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

Preface　序文

Public Comment　パブリックコメント

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.

コメントと提案は 当局に対する懸念事項として 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 へ提出できる。コメントを提出する際は、本ガイダンスドキュメントの正確なタイトルを言及のこと。ドキュメントが次回改訂またアップデートされるまで、コメントに対する当局側の具体的な対策はとられない。

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

the Center for Devices and Radiological Health (CDRH)や、本ガイダンスの使用や解釈に関する質問は、電話番号 (301) 594-4659 / email jfm@cdrh.fda.gov にて John F. Murray に問い合わせること。

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.

the Center for Biologics Evaluation and Research (CBER)や、本ガイダンスの使用や解釈に関する質問は、電話番号 (301) 827-6220 / email davis@cber.fda.gov にて Jerome Davis に問い合わせること。

Additional Copies　追加コピー

CDRH

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

追加コピーは、インターネット経由：http://www.fda.gov/cdrh/comp/guidance/938.pdf もしくは CDRH Facts-On-Demand 経由にて入手可能。FAX でのドキュメント入手を希望する場合は、電話番号：800-

899-0381 / 301-827-0111、タッチトーン電話にて CDRH Facts-On-Demand system へ問い合わせる。１を押しシステムに入る。次の音声プロンプトで１を押しドキュメントを注文する。ドキュメント番号９３８を入力後＃を押す。後に続く音声プロンプトに従い、リクエストが終了。

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.

追加コピーは、インターネッ経由：http://www.fda.gov/cber/guidelines.htm、書面：CBER, Office of Communication, Training, and Manufacturers' Assistance (HFM-40), 1401 Rockville Pike, Rockville, Maryland 20852-1448 もしくは電話番号：1-800-835-5709 / 301-827-1800 にて入手可能。

目 次

General Principles of Software Validation
ソフトウェアバリデーションの一般原則

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

本文書はガイダンスである。この表題においての食品医薬品局（ FDA ）最新の意見を表すものである。いかなる人物にいかなる権利を作り与えるものでなければ、 FDA や公衆に対し拘束力を行使するものでもない。代替アプローチが適切な法規と規制を満たすのであれば、代替アプローチを用いてもよい。

SECTION 1. PURPOSE 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のバリデーション、もしくは医療機器の設計、開発、製作に用いられるソフトウェアのバリデーションにおいて Food and Drug Administration(FDA)が適切であると判断する一般的なバリデーションの原則を説明するものである。このガイダンスの最終版 Version 2.0 は General Principles of Software Validation, Version 1.1, dated June 9, 1997.の差替えである。

SECTION 2. SCOPE 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.

本ガイダンスは、医療機器品質システム規則にまつわる規定が、ソフトウェアやソフトウェアバリデーションシステムを評価する当局の最新のアプローチに対し、どのように適用されるかを説明する。例えば、本文書ではソフトウェアのバリデーションにおいて FDA が許容する事項を記載してはいるものの、法を遵守する為の、全ての場合における活動やタスクが掲載さ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.

本ガイダンスの範囲は、用語の上でも厳密な定義のバリデーションよりも若干広いものとなった。

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.

計画、ベリフィケーション、テスト、トレーサビリティ、コンフィグレーション管理など、本ガイダンス内で述べられている良いソフトウェアのエンジニアリングに関するその他の側面は、全てが一つとなることで、ソフトウェアのバリデートという最終目的を支援する重要な活動である。

This guidance recommends an integration of software life cycle management and risk management activities.

本ガイダンスでは、ソフトウェアライフサイクル管理とリスク管理活動の統合を勧めている。

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.

ソフトウェアの意図される用途と開発されるソフトウェアに関連する安全リスクに基づいて、ソフトウェア開発者は特定のアプローチ、使用される技術の組合せ、そして労力のレベルを判断すべきである。本ガイダンスでは特定のライフサイクルモデルやその他特別な技法、方法を推奨しないが、ソフトウェアバリデーションとベリフィケーション活動が、ソフトウェアライフサイクル全体を通して行われるべきであることを強調したい。

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.

機器製造業者以外の者によりソフトウェアが開発される場合（例：オフ・ザ・シェルフ・ソフトウェア）、ソフトウェア開発者自身が、直接 FDA の規制遵守にあたる担当者でないことが望ましい。この場合、規制に関する義務を負う部門（例：機器製造業者）が必要とするのは、オフ・ザ・シェルフ・ソフトウェア開発者の活動の妥当性を見極め、機器製造業者の意図した用途でソフトウェアがバリデートされていることの立証に必要な更なる労力を確定することである。

2.1. APPLICABILITY　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 (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).

本ガイダンスは以下の項目に適応する：

- 医療機器のコンポーネント、パーツ、又はアクセサリーとして用いられるソフトウェア
- 医療機器であるソフトウェア（例：血液組織ソフトウェア）
- 装置の製造に用いられるソフトウェア（例：製造機器内の PLC）
- 機器製造業者用品質システムの履行に用いられるソフトウェア（例：機器の履歴を記録、メンテナンスするソフトウェア）

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.

本文書は一般的なソフトウェアバリデーション原理に基づくもので、あらゆるソフトウェアに該当する。FDA の意図としては、the Federal Food, Drug, and Cosmetic Act (the Act)の Section 201 (h)と最新の FDA software and regulatory policy で定義される、規制が適用される医療機器に関するソフトウェアに適用する。本文書は、どのソフトウェアが規制の適用を受けるか、規制の適用を受けないかを具体的に特定するものではない。

2.2. AUDIENCE　オーディエンス

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
- FDA Scientific Reviewers

本ガイダンスは、次の対象者に対し役立つ情報と提案を提供する：

- 医療機器品質システム規則の担当者
- 医療機器ソフトウェアの設計、開発、製造の責任者
- 医療機器の設計、開発または製造に使用する自動化ツールの責任者、あるいは品質システム自体のインプリメントに使用されるソフトウェアツールの設計、開発、製造、調達の責任者
- FDA の査察官
- FDA のコンプライアンス担当職員
- FDA の科学的なレビュア

2.3. THE LEAST BURDENSOME APPROACH　最小限の負荷となるアプローチ

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:
ttp://www.fda.gov/cdrh/resolvingdisputes/ombudsman.html.

医療機器規制のあらゆる分野において、最小限の負荷となるアプローチの試みに関して検討をするべきだと考える。本ガイダンスは、関連する科学的、法的要求を反映し、これら要求に従うことが、もっとも負担のないアプローチであろうと我々は信じている。もし、その他代替のアプローチがより負担を軽減するものであると思われる場合は、我々に一報をいただくことでその見解を検討する。本ガイダンス序文にリストされている担当者もしくは the CDRH Ombudsman に書面によるコメントが可能。連絡手段など CDRH Ombudsman に関する総合的な情報は、インターネット：
http://www.fda.gov/cdrh/resolvingdisputes/ombudsman.html.にて利用可能である。

2.4. REGULATORY REQUIREMENTS FOR SOFTWARE VALIDATION　ソフトウェアバリデーションの規制要求

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.

3140 件の医療機器リコールに対する FDA の分析は、1992 年から 1998 年の間に実施されたもので、その内 242 件（7.7%）はソフトウェアの動作不良に起因するものである。そのソフトウェア関連のリコールのうち、192 件（79%）は、初期の製造と販売後に、ソフトウェアに対し変更がなされた時に生じたソフトウェアの欠陥が原因であった。本ガイダンスで検討しているソフトウェアバリデーションとその他関連した推奨に値するソフトウェアエンジニアリング実践は、このような欠陥と結果的に起こりえたリコールを未然に回避するための重要な手段である。

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則の要求で、1996 年 10 月に the Federal Register で発行され、1997 年 6 月 1 日に発効となった。（Title 21 Code of Federal Regulations (CFR) Part 820, 61 Federal Register (FR) 52602, 各々参照）バリデーション要求は医療機器のコンポーネントとして使用されるソフトウェア、ソフトウェア自体が医療機器であるもの、機器製造業者の品質システムの導入、機器製造もしくは機器製造業者の品質システムのインプリメントに使用されるソフトウェアに適用される。

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.

分類規制から厳密に除外されない限り、1997 年 6 月 1 日より後に開発された全医療機器ソフトウェア製品は、機器部類に関らず、該当する設計管理規定の対象となる（21 CFR §820.30 参照）。この要求は、現行の開発プロジェクトの完結、あらゆる新規開発プロジェクト、現行の医療機器ソフトウェアに対し行われた変更を含む。デバイスソフトウェアのバリデーションに関する特定の要求は 21 CFR §820.30(g)に述べられている。その他設計管理（計画、入力、ベリフィケーション、レビュ）も医療機器ソフトウェアで必須となる。（21 CFR §820.30.参照）これら作業に対する結果を文書化したものは、医療機器ソフトウェアがバリデートされたことを証明する補助資料となる。

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 acceptance, manufacturing, labeling, packaging, distribution, complaint handling, or to automate any other aspect of the quality system.

機器製造プロセス、もしくは品質システムのあらゆる部分を自動化してきたソフトウェアは、21 CFR §820.70(i).で要求されているように、意図する用途においてバリデートされていなければならない。この要求は、自動化機器の設計、テスト、コンポーネント受け入れ、製造、ラベリング、パッケージング、販売、苦情対応、または品質システムに関するあらゆる側面について、すべ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.

また、電子記録の作成、修正、保持と電子署名の管理に用いられたコンピュータシステムは、バリデーション要求の対象となる（21 CFR §11.10(a).参照）。これらコンピュータシステムは、正確性、信頼性、一貫した意図する性能の発揮、無効もしくは改ざんされた記録を識別する能力を保証する上でバリデートされていなければならない。

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のソフトウェアは、社内もしくは契約に基づき開発することができる。しかし、ソフトウェアは特定の使用目的の下、オフ・ザ・シェルフを購入される頻度が高い。全製品/品質システムソフトウェアは、オフ・ザ・シェルフとして購入されても、その使用用途を定義した要求と比較できるテスト結果とその他エビデンスについての情報を文書化して、そのソフトウェアが意図する用途に対しバリデートされていることを示すべきである。

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

自動化された医療機器と、自動化による製造、品質システム運用においてオフ・ザ・シェルフ・ソフトウェアの用途は増加している。オフ・ザ・シェルフ・ソフトウェアは多様な機能を持ち合わせるが、その中の一部の機能だけが機器製造業者にとって必要となる。機器製造業者は、機器を製造する際、機器の内部で使用されるソフトウェアの妥当性に責任を負う。機器製造業者はオフ・ザ・シェルフ・ソフトウェアを購入した際、選択したアプリケーションで意図した性能を発揮することを確実にしなければならない。製造、もしくは品質システムに用いられるオフ・ザ・シェルフ・ソフトウェアに関しては、本文書 Section 6.3 に補足ガイダンスが盛り込まれている。デバイスソフトウェアに関しての便利な情報は、FDA の Guidance for Industry, FDA Reviewers, and Compliance on Off-The-Shelf Software Use in Medical Devices.に掲載されている。

2.4. QUALITY SYSTEM REGULATION VS PRE-MARKET SUBMISSIONS　品質システム規則と市販前申請

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.

