査を許可しない場合もある。必要なバリデーションの情報がベンダーから入手できない場合は、医療機器製造業者は、ソフトウェアが「ユーザのニーズおよび使用目的」に適合することを確立するため、システム・レベルの十分な「ブラック・ボックス」テストを実施する必要が出てくる。しかし、多くのアプリケーションでは、ブラック・ボックス・テストだけでは不十分である。製造する医療機器のリスクや、製造プロセスにおける OTS ソフトウェア

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.	の役割、ベンダー監査の可否、およびベンダーが提供する情報が十分かどうかによって(特に他に代替手段がある場合) OTS ソフトウェア／機器を使用することは適切でないかもしれない。医療機器製造業者は、ベンダーがサポートを終了した場合、OTS ソフトウェアの継続メンテナンスおよびサポートに及ぼす影響(影響がある場合)についても考慮すること。
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.	ソフトウェア・コンパイラやリンカ、エディタ、オペレーティング・システムなどの、市販のソフトウェア開発ツールには、医療機器製造業者が、完全なブラック・ボックス・テストを行うことが実質的に不可能なものもある。バリデーション業務の中で主要な要素であるこのようなテストを実施しなければ、ソフトウェア・ツールをバリデートできないかもしれない。しかし、そのソフトウェアが適切に動作するかどうかは、他の手段で十分推測できる場合もある。例えば、コンパイラは、独立した第三者のテストにより認定されることが多く、市販のソフトウェア製品には、ベンダーから入手できる「バグのリスト」やシステム要件等の運用上の情報があることもある。この情報を使用目的と比較すれば、「ブラック・ボックス」テストの焦点を絞る上で役立つ。市販のオペレーティング・システムは、個別のプログラムとしてバリデートする必要はない。しかし、アプリケーション・ソフトウェアのシステム・レベルでのバリデーション・テストにおいては、最大負荷条件やファイル操作、システム・エラー条件の扱い、、メモリ制約を含む、アプリケーション・プログラムの使用目的に当てはまる、使用する全てのオペレーティング・システム・サービスを対象とすること。

For more detailed information, see the production and process software references in Appendix A.	より詳細な情報については、付録 A に挙げた製造およびプロセス・ソフトウェアに関する参考資料を参照のこと。

APPENDIX A - REFERENCES

付録 A – 参考資料

Food and Drug Administration References

食品医薬品局の参考資料

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
その他の政府参考資料

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/CR-6293. 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
国際的および国内の合意基準

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/DO-178B, December 1992.

Production Process Software References
生産プロセスのソフトウェアに関する参考資料

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

一般的なソフトウェア品質に関する参考資料

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 Publishing Company, 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 Society Press, McGraw-Hill, 1996. ISBN 0-07-039400-8.

Steven R. Mallory, *Software Development and Quality Assurance for the Healthcare Manufacturing 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 FDA Requirements*, Computer Application Specialists, San Diego, CA, 1994.

William Perry, *Effective Methods for Software Testing*, John Wiley & Sons, Inc. 1995. ISBN 0-471-06097-6.

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

付録 B - 開発チーム

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

Uploaded on January 11, 2002