本文書では、ソフトウェアバリデーションの実施を含む、品質システム規則に関する問題も取り扱う。ここでは、ソフトウェアバリデーションプロセスを管理しコントロールするためのガイダンスを提供する。ソフトウェアバリデーションプロセスの管理とコントロールは、例えば、自動化製造プロセスにおけるバリデーションプロセスといった他のバリデーション要求と混同してはならない。

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

機器製造業者は、FDA への市販前に行う申請と同様、品質システムと設計管理要求に準拠する為、同じ手順と記録を使用することもある。本文書は、ソフトウェアバリデーションに関連する特定の安全性、有効性にまつわる問題を取り扱うものではない。規制をうけるソフトウェアで市販前の申請に必要となる設計に関する問題と文書化の要求は、本文書に記載されていない。安全性、有効性に関連すること、そして市販前の申請に必要となる文書に特化した問題は、the Office of Device Evaluation (ODE), Center for Devices and Radiological Health (CDRH)もしくは the Office of Blood Research and Review, Center for Biologics Evaluation and Research (CBER)に記載されるだろう。市販前申請のための推奨する FDA ガイダンスドキュメントに関しては、Appendix A を参照のこと。

SECTION 3. CONTEXT FOR SOFTWARE VALIDATION　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.

多くの人がこれまでに要望してきたのは、ソフトウェアバリデーションに関する品質システム規則の遵守について、FDA が何を求めているかを述べた具体的なガイダンスである。本文書が提供する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。4 章と 5 章に列記した原則とタスクを使ったソフトウェアのバリデーションは、約 20 年以上にわたるソフトウェア業界の多くの場面を導いてきたものである。

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、製造施設等の非常な多様性により、適切なバリデーション要求の全てを一つのドキュメント内において述べることはできない。しかしながら、一般的な幾つかの大まかなコンセプトの適用は、ソフトウェアバリデーションのガイダンスとして上手に活用することができる。これら大まかなコンセプトは、ソフトウェアバリデーションの包括的なアプローチを構築する受け入れ可能なフレームワークを提供する。追加情報は Appendix A に列記した多くの参照文献から利用可能である。

3.1. DEFINITIONS AND TERMINOLOGY　定義と専門用語

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.

品質システム規則で定義されない限り、もしくは別の用法にて下記に明記されない限り、本ガイダンスに使用されるその他用語の全ては、the FDA Glossary of Computerized System and Software Development Terminology.の最新版の中で定義している。

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.

医療機器品質システム規則(21 CFR 820.3(k))では、"**establish**"を“定義する、文書化する、インプリメ

ントする”と定義づけている。本ガイダンス内で、"establish"、"established"という言葉は、これと同様の意味を有するものとして解釈すること。

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.

医療機器品質システム規則に述べられている幾つかの用語の定義は、ソフトウェア業界で一般的に使用される専門用語と比較すると混乱してしまう。例をあげると、要求、仕様、ベリフィケーション、バリデーションなどである。

3.1.1 Requirements and Specifications 要求と仕様

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 software requirements is a crucial factor in successful validation of the resulting software.

品質システム規則は、設計のための要求は文書化されなければならない、そしてその特定の要求は検証されなければならないと述べている一方で、“要求”と“仕様”の違いをそれほど明確に述べていない。**要求**とは、システムやソフトウェアに対するあらゆるニーズ、期待値である。要求は明示されたあるいは暗黙の顧客のニーズを反映し、市場からのものや、契約によるもの、法を遵守するためのものであったり、組織内部の要求でもあり得る。多種多様な要求書（例：設計、機能、インプリメンテーション、インターフェース、パフォーマンス、物理的要求）が存在する。ソフトウェア要求は、一般的にソフトウェアに割り当てられたシステムの機能といった側面に対するシステム要求から生成される。ソフトウェア要求は一般的に機能的な条件として記載され、開発プロジェクトが進むにしたがって定義、改良、更新される。ソフトウェア要求を正確かつ完全に文書化することは、最終的にソフトウェアのバリデーションを成功させることの重大な要因となる。

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.

仕様という言葉は、“要求を記述した文書”と定義される（21 CFR §820.3(y).参照）。それは図やパターン、その他関連ある文書を参照するか含み、たいていの場合、それによって要求が満たされることを確認できる内容と条件を表している。多種多様な仕様書ー例としてシステム仕様書、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がある。これらすべてのドキュメントが、“要求事項の特定”をおこない、設計の結果として、様々なベリフィケーションのためのフォームが必要となる。

3.1.2 Verification and Validation ベリフィケーションとバリデーション

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.

品質システム規則は、ISO 8402:1994 の解釈と同様、"ベリフィケーション" と"バリデーション"を別々の異なる用語として扱っている。一方では、多くのソフトウェアエンジニアリング雑誌の記事や教科書では、"ベリフィケーション" と"バリデーション"という用語を交互に用いていたり、いくつかのケースでは、ソフトウェアの"ベリフィケーションとバリデーションとテスト(VV&T)"などのように、ひとつのコンセプトとして言及し、区別は存在しないとしていることもある。

Software verification provides objective evidence that the design outputs of a particular phase of the software development life cycle meet all of the specified requirements for that phase. Software verification looks for consistency, completeness, and correctness of the software and its supporting documentation, as it is being developed, and provides support for a subsequent conclusion that software is validated. Software testing is one of many verification activities intended to confirm that software development output meets its input requirements. Other verification activities include various static and dynamic analyses, code and document inspections, walkthroughs, and other techniques.

ソフトウェアのベリフィケーションは、ソフトウェア開発ライフサイクル中のあるフェーズの設計結果が、そのフェーズにおける指定された要求を満たすことを示す証拠を提供する。ソフトウェアのベリフィケーションは、ソフトウェアとその関連文書の一貫性、完全性、および正確性を検討し、その開発がおこなわれた際に、ソフトウェアがバリデートされたということを後に結論付けることに役立つ。ソフトウェアのテストは、多くのベリフィケーション作業のうちの一つであり、ソフトウェアの開発結果が要求を満たすことを確認しようとするものである。その他のベリフィケーション作業には、様々な静的および動的な分析、コードとドキュメントの検査、ウォークスルー、その他のテクニックがある。

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 are typically included as components of an overall design validation program for a software automated device.

ソフトウェアバリデーションは、完成した機器の設計バリデーションの一部であるが、品質システム規則の中では別個に定義されていない。本ガイダンス目的として、FDA がソフトウェアバリデーションを、"**ソフトウェアの仕様がユーザニーズや意図する用途に一致しており、特定の要求がソフトウェアに一貫して満たされていることを検査と客観的な証拠により確認すること**"と考えている。実際に、ソフトウェアバリデーション活動は、全要求事項が満たされたことを保証するために、ソフトウェア開発ライフサイクルの開発中と終了後の両方に起こりうる。通常ソフトウェアは大きなハードウェアシステムの一部なので、ソフトウェアのバリデーションは、一般的に全ソフトウェア要求が正確かつ完全にインプリメントされ、システム要求に対してトレーサブルであることなどの証拠を含んでいる。ソフトウェアがバリデートされたという結論は、包括的なソフトウェアテスト、検査、分析、その他ソフトウェア開発ライフサイクルの各段階におけるベリフィケーションのタスクに大きく依存する。デバイスソフトウェアに対して、利用環境をシミュレートした機能テストやユーザサイトにおけるテストを実施することは、一般的に自動化機器ソフトウェアの為の全体的設計バリデーションプログラムのひとつとして含まれる。

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.

ソフトウェアのベリフィケーションとバリデーションは困難で、その理由は、開発者が永久的にテストできないことと、どの程度をもって証拠が十分であるかを判断することが難しいからである。大きな尺度で見た場合、ソフトウェアバリデーションは、機器が、ソフトウェア自動化機能と機器の特性に対して、全要求事項とユーザの期待値を満たすことである“信用性のレベル”の開発作業の問題である。仕様書で見つかった欠陥、引き続き存在する欠陥の予測、テスト範囲、そしてその他のテクニックといった基準は製品の出荷前に受け入れられる信用性の問題を明らかにするため、すべて用いられる。信用性のレベル、つまりソフトウェアバリデーション、ベリフィケーション、必要になるテスト労力のレベルは、機器の自動化機能とのよりもたらされた安全リスク（ハザード）により異なるであろう。ソフトウェアの安全リスク管理に関するガイダンスは、FDA Guidance for the Content of Pre-market Submissions for Software Contained in Medical Devices の Section 4, Appendix A にある参照文献、国際基準 ISO/IEC 14971-1 and IEC 60601-1-4 で述べられている。

3.1.3 IQ/OQ/PQ

For many years, both FDA and regulated industry have attempted to understand and define software validation within the context of process validation terminology. For example, industry documents and other FDA validation guidance sometimes describe user site software validation in terms of installation qualification (IQ), operational qualification (OQ) and performance qualification (PQ). Definitions of these terms and additional information regarding IQ/OQ/PQ may be found in FDA’s Guideline on General Principles of Process Validation, dated May 11, 1987, and in FDA’s Glossary of Computerized System and Software Development Terminology, dated August 1995.

長期にわたり、FDA と規制の適用を受ける業界は、プロセスバリデーションにおける専門用語で、ソフトウェアバリデーションの理解や定義付けを試みた。例えば、業界文書やその他 FDA バリデーションガイダンスは、installation qualification (IQ：設置適格性検証), operational qualification (OQ：稼動適格性検証) and performance qualification (PQ：性能適格性検証)の観点からユーザによるソフトウェアバリデーションを何度か記載している。これらの用語の定義と IQ、OQ、PQ に関する追加的情報は、FDA の 1987 年 5 月 11 日付け Guideline on General Principles of Process Validation、1995 年 8 月付け Glossary of Computerized System and Software Development Terminology で述べている。

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 の専門用語はその目的に十分沿い、ユーザ側でのソフトウェアバリデーションタスクを系統づける、数ある合法的な方法の一つではあるが、この専門用語は多くのソフトウェア専門家の間ではよく理解がされていないおそれがあり、本文書でも別の箇所では扱っていない。しかしながら、FDA

職員と機器製造業者は、ソフトウェアバリデーションに関する情報を求め、提供する立場にあることから、これら用語の違いを把握することが必要となる。

3.2. SOFTWARE DEVELOPMENT AS PART OF SYSTEM DESIGN　システムデザインの一端となるソフトウェア開発

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.

ソフトウェアを用いたシステム機能の導入の決定は、通常システム設計時に行われるものである。ソフトウェア要求は一般的に総合的なシステム要求と、導入が見込まれるソフトウェアを使用するシステムにおいてはその要求面の設計から得られる。完成した機器に対するユーザのニーズや意図する用途はあるが、一般的にユーザは、これらの要求がハードウェア、ソフトウェア、もしくは両方を組み合わせたものが要求に合うかどうかを特定しない。したがって、ソフトウェアバリデーションは、システムの総合的な設計バリデーションの内容で考慮されなければならない。

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.

要求仕様書は製品が開発される過程のユーザニーズや意図する用途を表している。ソフトウェアバリデーションの主たるゴールは完成したソフトウェア製品が、文書化されたソフトウェアとシステム要求を遵守していることを証明することである。システム要求とソフトウェア要求における正確性と完全性は、機器の設計バリデーションプロセスの一部として明記すべきである。ソフトウェアバリデーションでは、全ソフトウェア仕様書の整合性と、全ソフトウェア要求がシステムの仕様書とトレースができることの確認作業も含まれる。確認作業は、全体的な設計バリデーションの重要な役割を担い、医療機器がユーザニーズと意図した用途に適合するというあらゆる側面を保証するものである。

3.3 Software is Different from Hardware ソフトウェアはハードウェアと異なる

While software shares many of the same engineering tasks as hardware, it has some very important

differences. For example:

ソフトウェアはハードウェアと同様に、多くのエンジニアリングタスクを費やすけれども、非常に重要な相違点が存在する。例えば：

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

- ソフトウェアにまつわる問題の大部分は、設計、開発プロセスの間に生じたエラーをトレース可能である。ハードウェア製品の品質は、設計、開発、製造に大きく依存しているが、ソフトウェア製品の品質は、ソフトウェアの製造に関してさほど気遣うことなく、主に設計と開発に依存している。ソフトウェアの製造は容易に検証できる再生によるものである。オリジナルと同じように機能する大多数のプログラムコピーの製造は難しくない。難しいのは、全仕様書に見合うオリジナルプログラムを入手することである。

·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.

- 最も重要なソフトウェアの特徴の一つは、条件分岐である。つまり、異なる入力によって、別の種類のコマンドを実行する能力である。この特徴はソフトウェアの他の特徴よりも最もそのものを複雑にする要因である。短いプログラムでさえも複雑になり、完全に理解することが困難な場合がある。

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

- 通常、テストだけでは、ソフトウェアが完全で正確であることをすべて検証できない。テストに加えて、他のベリフィケーションテクニックや構造化及び文書化された開発プロセスが組み合わされて、包括的なバリデーションアプローチを保証すべきである。

·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.

- ハードウェアと異なり、ソフトウェアは物理的実体でなく、消耗しない。実際に、潜在的欠陥が発見され取り除かれていくので、経時的にソフトウェアは改善されるだろう。しかし、ソフトウェアは絶えずアップデートされ、または変更されるので、変更時にソフトウェアに新たな欠陥がもたらされることにより、改善が逆効果を与える場合もある。

·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 marketplace.

- ハードウェアの欠陥と異なり、ソフトウェアの欠陥は事前の警告なしに発生する。ソフトウェア条件分岐において、実行時に異なるパスへ誘導してしまうという欠陥が、ソフトウェア製品が市場に出て長い時間が経つまで潜んでしまうことがある。

·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.

- ソフトウェアのその他関連する特徴は、その変更されるスピードと容易性である。この要因は、ソフトウェアの専門家や非専門家に、ソフトウェアの問題は容易に修正できると信じさせてしまうことになる。ソフトウェアの理解不足に加え、厳しくコントロールしたエンジニアリングは、ハードウェアに要求されるほどソフトウェアには必要ではないと、マネージャに認識させてしまうことがある。実際逆も真である。**複雑だからこそ、ソフトウェアの開発プロセスは、開発プロセス以降では容易に発見できない問題を防ぐため、ハードウェアよりも厳しくコントロールされるべきである。**

・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.

- ソフトウェアコードの一見重要でないと思われる変更も、ソフトウェアプログラムのどこかで、予期しないとても重大な問題をもたらすことにつながる。ソフトウェア開発プロセスは、綿密に計画、管理、文書化され、ソフトウェアの変更による予期しなかった結果を発見し、修正できるものでなければならない。

·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に対する高い需要と、頻繁な作業要員の異動により、ソフトウェアの変更を受け持つソフトウェア要員は、オリジナルのソフトウェア開発に携わっていなかったかも知れない。それ故、正確で完全なドキュメンテーションが重要である。

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

- 従来、ソフトウェアのコンポーネントはハードウェアのコンポーネントのように、頻繁に標準化されることはなく、交換可能なものでもなかった。しかし、医療デバイスソフトウェア開発者は、コンポーネントベースの開発ツールと技術を使用し始めている。オブジェクト指向の方法論やオフ・ザ・シェルフ・ソフトウェアコンポーネントの使用により、より早く、より安価なソフトウェア開発が可能になった。しかしながら、コンポーネントベースのアプローチは、インテグレーションにおいて、非常に慎重な注意が必要である。インテグレーションの前に、再利用可能なソフトウェアコードのすべての定義と開発、そしてオフ・ザ・シェルフコンポーネントの動作をすべて理解するための時間が必要になる。

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

上記の理由とその他理由により、ソフトウェアエンジニアリングは、ハードウェア以上の、管理者による綿密な調査と、管理に高い水準を求められる。

3.4. BENEFITS OF SOFTWARE VALIDATION　ソフトウェアバリデーションの利点

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 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.

ソフトウェアバリデーションは、デバイスソフトウェアやソフトウェアによる自動作業の品質を保証する重要なツールである。ソフトウェアバリデーションは、機器の有用性や信頼性を向上させることが

可能になり、結果として故障率の低下、回収と是正措置の低減、患者やユーザに対するより低いリスク、機器製造業者の責任の軽減をもたらす。またソフトウェアバリデーションは、ソフトウェアの修正を確実にしたり、ソフトウェアの変更を再度バリデートすることにより一層容易に、コストのかからない形にすることで、長期にわたるコストの削減も可能になる。ソフトウェアメンテナンスは、ソフトウェアの全ライフサイクルにかかるすべてのコストに対して、大きなパーセンテージを占めている。確立した包括的ソフトウェアバリデーションプロセスにより、ソフトウェアの次回リリースに伴うバリデーションコストを減らすことができ、ソフトウェアの長期にわたるコストを削減することができる。

3.5 DESIGN REVIEW　デザインレビュ

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.

デザインレビュは、文書化され、理解しやすく、体系的な調査であり、設計要求の妥当性や、その要求を満たす設計の有用性を評価し、問題を特定するものである。ソフトウェアプロジェクトの開発チーム内には、多くの非公式的なテクニカルレビュが発生する可能性はあるが、公式なデザインレビュはより一層構造化されたもので、開発チーム以外の参加者を含むものである。デザインレビュはソフトウェアがシステム内でハードウェアと統合した後ソフトウェアに対し、あるいはソフトウェアとハードウェアに対し、個々に行われる。デザインレビュは、開発プランの検討、要求仕様書、設計仕様書、テスト計画書と手順書、プロジェクトに関連するあらゆるドキュメントと活動、定義されたライフサイクルの各ステージにおけるベリフィケーション結果、全体的な機器のバリデーション結果を盛り込まなければならない。

Design review is a primary tool for managing and evaluating development projects. For example, formal design reviews allow management to confirm that all goals defined in the software validation plan have been achieved. The Quality System regulation requires that at least one formal design review be conducted during the device design process. However, it is recommended that multiple design reviews be conducted (e.g., at the end of each software life cycle activity, in preparation for proceeding to the next activity). Formal design review is especially important at or near the end of the requirements activity, before major resources have been committed to specific design solutions. Problems found at this point can be resolved more

easily, save time and money, and reduce the likelihood of missing a critical issue.

デザインレビュは、開発プロジェクトの管理、評価をしていく為の主要なツールである。例えば、公式デザインレビュにより、ソフトウェアバリデーション計画に定義される全ての目的が達成されたことを確認する管理が可能となる。品質システム規則は、最低でも一つの公式なデザインレビュが機器設計プロセス時に行われることを求めている。しかし、複数のデザインレビュを行うことを推奨する（例：各ソフトウェアライフサイクル活動の最終局面、次の活動に引き続く準備期間)。主要なリソースが特定の設計に関する解決法にあてがわれる以前に、公式デザインレビュは要求活動の最終局面時、もしくはその前後で特に重要である。この時点で発見された問題はより容易に解決され、時間とコストを節約し、重大な問題を見逃してしまう可能性を減らすことができる。

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

キーポイントになる質問の回答は、公式デザインレビュの間に文書化すること。また、以下を含めること：

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

- 各ソフトウェアライフサイクル活動に対し適切なタスクと予測された結果、出力、あるいは成果が達成されたか？

・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?

- 各ソフトウェアライフサイクル活動のタスクと予測された結果、出力、もしくは成果を行う：
  - ✓ 正確性、完全性、一貫性、精密性において、その他ソフトウェアライフサイクル活動の要求を遵守しているか？
  - ✓ 当該活動の標準、実践、ルールを満たしているか？
  - ✓ 次のソフトウェアライフサイクル活動の初期タスクに対して、適切な基盤が整っているか？

SECTION 4. PRINCIPLES OF SOFTWARE VALIDATION 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.

本セクションは、ソフトウェアのバリデーションとして考慮すべき一般的原則を述べている。

4.1. REQUIREMENTS 要求

A documented software requirements specification provides a baseline for both validation and verification. The software validation process cannot be completed without an established software requirements specification (Ref: 21 CFR 820.3(z) and (aa) and 820.30(f) and (g)).

文書化されたソフトウェア要求仕様書はバリデーションとベリフィケーションのベースラインである。ソフトウェアバリデーションプロセスは、確立したソフトウェア要求仕様書なしに完結しない(Ref: 21 CFR 820.3(z) and (aa) and 820.30(f) and (g)).

4.2. DEFECT PREVENTION 欠陥の回避

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は、ソフトウェア開発プロセスに欠陥をもたらさないことに焦点を当てるべきで、ソフトウェアコードの書き込みが終了した後に、ソフトウェアコードに"テストの品質"を注入することではない。ソフトウェアテストは、ソフトウェアコードの全潜在的欠陥を表面化するということにかなり限定されている。例えば、多くのソフトウェアはその複雑性のため、徹底的にテストができない。**ソフトウェアのテストは必須な活動である。しかし、ほとんどの場合、ソフトウェアのテストは、ソフトウェアが意図した用途を満たすものであるという信頼性を保証する上で、十分なものではない。**信頼性を保証する為には、ソフトウェア開発者は、種々の方法とテクニックを使い合わせ、ソフトウェアのエラーを回避し、起こりうるソフトウェアのエラーを発見しなければならない。方法の"ベストミックス"は、開発環境、アプリケーション、プロジェクトの規模、言語、リスク等多くの要素に依存する。

4.3. TIME AND EFFORT　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ている状況を作りだすことは、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、設計と開発計画や設計計画時などの早い時期に行われるべきである。ソフトウェアがバリデートされた最終的結果は、ソフトウェアライフサイクルを通して実施された計画に基づいた労力から収集した証拠に基づかなければならない。

4.4. SOFTWARE LIFE CYCLE　ソフトウェア 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、確立したソフトウェアライフサイクル環境内で行われる。ソフトウェアライフサイクルは、ソフトウェアバリデーションの取組みのサポートに必要なソフトウェアエンジニアリングタスクと文書を含む。加えて、ソフトウェアライフサイクルは、ソフトウェアの意図する用途に見合った、特定のベリフィケーションとバリデーションタスクも含む。本ガイダンスは特定のライフサイクルモデルを推奨するものではないが、ソフトウェア開発プロジェクトに対して、選択し使用するものである。

4.5. PLANS　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 specify areas such as scope, approach, resources, schedules and the types and extent of activities, tasks, and work items.

ソフトウェアバリデーションプロセスは、計画を通して定義、管理される。ソフトウェアバリデーション計画は“何が”ソフトウェアバリデーションの取組みによって成し遂げられるかを定義する。ソフトウェアバリデーション計画は、重要な品質システムツールであり、範囲、アプローチ、リソース、スケジュール、活動、タスク、作業アイテムのタイプと範囲などの領域を明確にする。

4.6. PROCEDURES　手順書

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.

ソフトウェアバリデーションプロセスは、手順書に沿って実行される。これら手順書は“どのように”ソフトウェアバリデーションの取組みを実行するのかを定める。手順書は、各バリデーション活動、タスク、作業アイテムを完結するために、特定のアクションや一連のアクションを明確にしなければならない。

4.7. SOFTWARE VALIDATION AFTER A CHANGE 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.

ソフトウェアの複雑性により、小さく、ローカル規模にみなされる変更も、重大なグローバルシステムインパクトを持つことがある。ソフトウェアに対しいかなる変更（小さな変更でも）がなされた時、ソフトウェアのバリデーション状態は、再度構築する必要がある。いつソフトウェアが変更しようとも、バリデーション分析は、個別の変更のバリデーションだけに対し行われるのでなく、全体のソフトウェアシステムにおいて、変更の範囲と影響を見極めなければならない。この分析に基づき、ソフトウェア開発者は、変更されていないが、障害をうけやすいシステムの部分が、逆に影響を受けていないことを示す為、ソフトウェアのレグレッションテストを適切なレベルで行わなければならない。設計管理と適切なレグレッションテストは、ソフトウェア変更後、ソフトウェアがバリデートされたことの確信を与えるものである。

4.8. VALIDATION COVERAGE バリデーション範囲

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.

バリデーションの範囲は、ソフトウェアの複雑性と安全リスクに基づくべきで、企業規模やリソースの節約に基づいてはならない。バリデーション活動、タスク、作業アイテムの選定は、ソフトウェア設

計の複雑性と特定に意図した用途をもつソフトウェアの仕様に関連したリスクに比例するもでなければならない。リスクが低い機器に対しては、ベースラインに沿ったバリデーション活動のみが実行されればよい。リスクが増加するほど、そのリスクに対応する追加のバリデーション活動が必要となる。バリデーション文書は全てのソフトウェアバリデーション計画と手順が無事に完結したことを示す上で、必要となる。

4.9. INDEPENDENCE OF REVIEW　レビュの独立

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.

バリデーション活動は、"レビュの独立"という基本的な品質保証の指針を用いて行われなければならない。セルフバリデーションはとても困難である。可能な場合は、特にリスクが高いアプリケーションの場合、独立した評価を行うことが常に望ましい。第三者機関に独立したベリフィケーション、バリデーションの外注を出している会社もあるが、この解決法は、必ずしも適したものではない。他のアプローチは、特定の設計、もしくはインプリメントに関りがないがプロジェクトを評価し、ベリフィケーション、バリデーション活動を行う十分な知識をもつ社内のスタッフメンバーを任命することである。規模が小さな会社は、社内でレビュの独立を維持するため、タスクの整理と割り当てに関してクリエイティブであることが望まれる。

4.10. FLEXIBILITY AND RESPONSIBILITY　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.

これらソフトウェアバリデーション原則の導入は、各アプリケーションで全く異なる可能性がある。機器製造業者はバリデーション原則の適用方法を柔軟に選定するが、ソフトウェアがバリデートされていることを証明する根本的な責任も保持する。

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:

ソフトウェアは、幅広い環境の範囲に応じて、そしてリスクレベルの異なる機器の種類に応じて、設

計、開発、バリデート、規制化されている。FDA 規制化医療機器のアプリケーションは、以下の特徴をもつソフトウェアを含む：

·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.

- 医療機器のコンポーネント、パーツ、アクセサリーである
- それ自体が医療機器である
- 製造、設計と開発、その他品質システムのパーツとして使用される

In each environment, software components from many sources may be used to create the application (e.g., in-house developed software, off-the-shelf software, contract software, shareware). In addition, software components come in many different forms (e.g., application software, operating systems, compilers, debuggers, configuration management tools, and many more). The validation of software in these environments can be a complex undertaking; therefore, it is appropriate that all of these software validation principles be considered when designing the software validation process. The resultant software validation process should be commensurate with the safety risk associated with the system, device, or process.

各環境では、多くのリソースからなるソフトウェアコンポーネントはアプリケーションの作成に用いられる（例：社内開発ソフトウェア、オフ・ザ・シェルフ・ソフトウェア、コントラクトソフトウェア、シェアウェア)。加えて、ソフトウェアコンポーネントは、多くの異なるフォームがある（例：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、コンフィグレーション管理ツール、その他)。これら環境におけるソフトウェアバリデーションは、複雑な作業となる。それゆえ、ソフトウェアバリデーションプロセスを設計する際、これら全てのソフトウェアバリデーションの原則が考慮されることが適切である。最終的なソフトウェアバリデーションプロセスは、システム、機器、プロセスに関連する安全リスクに比例していなければならない。

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の活動とタスクは、異なるロケーションで起こり、異なる組織により実行されることで分散化される可能性がある。しかし、タスクの分配化、契約関係、コンポーネントのリソース、開発環境に関係なく、機器開発者もしくは仕様書開発者は、ソフトウェアがバリデートされていることを保証する責任を保持する。

SECTION 5. ACTIVITIES AND TASKS 活動とタスク

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を通して完了する。これらタスクは、用いられるライフサイクルモデルと、ソフトウェアプロジェクトプロセスとしての変更の範囲により、発生が一回のこともあれば、何回か繰り返されることもある。

5.1. SOFTWARE LIFE CYCLE ACTIVITIES 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 life cycle model include the following:

本ガイダンスは特定のソフトウェアライフサイクルモデルの活用を勧めるものではない。ソフトウェア開発者は、製品と組織に適切なソフトウェアライフサイクルモデルを構築しなければならない。選定したソフトウェアライフサイクルモデルは、ソフトウェアの誕生から廃棄までをカバーするもでなければならない。一般的なソフトウェアライフサイクルモデルの活動は以下を含む：

- Quality Planning
- System Requirements Definition
- Detailed Software Requirements Specification
- Software Design Specification
- Construction or Coding
- Testing
- Installation
- Operation and Support
- Maintenance
- Retirement

- 品質計画
- システム要求定義
- 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
- 導入

- 運用とサポート
- メンテナンス
- 廃棄

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.

ベリフィケーション、テスト、その他ソフトウェアバリデーションをサポートするタスクは、各活動の間に実行される。ライフサイクルモデルは、様々な方法でソフトウェア開発活動を組織し、ソフトウェア開発プロジェクトをモニターし、管理するフレームワークを提供する。幾つかのソフトウェアライフサイクルモデル（例：ウォータフォール、スパイラル、早期開発プロジェクト、追加開発等、）は 1995 年 8 月付 FDA の Glossary of Computerized System and Software Development Terminology,にて定義されている。これらとその他多くのライフサイクルモデルは Appendix A の参考文献で記載されている。

5.2. TYPICAL TASKS SUPPORTING VALIDATION 標準的タスクサポートバリデーション

For each of the software life cycle activities, there are certain "typical" tasks that support a conclusion that the software is validated. However, the specific tasks to be performed, their order of performance, and the iteration and timing of their performance will be dictated by the specific software life cycle model that is selected and the safety risk associated with the software application. For very low risk applications, certain tasks may not be needed at all. However, the software developer should at least consider each of these tasks and should define and document which tasks are or are not appropriate for their specific application. The following discussion is generic and is not intended to prescribe any particular software life cycle model or any particular order in which tasks are to be performed.

各ソフトウェアライフサイクル活動には、ソフトウェアがバリデートされたことの結果を裏付ける"典型的"なタスクがある。しかし、実行されるタスク、実行する順序、実行の繰り返しとタイミングは、選定されたソフトウェアライフサイクルモデルとソフトウェアアプリケーションに関連する安全リスクにより決定する。リスクがとても低いアプリケーションについては、全く必要のないタスクも考えられる。しかし、ソフトウェア開発者は、最低限、これら各タスクを考慮し、特定のアプリケーションにおいてどのタスクが適切もしくは、適切でないかを定義し文書化することが必要である。以下の論議は一般的なもので、特定のソフトウェアライフサイクルモデルや実行されるタスクの順番を指示するものではない。

5.2.1. Quality Planning　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:

設計と開発の計画は、必要なタスク、異常性の報告と解決に関する手順、必要なリソース、正式なデザインレビュも含むマネージメントレビュ要求を特定する計画とならなければならない。ソフトウェアライフサイクルモデルと関連する活動はまた、各ソフトウェアライフサイクル活動において必要なタスクと同様に明確にすべきである。計画は以下を含む：

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

- 各ライフサイクル活動に特定のタスク
- 重要な品質要因の一覧表（例：信頼性、保守性、有用性）
- 各タスクの方法と手順
- タスクの受入条件
- 入力要求に適合すると評価される出力の定義と文書化の条件
- 各タスクのイ入力
- 各タスクからの出力
- 各タスクの役割、リソース、責任
- リスクと仮定
- ユーザニーズの文書化

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.

マネージメントは適切なソフトウェア開発環境とリソースを特定し、準備しなければならない。(See 21 CFR §820.20(b)(1) and (2).参照)。一般的に各タスクは物理的リソースと同様に人員を必要とする。計画では各タスクとリスク管理(ハザード)が行う役割に対し、人員、施設及び設備のリソースを特定する。コンフィグレーション管理計画では、複数の並行する開発活動を管理しコントロールするよう作成し、適切なコミュニケーションと文書化を保証しなければならない。あらゆる承認済バージョンにわたる仕様書、ソースコード、オブジェクトコード、ソフトウェアシステムを構成するテストパッケージソフトにおいて、コントロールの完全性と正確性が保証されていることが要求される。またコントロールは、現在の承認済バージョンを正確に特定し、アクセスを保証しなければならない。

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.

手順書は、バリデーションやその他活動を通して発見したソフトウェアの異常を報告し、解決するために作成される。マネージメントは報告を確認し、各レポートの内容、フォーマット、組織の責任要員を明確にする。手順書はまた、レビュ、承認などの組織の責任要員などソフトウェア開発結果のレビュおよび承認においても必要である。

Typical Tasks – Quality Planning
·Risk (Hazard) Management Plan
·Configuration Management Plan
·Software Quality Assurance Plan
−Software Verification and Validation Plan
▫Verification and Validation Tasks, and Acceptance Criteria
▫Schedule and Resource Allocation (for software verification and validation activities) q
Reporting Requirements
−Formal Design Review Requirements
−Other Technical Review Requirements
·Problem Reporting and Resolution Procedures
·Other Support Activities

一般的なタスク - 品質計画

- リスク（ハザード）管理計画
- コンフィグレーション管理計画
- ソフトウェア品質保証計画

  －ソフトウェアベリフィケーションとバリデーション計画

  □ ベリフィケーションとバリデーションタスク、受入条件

  □ スケジュールとリソース分配（ソフトウェアベリフィケーションとバリデーション活動）

  □ 要求事項の報告

  －公式なデザインレビュ要求

  －その他テクニカルレビュ要求
- 問題報告と解決手順
- その他サポート活動

5.2.2. Requirements　要求書

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.

要求の策定は、その要求の特定化、分析、機器とその使用目的に関する情報の文書を考慮する。特に重要な分野としては、ハードウェア/ソフトウェアのシステム機能の割り当てや、稼動状況、ユーザの特性、潜在的危険性、予期されるタスクがある。加えて、要求事項は、明確にソフトウェアの使用目的を記載してあること。

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では、ソフトウェア機能の定義が書き記されているべきである。ソフトウェア要求事項が事前に定義され、文書化されていない状態では、ソフトウェアをバリデートすることができない。一般的なソフトウェア要求事項は以下を明確にする：

·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.

- 全ソフトウェアシステムの入力
- 全ソフトウェアシステムからの出力
- ソフトウェアシステムで実施される全機能
- ソフトウェアが満たす、すべての性能要求（例：データ・スループット、信頼性、タイミング）
- 内部のソフトウェアシステムインターフェースの他に、外部およびユーザ等すべてのインターフェースの定義
- ユーザとシステムの相互作用の仕方
- エラーの原因と対処法
- 必須な応答時間
- 設計に制約がある場合の、ソフトウェアの稼動環境（例：ハードウェアプラットフォーム、オペレーティングシステム）
- ソフトウェアが受け入れられる全範囲、リミット、デフォルト、特定の値
- ソフトウェアに導入されるあらゆる安全関連要求、仕様、特徴、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 harm.

ソフトウェア安全要求は、システム要求開発プロセスに綿密に一体化したテクニカルリスク管理プロセスに由来している。ソフトウェア要求仕様書では、ソフトウェアに導入された安全要求の他に、システム内のソフトウェアの欠陥による潜在的な危険性も明確に特定しなければならない。ソフトウェア欠陥の結果は、これら欠陥を軽減する手段（例：ハードウェアの軽減、ディフェンシブプログラミング）を用いて評価すべきである。この分析により、危害を防ぐ際必要な、最も適切な手段を特定することが可能であると考えられる。

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:

品質システム規則は、不完全、不明瞭、また相反する要求を知らせるメカニズムを必要とする（See 21 CFR 820.30(c).）。ソフトウェア要求仕様書にて定義された各要求は（例：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、安全性）、精密性、完全性、一貫性、テスト容易性、正確性、明瞭性について評価されなければならない。例えば、ソフトウェア要求においては、以下の点を評価する：

· 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.

- 要求事項間に不整合がないこと
- システムの全性能要求が詳細に規定されている
- 耐障害性、安全性、セキュリティ要求が完全で正確である
- ソフトウェア機能の割り当てが正確で完全である
- システムの危険に対し、ソフトウェア要求が適切である
- 全要求事項が測定可能で、物理的に検証可能なものとして述べられている

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

ソフトウェア要求事項のトレーサビリティ分析では、システム要求事項に対する(からの)ソフトウェア要求事項、およびリスク分析結果までのトレースをおこなう。ソフトウェア要求事項の検証に用いられるその他分析や文書に加え、公式なデザインレビュが推奨されるのは、要求事項が完全に明記され、適切な状態であることを大規模なソフトウェア設計への取組みが始まる前に確認するためである。要求事項は承認され、追加的にリリースされるが、ソフトウェア（そしてハードウェア）要求事項での相互作用とインターフェースが適切にレビュ、分析、管理されることのケアが必要となる。

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

一般的タスク - 要求事項

- 予備的リスク分析
- トレーサビリティ分析
  ーソフトウェア要求事項からシステム要求事項へ（逆も同様）
  ーソフトウェア要求事項からリスク分析へ
- ユーザ特性の定義
- 特性のリストとプライマリー、セカンダリーメモリの制限
- ソフトウェア要求事項の評価
- ソフトウェアユーザインターフェース要求事項分析
- システムテストプラン作成
- 受諾テストプラン作成
- 不明瞭なレビュもしくは分析

5.2.3. Design　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 agreed upon requirements and design.　A complete software design specification will relieve the programmer from the need to make ad hoc design decisions.

設計プロセスでは、ソフトウェア要求仕様書では、導入されるソフトウェアを論理的、物理的な説明しなおす。ソフトウェア設計仕様書では、何をソフトウェアが行い、どのように実行するかを記載している。プロジェクトの複雑性、または広範囲にわたるテクニカルの責任者たちに、明確に設計情報を理解させるため、設計仕様書は、設計の高レベルサマリーと詳細な設計情報の両方を含むことがある。完成したソフトウェア設計仕様書によって、同意された要求事項および設計の趣旨にプログラマー/コーダー

が沿うことができる。完成したソフトウェア設計仕様書により、プログラマーはアドホックな設計の決断をする必要がなくなる。

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.

ソフトウェア設計は人的要因に対応する必要がある。過度に複雑であったり、稼動に関してユーザの予想に反する設計により生じた使用上のエラーは、FDA が直面している最も永続的で重大な問題である。頻繁に、ソフトウェアの設計はこのような使用上のエラー要因になる。人的要因に関するエンジニアリングは、機器設計要求、分析、テストなど全体的な設計・開発プロセスに盛り込まれるべきである。機器の安全性と有用性に関する問題は、フローチャート、状態図、プロトタイピングツール、テスト計画の開発時に考慮すべきである。また、タスク・機能分析、リスク分析、プロトタイプテスト・レビュ、全部の有用性テストも行うべきである。これら方法論を適用する際、ユーザ関係者は参加すること。

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 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.

ソフトウェア設計仕様書は以下を含む：

- ソフトウェアの承諾のための規定された条件など、ソフトウェア要求仕様書
- ソフトウェアリスク分析
- 開発手順とコーディングガイドライン（またはその他プログラミング手順）
- ハードウェア、ソフトウェア、物理的環境を含め、プログラムが意図したように機能するシステム状況を記載したシステム文書（例：ナラティブもしくはコンテキストダイアグラム）
- 使用されるハードウェア
- 測定、記録されるパラメータ
- 論理構造（コントロールロジックを含む）と論理的プロセスステップ（例：アルゴリズム）
- データ構造とデータフローダイアグラム
- 変数（コントロール・データ）の定義と使用される場所の概要
- エラー、アラーム、警告メッセージ
- 支援ソフトウェア（例：オペレーティングシステム、ドライバー、その他アプリケーションソフトウェア）
- コミュニケーションリンク（ソフトウェア内部モジュール間のリンク、支援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）
- セキュリティ対策（物理的セキュリティ、論理的セキュリティ）
- 上記事項に明記されていないその他追加的制約

The first four of the elements noted above usually are separate pre-existing documents that are included by reference in the software design specification. Software requirements specification was discussed in the preceding section, as was software risk analysis. Written development procedures serve as a guide to the organization, and written programming procedures serve as a guide to individual programmers. As software cannot be validated without knowledge of the context in which it is intended to function, systems documentation is referenced. If some of the above elements are not included in the software, it may be helpful to future reviewers and maintainers of the software if that is clearly stated (e.g., There are no error messages in this program).

上記、最初の四つまでの事項は、ソフトウェア設計仕様書に参照文献として盛り込まれているドキュメントとは異なるものである。ソフトウェア要求仕様書は前セクションでソフトウェアリスク分析として議論された。開発手順書は組織に対しガイドの役割を果たし、プログラミング手順書は、個々のプログラマーに対しガイドを提供する。ソフトウェアは、意図するソフトウェアの機能に関する知識をなくしてバリデートされることは不可能なので、システム文書を参照することになる。もし、上記の事項の幾つかがソフトウェアに含まれていない場合は、それらが明確に記載されることは、将来的なレビュアやソフトウェアの保守管理者にとって有用となる。（例：プログラム内にはエラーメッセージは存在しない。）

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.

ソフトウェア設計段階に行われる活動には幾つかの目的がある。ソフトウェア設計評価は、設計が完全、正確、一貫性があり、明白、適正で、メンテナンスが可能かを判断するために行われる。設計期間のソフトウェア構築（例：モジュラー構造）における適切な考慮とは、ソフトウェアの変更が求められた際の将来的バリデーションに費やす重要性を減らしていくことである。ソフトウェア設計評価は、コントロールフロー、データフロー、複雑性、タイミング、サイズ、メモリの割り当て、重大性の分析やその他設計に関する事項を含む可能性がある。トレーサビリティ分析は、ソフトウェア設計がソフトウェアの全要求を実施するものであることを証明するために実行される。十分でない要求事項を特定する技術として、トレーサビリティ分析は設計のすべての面がソフトウェア要求についてトレース可能であることも検証しなければならない。コミュニケーションリンクの分析は、ハードウェア、ユーザ、関連するソフトウェア要求事項に関して提起された設計を評価するために行われなければならない。ソフトウェアリスク分析は、設計により追加の危険が特定されていないか、新たな危険がもたらされていないかを判断するために、再検査される。

At the end of the software design activity, a Formal Design Review should be conducted to verify that the design is correct, consistent, complete, accurate, and testable, before moving to implement the design. Portions of the design can be approved and released incrementally for implementation; but care should be taken that interactions and communication links among various elements are properly reviewed, analyzed, and controlled.

ソフトウェア設計作業の最後に、設計が正確で、一貫しており、完全で、正確で、テストが可能なものであることを証明するため、設計から製造に移行する前に、正式なデザインレビュを行うべきである。設計の一部は、製造に伴い、追加的に承認され、リリースされる。しかし、様々な要素間の相互作用とコミュニケーションリンクが適切にレビュ、分析、管理されていることを配慮しなければならない。

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繰り返しをともなうことになる。つまりソフトウェア要求仕様書とソフトウェア設計仕様書が幾つかのバージョンをもつことになる可能性が高い。承認された全バージョンは、コンフィグレーション管理手順に基づきアーカイブ、管理されなければならない。

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)

一般的タスク - 設計

- アップデートされたソフトウェアリスク分析
- トレーサビリティ分析 - 設計仕様書からソフトウェア要求（逆も同様）
- ソフトウェア設計評価
- 設計コミュニケーションリンク分析
- モジュールテストプラン作成
- インテグレーションテストプラン作成
- テスト設計作成（モジュール、インテグレーション、システム、アクセプタンス）

5.2.4. Construction or Coding 構築またはコーディング

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 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.

ソフトウェアは、新規アプリケーションを使用する際、コーディング（プログラミング）または既作成ソフトウェアコンポーネント（例：ライブラリーやオフ・ザ・シェルフ・ソフトウェアなどから）の組み合わせにより構築される。コーディングは詳細な設計仕様書がソースコードとして導入されるソフトウェア活動である。コーディングはソフトウェア開発プロセスの最も低い抽象的な部分にあたる。モジュール仕様がプログラミング言語に書き換えられる、ソフトウェア要求の分解における最終局面である。

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 issues unresolved.

コーディングでは通常、高級プログラミング言語を用いるが、スピードを重視するオペレーションのためには、同時にアセンブリ言語（マイクロコード）の使用も必要とする。ソースコードは当該ハードウェアプラットフォームにあわせてコンパイルまたは実行時翻訳される。プログラミング言語とソフトウェア構築ツール（アセンブラ、リンカ、コンパイラ）の選定を決定することは、後に続く品質評価タスク（例：選択した言語のデバッキング、テストツールの可用性）への影響を考慮しなければならない。コンパイラには、コードのデバッキングを支援するエラーチェックを任意のレベルやコマンドで提供しているものもある。異なるレベルのエラーチェックはコーディングプロセスを通して使用され、コンパイラからの警告やその他メッセージは記録される場合もあれば、記録されない場合もある。しかし、コーディング、デバッキングプロセスの最終段階では、通常、最も厳しいレベルのエラーチェックを実施し、どのようなデバッグエラーがソフトウェアに残っているかを文書化する。もし最も厳しいエラーチェックをソースコードの最終翻訳に使用しなかった場合、厳密性の劣る翻訳エラーチェックを使用する正当な理由を文書化しなければならない。また最終的なデバッグとして、コンパイラからの警告やその他のメッセージ、その解決策もしくは未解決の問題を残しておく場合の正当な理由などとともに、デバッグプロセスとその成果を文書化すべきである。

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.

企業は頻繁に、ソフトウェアコーディングプロセスに関する品質ポリシーと手順を確立する特定のコーディングガイドラインを採用している。ソースコードは、指定のコーディングガイドラインを遵守していることを証明するため、評価されなければならい。ガイドラインは、明瞭さ、スタイル、複雑性管理、コメントに関するコーディングの規定を含むべきである。コードコメントには、予想される入力・出力、参照する変数、予期するデータタイプ、実行されるオペレーションなど、モジュールに関する有用な説明などの情報を提供すべきである。ソースコードもまた、対応する詳細な設計仕様書を遵守していることを証明するため、評価されなければならない。インテグレーションとテストの準備が整っているモジュールは、コーディングガイドラインとその他適切な品質ポリシーおよび手順書を遵守していることを文書化しなければならない。

Source code evaluations are often implemented as code inspections and code walkthroughs. Such static analyses provide a very effective means to detect errors before execution of the code. They allow for examination of each error in isolation and can also help in focusing later dynamic testing of the software. Firms may use manual (desk) checking with appropriate controls to ensure consistency and independence. Source code evaluations should be extended to verification of internal linkages between modules and layers (horizontal and vertical interfaces), and compliance with their design specifications. Documentation of the procedures used and the results of source code evaluations should be maintained as part of design verification.

ソースコード評価は通常、コード検査およびコードウォークスルーにより実施される。このような固定的分析は、コードを実行する前にエラーを防ぐ有効的手段を提供する。固定的分析により、個々のエラーを分離して調査し、後のソフトウェア動的テスト法に焦点をあてることが有効となる。企業は、適切な管理の下、マニュアル（机上）チェックを用いて一貫性と独立性を確保することもできる。ソースコード評価は、モジュールとレイヤー（垂直方向と水平方向のインターフェース）間の内部のリンケージのベリフィケーションまで範囲を及ばせるべきで、設計仕様書を遵守しなければならない。使用された手順書とソースコード評価結果の文書は、設計ベリフィケーションの一部として保存する。

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:

ソースコードトレーサビリティ分析は、全コードが構築した仕様書とテスト手順書にリンクしていることを検証する重要なツールである。ソースコードトレーサビリティ分析は、以下の項目を実行し文書化するものである。

·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.

- ソフトウェア設計仕様書の各要素は、コードに組み込まれている
- コードに組み込まれたモジュールと機能は、ソフトウェア設計仕様書の要素とリスク分析へトレースできる
- モジュールと機能のテストは、ソフトウェア設計仕様書の要素とリスク分析へトレースできる
- モジュールと機能のテストは、同じモジュールと機能のソースコードへトレースできる

Typical Tasks – Construction or Coding ·Traceability Analyses −Source Code to Design Specification (and vice versa) −Test Cases to Source Code and to Design Specification ·Source Code and Source Code Documentation Evaluation ·Source Code Interface Analysis ·Test Procedure and Test Case Generation (module, integration, system, and acceptance)

一般的タスク - 構築またはコーディング

- トレーサビリティ分析
  ーソースコードから設計仕様書（逆も同様）
  ーテストケースからソースコードおよび設計仕様書
- ソースコードとソースコード文書評価
- ソースコードインターフェース分析
- テスト手順書とテストケース作成（モジュール、インテグレーション、システム、アクセプタンス）

5.2.5. Testing by the Software Developer　ソフトウェア開発者によるテスト

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.

ソフトウェアテストは、予期される結果と比較できるよう、定義済みの入力と文書化した成果が存在する公の条件の下で、ソフトウェア製品を実行することが必要とされる。これは時間がかかり、困難で、不完全な活動である。従って、効率的、効果的であるよう、早期計画が必須となる。

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:

テスト計画とテストケースは、可能な限りソフトウェア開発プロセスの初期に作成することが望まれる。また、スケジュール、環境、リソース（人員、ツール等)、方法論、ケース（入力、手順書、出力、結果の期待値)、文書化、条件の報告が確認できるものでなければならない。テストプロセスを通じて費やされる労力の規模は、複雑性、重大性、信頼性、安全性の問題（例：障害の許容度の徹底的なテストにより、重大な結果を生じた機能とモジュールに要求）に関連する。ソフトウェアカテゴリーとソフトウェアテストの試みは、文献に記載されている。例えば：

·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.

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

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

ソフトウェアテスト計画書は、各開発段階で実行されるタスクを特定し、完全性の条件への対応を表す達成度の正当性を記載したものである。

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.

ソフトウェアテストにおいて、特定のソフトウェア製品のテストを計画する際、認識し考慮しておかなければならない限界がある。最もシンプルなプログラムを除けば、ソフトウェアは徹底的にテストされることはない。そもそも、全利用可能なインプットを用いてソフトウェア製品をテストすることは不可能で、またプログラム実行時のあらゆるデータプロセスパスをテストすることも不可能である。特定のソフトウェア製品が徹底的にテストされることを確実にする、テストやテスト方法論の固有のタイプは存在しない。全プログラム機能のテストは、全プログラムがテストされたことを意味するのではない。全プログラムコードのテストは、プログラムに必須となる全機能が存在することを示すのではない。全プログラム機能と全プログラムコードのテストは、プログラムが 100%正確であることを意味するのではない。エラーを発見しなかったソフトウェアテストを、ソフトウェア製品にエラーが存在しないと結論付けてもいけない。ソフトウェアテストは表面的なものである可能性があるからである。

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 effective software testing lies in the definition of what is to be tested rather than in the performance of the test.

ソフトウェアテストケースに必須な事項は、予期される結果である。それは実際のテスト結果の評価として認めるためのキーとなる詳細である。この重要なテスト情報は、対応する、事前の定義つまり仕様書から得ることができる。ソフトウェア仕様書は、エンジニアリング（測定ができる、もしくは客観的に証明できる）詳細レベルと共に、何が、いつ、どのように、いかなる理由で、などで確立するのかを、テストを通して確認できるように、特定しなければならない。有効なソフトウェアテストに関して本来試みることは、テストのパフォーマンスよりも、何がテストの対象になるのかを定義することにある。

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

ソフトウェアテストプロセスは、ソフトウェア製品の説明に有効性をもたせることを助長するという原則に基づくべきである。該当するソフトウェアテストの信条は以下を含む

- 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.

- 予期されるテスト結果が定義されている
- 良いテストケースは高い確率でエラーを発見する
- 成功するテストとは、エラーを発見するものである
- 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）の専門家が参画している
- テスターはコーダーと異なるツールを使用する
- 通例のケースのみを検査するだけでは不十分である
- テストの文書化では、テスト文書の再利用と、次に続くレビュの間、テスト結果の合格/不合格を独立して確認することができる

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.

必須条件であるタスク（例：コード検査）が成功裏に完結したら、ソフトウェアテストが始まる。ユニットレベルテストに始まり、システムレベルテストで完結する。厳密なテストレベルのインテグレーションがある可能性もある。ソフトウェア製品は、内部の構造と外部仕様書に基づいたテストケースをもとに、厳密に調べられるべきである。これらテストは、当該ソフトウェア製品が、機能、パフォーマンスおよびインターフェースの定義とインターフェース要求を遵守していることを、完全にそして厳密に説明するべきものでなければならない。

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.

コードベースのテストは、構造的テストもしくは“ホワイトボックス”テストとして知られている。それはテストケースをソースコード、詳細な設計仕様書、その他開発文書から得られる知識に基づいて特定するものである。これらテストケースは、プログラムによる制御の決定やコンフィグレーションテーブルに含まれるプログラムのデータ構成を厳密に調べるものである。構造的テストはプログラム稼動時に実行不可能な“dead”コードを認識することが出来る。構造的テストは主にユニット（モジュール）レベルテストをもって成し遂げられるが、異なるレベルのソフトウェアテストに拡張することも可能である。

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:

構造的テストのレベルは、構造的テストの間にソフトウェア構造が何パーセント評価されたかを示すよう作成されたメトリックスを用いて評価できる。これらメトリックスは通常“カバレッジ”と呼ばれ、テスト選択条件に関する完全性の尺度である。構造的なカバレッジの大きさは、ソフトウェアによって引き起こされるリスクレベルと比例しなければならない。“カバレッジ”という用語を使用する場合、通常 100%カバーされているという意味になる。例えば、テストプログラムが“ステートメント・カバレッジ“に達したということは、ソフトウェアの 100%のステートメントが最低一回は実行されたことを示す。通常の構造的カバレッジメトリックスは以下を含む

·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.

- Statement Coverage‐この条件が要求するのは、各プログラムステートメントが、最低でも一回は実行されることをみたすテストケースである。しかし、ソフトウェア製品の動作の確認においては、決して十分ではない。

·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.

- Decision (Branch) Coverage‐この条件が要求するのは、各プログラムの判定もしくは分岐が実行され、付随する結果が最低でも一回は生じることをみたすテストケースである。大部分のソフトウェア製品は最小限のレベルはカバーされていると見なされるが、decision coverage だけでは統合性の高いアプリケーションに対して不十分なものである。

·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.

- Condition Coverage‐この条件が要求するのは、各条件がプログラムの判定の際に、予期される結果すべてを最低でも一回は生じることをみたすテストケースである。決定を下す際、複数の条件を評価しなければならい時に限り、branch coverage と異なるものである。

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

- Multi-Condition Coverage‐この条件が要求するのは、プログラムの判定において、可能性のある条件の組み合わせすべてが実行されることをみたすテストケースである。

·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.

- Loop Coverage‐この条件が要求するのは、初期化、通常稼動、終了（境界）条件をカバーする全プログラムループが 0、1、2 回、そして何度も反復して実行されることをみたすテストケースである。

·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 amount of path coverage is normally established based on the risk or criticality of the software under test.

- Path Coverage‐この条件が要求するのは、定義されたプログラムセグメントのスタートから終わりまで、適切なパス、ベーシスパス等が最低でも一回実行されることをみたすテストケースである。ソフトウェアプログラムを通して可能性のあるパスは非常に大規模な数になるため、path coverage は本来達成不可能である。path coverage の数は通常、テスト中のソフトウェアのリスクもしくは重大性に基づき確立する。

・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.

- Data Flow Coverage‐この条件が要求するのは、可能な各データフローが最低でも一回実行されることをみたすテストケースである。複数のデータフローテスト計画が利用可能である。

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.

定義ベースもしくは仕様書ベースのテストは、機能テストもしくは“ブラックボックス”テストとし

て知られている。これは、ソフトウェア製品（ユニット（モジュール）であっても、完全なプログラムであっても）意図した動作の定義に基づいたテストケースであることを確認するものである。これらテストケースは、使用用途やプログラム機能、プログラムの内部および外部インターフェースをテストするものである。機能的テストは、ユニットからシステムレベルテストまでの、ソフトウェアテストの全レベルにおいて適用される。

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

以下のソフトウェアの機能的テストのタイプは、一般的に労力のレベルを上昇させることになる。

·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.

- Normal Case‐通常の入力を伴うテストが必要。しかし、予測でき有効な入力に限定してソフトウェア製品をテストすることは、ソフトウェア製品を完全にテストすることにならない。それ自体では、通常のケーステストはソフトウェア製品の独立性に関する確信を、十分に提供することはできない。

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

- Output Forcing‐選定した（もしくは全部の）ソフトウェア出力がテストで生成されたことを確実にするために、テスト入力を選択

·Robustness – Software testing should demonstrate that a software product behaves correctly when given unexpected, invalid inputs. Methods for identifying a sufficient set of such test cases include Equivalence Class Partitioning, Boundary Value Analysis, and Special Case Identification (Error Guessing). While important and necessary, these techniques do not ensure that all of the most appropriate challenges to a software product have been identified for testing.

- Robustness‐ソフトウェアテストでは、予期しない、有効でない入力を与えられた際、ソフトウェア製品が正常に動作することを実証しなければならない。このようなテストケースにふさわしいとされる方法に、同値類群分離、境界値分析、特別ケース確認（エラー推測）がある。重要かつ必要であるのに、これらテクニックは、ソフトウェア製品に対する最適な取組みの全てが、テストとみなされることを保証するものではない。

·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.

- Combinations of Inputs‐上記で特定された機能的テスト方法は、個々のあるいはシングルテスト入力を重視する。大部分のソフトウェア製品は、その使用状況において複数の入力を伴い稼動する。完全なソフトウェア製品テストは、ソフトウェアユニットやシステムが稼動時に直面する可能性のある入力の組合せを考慮しなければならない。エラー推測は、入力の組合せの特定に枠を広げることができるが、これは特定のテクニックである。原因と効果の図式化は、テストケースに含まれるソフトウェア製品への入力の組合せを体系的に特定する、機能的ソフトウェアテストテクニックである。

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 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.

機能的、構造的ソフトウェアテストケースの特定テクニックは、ランダムなテスト入力ではなく、テスト用に特定の入力を提供する。これらテクニックの弱点は、構造的、機能的テスト完了条件をソフトウェア製品の信頼性にリンクすることが困難なことである。統計的なテストのように高度のソフトウェアテスト方法は、ソフトウェア製品が信頼できるという保証を強化するために用いられる。統計的テストでは、稼動用プロファイル（例：ソフトウェア製品の予想される用途、危険な用途、悪意ある用途）を基にした定義済分布からランダムに生成したテストデータを使用する。大量のテストデータが生成され、ソフトウェア製品の設計者もしくはテスターが予期できない単一もしくは複数の稀な動作条件を特定する可能性を増やしていくことで、それらは特定の分野、懸念事項をカバーするターゲットとなる。統計的テストはまた構造的カバレッジを高める。それは安定したソフトウェア製品を必要とするものではない。このように構造的、機能的テストはソフトウェア製品の統計的テストの必須条件である。

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 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.

ソフトウェアテストの別の要素は、ソフトウェア変更に対するテストである。変更はソフトウェア開発期間に頻繁に起こる。これら変更は、

1) エラーを発見し、修正されたときのデバック
2) 新規または変更された要求事項（"requirements creep"）
3) より一層効果的、能率的な構築方法が見つかり設計を変更

などの結果である。ソフトウェア製品が一度基準化(承認)されると、その製品に対するいかなる変更は、テストを含む固有の“ミニライフサイクル”を持つものである。変更されたソフトウェア製品のテストは、追加的な試みが必要となる。変更が正確に行われたことを証明するだけでなく、変更によりソフトウェア製品のほかのパーツに悪影響を与えなかったこともあわせて証明しなければならない。レグレッション分析とテストは、変更によりソフトウェア製品のどこにも問題が生じなかったことを保証するために行われる。レグレッション分析は、実行に必要なレグレッションテストを特定するため、関連文書（例：ソフトウェア要求仕様書、ソフトウェア設計仕様書、ソースコード、テスト計画、テストケース、テストスクリプト等）のレビュに基づく変更の影響を決定することである。レグレッションテストは、プログラムが以前に正常に実行したテストケースの再実行であり、ソフトウェア変更の意図されていない影響を検出するために、現状の結果を以前の結果と比較するものである。レグレッション分析とテストは、インテグレーション方法を用いてソフトウェア製品を構築する際に使用し、新しく統合されたモジュールが以前に導入されたモジュールに対して悪影響を与えないことを保証すべきである。

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.

ソフトウェア製品の完全、厳密な検査を提供するため、開発テストは通常レベル別に行われる。例えば、ソフトウェア製品のテストは、ユニットレベル、インテグレーションレベル、システムレベルでテストが体系付けられている。

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、コンポーネント）レベルテストは、サブプログラム機能の早期検査に焦点をあて、システムレベルで見ることのできない機能がテストで検査されることを保証する。ユニットテストは、完成したソフトウェア製品への統合に対し、高品質のソフトウェアユニットが備わっていることを保証する。

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

2) インテグレーションレベルテストは、プログラムの内部、外部インターフェースへのデータ移行と管理に焦点をあてる。外部インターフェースは、他のソフトウェア（オペレーションシステムソフトウェア含む)、システムハードウェア、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説明することができる。

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) システムレベルテストは、指定した全機能が存在し、ソフトウェア製品が信頼できるものであることを証明するものである。このテストは、ソフトウェア製品に関する要求事項に関する構築されたプログラム機能とパフォーマンスが、特定のオペレーションプラットフォーム上に現れることを検証するものである。システムレベルソフトウェアテストは、機能的な懸念事項と、以下に続く意図した用途に関するデバイスソフトウェアの以下の事項に対処するものである。

·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.

- パフォーマンスに関する問題（例：応答時間、信頼性の測定結果）
- ストレス状態への対応（例：最大量読込み中の動作、連続使用）

- 内部、外部セキュリティ対策のオペレーション
- 災害復旧など、復旧手順書の効果
- 有用性
- 他のソフトウェア製品との互換性
- 各定義済みハードウェアコンフィグレーションの動作
- 文書化の正確度

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

規制措置（例：トレーサビリティ分析）は意図したカバレッジが達成されたことを証明するため行われる。

System level testing also exhibits the software product's behavior in the intended operating environment. The location of such testing is dependent upon the software developer's ability to produce the target operating environment(s). Depending upon the circumstances, simulation and/or testing at (potential) customer locations may be utilized. Test plans should identify the controls needed to ensure that the intended coverage is achieved and that proper documentation is prepared when planned system level testing is conducted at sites not directly controlled by the software developer. Also, for a software product that is a medical device or a component of a medical device that is to be used on humans prior to FDA clearance, testing involving human subjects may require an Investigational Device Exemption (IDE) or Institutional Review Board (IRB) approval.

システムレベルテストは、意図したオペレーション環境でのソフトウェア製品の動作を示すものである。このようなテストのロケーションは、目標とするオペレーション環境を整えるソフトウェア開発者の能力に依存する。状況次第では、(潜在的）顧客のロケーションにて、シミュレーションおよびまたはテストを行うことが役立つだろう。テスト計画では、計画されたシステムレベルテストが直接ソフトウェア開発者の管理しない状況で実行されたとき、意図したカバレッジが達成され、適切な文書が作成されていることを保証するために必要なコントロールを特定しなければならない。また、FDA 査察に先立って、人間に用いられる医療機器や医療機器のコンポーネントとなるソフトウェア製品に対しては、人間を対象とするテストは、Investigational Device Exemption (IDE) または Institutional Review Board (IRB)の承認が必要となる場合がある。

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、テスト結果は、対象に対し合格/不合格の決定が下せるよう文書化される。また、レビュやテストの実行後になされる客観的な決定に適しており、後のレグレッションテストにも適していなければならない。テスト中に発見されたエラーは、ソフトウェアのリリースに先立ち、ログ、分類、レビュ、解決されなければならない。開発ライフサイクル期間に回収され分析されたソフトウェアのエラーデータは、ソフトウェア製品が市販に向けてリリースに適しているかを決定するのに用いられる。テスト報告は対応するテスト計画の要求事項に適合しなければならない。

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

医療機器もしくはその製造に便利な機能をもつソフトウェア製品は、たいてい複雑である。ソフトウェアテストツールは、このようなソフトウェア製品のテストにおいて、一貫性、完全性、有効性を保証し、計画されたテスト活動内の要求事項を満たすため、頻繁に用いられる。これらのツールには、市販されているソフトウェアテストツールと同様に、ユニット(モジュール)テストと引き続き行われるインテグレーションテスト(例、ドライバーおよびスタブ) を促進する社内で構築された支援ソフトウェアが含まれる。そういったツールは開発に使用されたソフトウェアツールに同等の品質がなくてはならない。これらのソフトウェアツールの意図した用途に対してバリデーションを証明する適切な文書が維持されていなければならない(本ガイダンスの section 6 を参照のこと)

Typical Tasks – Testing by the Software Developer
・Test Planning
・Structural Test Case Identification
・Functional Test Case Identification
・Traceability Analysis - Testing
−Unit (Module) Tests to Detailed Design
−Integration Tests to High Level Design
−System Tests to Software Requirements
・Unit (Module) Test Execution
・Integration Test Execution
・Functional Test Execution

·System Test Execution ·Acceptance Test Execution ·Test Results Evaluation ·Error Evaluation/Resolution ·Final Test Report

一般的タスク - ソフトウェア開発者によるテスト

- テスト計画
- 構造的テストケース検証
- 機能的テストケース検証
- トレーサビリティ分析ーテスト
  ーユニット（モジュール）テストから詳細設計
  ーインテグレーションテストから高レベル設計
  ーシステムテストからソフトウェア要求
- ユニット（モジュール）テスト実行
- インテグレーションテスト実行
- 機能的テスト実行
- システムテスト実行
- 受入テスト実行
- テスト結果評価
- エラー評価/解決
- 最終的テスト報告

5.2.6. User Site Testing ユーザによるテスト

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.)

ユーザによるテストは、ソフトウェアバリデーションにおいて重要である。品質システム規則は、適切なインストールを証明するための検査、テストの文書だけでなく、インストールと検査の手順（適切な状況におけるテストも含む）も必要とする。(21 CFR §820.170.参照) このように、機器の製造は、指定された要求をみたし、自動化システムは、その意図する用途に対しバリデートされなければならない。(21 CFR §820.70(g) と 21 CFR §820.70(i) を各々参照)

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専門用語は、分かりにくいものである。ベータテスト、サイトバリデーション、ユーザ受入テスト、インストレーションベリフィケーション、インストールテストなどの用語は、すべてユーザサイトテストを述べる際に用いられる。本ガイダンスの目的としては、“ユーザサイトテスト”という用語は、これらのテストと開発者の管理環境以外で行われたあらゆるテストをすべて包含するものである。このテストは、ユーザサイトで、インストールされたシステムコンフィグレーションの一部となる実際のハードウェアとソフトウェアを用いて行うべきである。テストは、意図する機能の範囲内でテストされるソフトウェアを実際の使用もしくは使用をシミュレートして完了する。

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

ここに盛り込まれているガイダンスは本質的な概要で、いかなるユーザサイトテストに適用するものである。しかし、ある部分においては（例：血液構築システム)、ユーザサイトテストの計画として考慮される必要のある、特定のサイトバリデーション問題がある。テスト計画者は、ユーザサイトテストに関して、追加的規制要求がないかどうかを決断するため、FDA の該当する製品管轄の部署に問い合わせをするべきである。

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では、テストの正式なサマリーと正式な受入記録を伴った、事前に定義された書面の計画に従う。全テスト手順、テスト入力データ、テスト結果の文書による証拠は保存しなければならない。

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.

ハードウェアとソフトウェアが、指定されたようにインストール、設定された証拠がなければならない。その手段は、全システムコンポーネントがテストの最中稼動し、これらコンポーネントのバージョンは指定されたものであることを保証しなければならない。テスト計画では、オペレーション状況の全範囲にわたるテストを指定し、通常の作業中には明確にならない潜在的欠陥を発見する活動におけるさまざまな状況とイベントにシステムを遭遇させるため、連続した十分な時間を指定しなければならない。

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.

早期、開発者サイトで開発者により行われた評価の幾つかは、実用サイトで再度行うべきである。これらテストには、高容量のデータ、大量のロードストレス、セキュリティ、欠陥テスト（回避、検出、耐性、回復）エラーメッセージ、安全要求の実施を含む。開発者は、この用途で用いるテストデータセットをユーザに提供することができる。

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.

意図する機能を適切に実行するシステム能力評価に加え、システムを理解し、正確にインターフェースできるユーザ能力評価も必要である。オペレーターは、意図する機能を実行し、あらゆるアラーム、警告、エラーメッセージに対し適切にタイムリーに対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の期間は、適切なシステム性能と直面した全システム欠陥の両記録を保持する。ユーザサイトテスト期間に発見された欠陥を補うためのシステム改訂は、他のソフトウェア変更と同様の手順とコントロールに従う。

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.

ソフトウェアの開発者は、ユーザサイトテストに参加する場合もあれば、参加しない場合もある。開発者が参加した場合は、ユーザによる設計レベルシステムテストの最終部分を途切れなく繰り返す可能性がある。参加しない場合は、ユーザが綿密なテスト計画の重要性、予期するテスト結果定義、すべてのテスト出力の記録を理解する人がいることが最も重要となる。

Typical Tasks – User Site Testing ·Acceptance Test Execution ·Test Results Evaluation ·Error Evaluation/Resolution ·Final Test Report

一般的タスクーユーザサイトテスト

- 受入テスト実行
- テスト結果評価
- エラー評価/解決
- 最終テスト報告

5.2.7. Maintenance and Software Changes　メンテナンスとソフトウェア変更

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.

ソフトウェアに適用される場合、メンテナンスとは、ハードウェアに適用されるものとは同様でない。ハードウェアとソフトウェアのオペレーションメンテナンスは異なり、これは欠陥/エラーのメカニズムが異なるからである。ハードウェアのメンテナンスは一般的に、予防のハードウェアメンテナンスアクション、コンポーネント交換、修正変更が含まれる。ソフトウェアメンテナンスでは、正確、完全、適切なメンテナンスを含むが、予防のメンテナンス作業やソフトウェアのコンポーネント交換は含まない。

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や欠陥を修正するための変更は、是正メンテナンスである。パフォーマンス、保全性、またはソフトウェアシステムのその他の属性の改善するためにソフトウェアになされる変更は、最適化メンテナンスにあたる。変更された環境にてソフトウェアシステムを利用可能にするソフトウェア変更は、順応性メンテナンスである。

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).

ソフトウェアシステムが変更されたとき、初期開発期間もしくはリリースメンテナンス後の期間に、ソフトウェアの変更に関与していない部分に悪影響がないことを証明するため、十分なレグレッション分析とテストを行う。これは実行した変更の正確性を評価する追加的なテストである。

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 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.

各ソフトウェア変更に必要な特定のバリデーションは、変更のタイプ、開発製品への影響、ソフトウェア稼動時の製品への影響により決定する。多様なモジュール、インターフェース等の設計構造と相互関係の完全な文書は、変更された際、必要とされるバリデーションの試みを軽減することができる。変更に対し完全にバリデートする試みのレベルは、オリジナルソフトウェアのどのバリデーションが文書化、アーカイブされたかの度合いに依存する。例えば、テスト文書、テストケース、事前ベリフィケーション結果、バリデーションテストは、もし後のレグレッションテストに利用できるようであれば、アーカイブされる必要がある。この情報をアーカイブできない場合、変更後のソフトウェアの再バリデーションの試みのレベルと費用は明らかに増幅するだろう。

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:

標準ソフトウェア開発プロセスの一部である、ソフトウェアバリデーションやバリデーションタスクに加え、以下のメンテナンスタスクも扱われる：

·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.

- Software Validation Plan Revision - 以前バリデートされたソフトウェアに対しては、改訂されたソフトウェアのバリデーションをサポートする目的で、現行のソフトウェアバリデーション計画を改訂する。ソフトウェアバリデーション計画の前例が存在しない場合、このような計画は改訂されたソフトウェアのバリデーションをサポートできるよう作成される。

·Anomaly Evaluation – Software organizations frequently maintain documentation, such as software problem reports that describe software anomalies discovered and the specific corrective action taken to fix each anomaly. Too often, however, mistakes are repeated because software developers do not take the next step to determine the root causes of problems and make the process and procedural changes needed to avoid recurrence of the problem. Software anomalies should be evaluated in terms of their severity and their effects on system operation and safety, but they should also be treated as symptoms of process deficiencies in the quality system. A root cause analysis of anomalies can identify specific quality system deficiencies. Where trends are identified (e.g., recurrence of similar software anomalies), appropriate corrective and preventive actions must be implemented and documented to avoid further recurrence of similar quality problems. (See 21 CFR 820.100.)

- Anomaly Evaluation - ソフトウェア組織は、発見されたソフトウェアの異常や、各異常を修正すべく対応などを記載したソフトウェア問題報告書のように、頻繁に文書を保持する。頻繁すぎるのだが、ミスは繰り返される。それはソフトウェア開発者が問題が生じた根源を判定する次の措置を講ぜず、問題の再発を防ぐために必要なプロセスや手順の変更をしないためである。ソフトウェアの異常は、重大性とシステムオペレーションへの影響度および安全性に応じ評価されるべきだが、同時に品質システムではプロセスの欠陥の症状として扱われなければならない。根本的原因分析により、品質システムの欠陥を特定できる。傾向が把握できれば（例：同様のソフトウェア異常の再発)、今後同様の品質問題の再発を防ぐよう、適切な修正策や予防策が講じられ、文書化される。

·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.

- Problem Identification and Resolution Tracking - ソフトウェアメンテナンス中に発見されたあらゆる問題は、文書化される。各問題の解決は、問題が修正され、経緯と傾向が確認できるように、証拠を残す。

·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.

- Proposed Change Assessment - 提起されたすべての修正、強化および追加事項は、各変更がシステムに与える影響を判断するために評価されるべきである。この情報で、反復が必要なベリフィケーションおよびまたはバリデーションタスクの範囲を判断しなければならない。

·Task Iteration - For approved software changes, all necessary verification and validation tasks should be performed to ensure that planned changes are implemented correctly, all documentation is complete and up to date, and no unacceptable changes have occurred in software performance.

- Task Iteration - 承認されたソフトウェアの変更は、必要なベリフィケーションとバリデーションタスクが遂行されて、計画上の変更が正常に実行され、全ての文書は完結し最新版で、ソフトウェア性能において受け入れられない変更がなかったことを確認しなければならない。

・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.

- Documentation Updating - 文書は、どの文書が変更によって影響を受けたかを把握する為、注意深くレビュをする。承認済みであるが影響を受けた文書は（例：仕様書、テスト手順書、ユーザマニュアル等)、コンフィグレーション管理手順書にしたがいアップデートされる。仕様書はメンテナンスと、ソフトウェア変更以前にアップデートされる。

SECTION 6. VALIDATION OF AUTOMATED PROCESS EQUIPMENT AND QUALITY SYSTEM SOFTWARE　自動化プロセス装置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.

品質システム規則は、"コンピュータや自動化データプロセスシステムが、製品もしくは品質システムの一部として使用されるとき、(機器) 製造者が確立されたプロトコールに基づき、その意図する用途でコンピュータソフトウェアをバリデートする"ことを必要とする。(21 CFR §820.70(i)参照) これは 1978 年、FDA の medical device Good Manufacturing Practice (GMP) での規制要求である。

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 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.

上記バリデーション要求に加え、機器製造業者の製造プロセスもしくは品質システム (他の FDA 規制で必要とする記録を作成し、保持する際使用される品質システム) をインプリメントするコンピュータシステムは、電子記録、電子署名の規制の適用を受ける。(See 21 CFR Part 11.参照) この規制は、記録が電子的に作成もしくは保持された際に付加されるセキュリティ、データ統合、バリデーション要求を定めたものである。これら追加的 Part11 要求は、慎重に考慮し、システムを管理する自動化記録のために、システム要求やソフトウェ要求に含める必要がある。システムバリデーションとソフトウェアバリデーションは Part11 要求がすべて満たされていることを証明しなければならない。

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:

コンピュータや自動化装置は、医療機器設計、臨床試験・分析、製品検査・受入、製造・プロセス管理、環境管理、パッケージ、ラベル、トレーサビリティ、文書管理、苦情管理、その他品質システムに関する事項の広範囲において用いられる。徐々に、自動化プラントフロアオペレーションは、以下の領域に組み込まれたシステムまでもその範囲を広げていくことができる：

·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.

- PLC
- デジタル機能コントローラ
- 統計的プロセスコントローラ
- 監視制御とデータ収集
- ロボット工学
- ヒューマンマシンインターフェース
- 入力/出力デバイス
- コンピュータ OS

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

ソフトウェアツールは、自動化医療機器を動かすソフトウェアの設計、構築、テストで頻繁に用いられる。ワードプロセッサ、表計算ソフト、データベース、フローチャートソフトウェアなどの多くの市販ソフトウェアアプリケーションは、品質システム導入に使用される。これらアプリケーションの全ては、ソフトウェアバリデーション要求の対象となるが、各アプリケーションにたいするバリデーションアプローチは大きく異なる。

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:

製品や品質システムソフトウェアがインハウスで機器開発者により開発され、請負業者により開発され、あるいはオフ・ザ・シェルフで購入された場合であっても、本ガイダンスで説明されている基本的原則に基づいて開発されるべきである。機器開発者はソフトウェアバリデーションの遂行方法の定義に

ついては寛容そして柔軟であるが、バリデーションは、ソフトウェアがどのように、そして誰により開発され、誰から購入されるかを決定においては、主要な考慮事項である。ソフトウェア開発者は、ライフサイクルモデルを定義する。バリデーションは一般的に以下の事項からサポートを受ける：

·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ステージからの出力のベリフィケーション
- 使用済ソフトウェアで、製造者の意図する使用環境においての適正稼動チェック

6.1. HOW MUCH VALIDATION EVIDENCE IS NEEDED? どの程度のバリデーション エビデンスが必要か？

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 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:

バリデーション作業のレベルは、自動作業によりもたらされるリスクと相応する。プロセスソフトウェアの複雑性といった、リスクの他要因に加え、安全で有効な機器を製造するための自動化プロセスに機器製造業者が依存する度合いはバリデーションの一部として必要なテストの本質と範囲を決定する。自動化プロセスの文書化の必要性とリスク分析は、ソフトウェアが意図する用途おいてバリデートされていることを示すエビデンス範囲を定義する際役立つものである。例えば、自動化フライス盤に関して、機器製造業者がオペレーションの出力がリリース前の仕様書に対し引き続き完全な形で立証されることを示せば、少ないテストで済むのである。一方、広範囲に渡るテストの必要性は、以下の場合求められる：

·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.

- 工場規模の電子記録・電子署名システム
- 殺菌サイクルの自動コントローラ
- 生命維持装置で使用したサーキットボードの検査・受入用自動テスト機器

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 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).)

多くの市販ソフトウェアアプリケーションは、品質システムの一部として使用される（例：品質システム計算に用いられる表計算ソフト、統計的パッケージ、傾向分析用グラフィックパッケージ、機器履歴の記録や規制遵守管理に用いられる市販データベース)。これらソフトウェアに必要なバリデーションエビデンスの及ぶ範囲は、機器製造業者が文書化したソフトウェアの意図する用途により決定する。例えば、ベンダーの供給するソフトウェアの全てを使用しないと判断した機器製造業者は、使用する機能に限定してバリデートし、結果、機器製造業者は製造もしくは品質システムの一部としてのソフトウェア結果に依存している。しかし、ハイリスク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が使用されていない状況においても、それと同じ環境にて使用するべきではない。メモリ分離やその他リソース保護のアプローチなどのリスク軽減テクニックは、高いリスクアプリケーションと低いリスクアプリケーションが同じオペレーション環境にて用いられる際、考慮することが必要である。ソフトウェアがアップグレードしたり、何らかの変更がソフトウェアになされた場合、機器製造業者はこれら変更がソフトウェアの"使用されている部分"に対して、どのような影響を与えるかを考慮し、使用されたソフトウェア部分のバリデーションを再確認しなければならない。

6.2. DEFINED USER REQUIREMENTS　ユーザ要求定義

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

ソフトウェアバリデーションのとても重要なキーは、以下を定義するユーザ要求仕様書である：

·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.

- ソフトウェアの"意図する用途"もしくは自動化設備

- 機器製造業者が基準とする、良質の医療機器の製造に使用するソフトウェアや設備の範囲

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:

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コンフィグレーション、ソフトウェアバージョン、ユーティリティ等、予期されるオペレーション環境を定義する必要がある。ユーザは、以下の内容も必要となる：

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

- システムパフォーマンス、品質、エラー対応、スタートアップ、シャットダウン、セキュリティ等要求事項を文書化する
- センサー、アラーム、インターロック、論理的プロセスステップ、コマンドシーケンスなどの安全に関する機能、特徴を明確にする
- 受入可能な性能を決定する条件を定義する

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ールに対応して行い、バリデーションの結果は文書化されなければならない（See 21 CFR §820.70(i).）。テストケースは、事前に決定した条件、特に大部分の条件パラメータに対し、パフォーマンスを調査するシステムで実行するよう文書化される。テストケースは、エラーやアラーム状態、スタートアップ、シャットダウン、全使用可能なユーザ機能、オペレーターコントロール、潜在的オペレーターエラー、許容値の最大・最小範囲、装置の意図する用途に適用するストレス条件に対処すべきものである。テストケースは実行され、その結果は記録され、評価されて、その結果がソフトウェアが意図する用途に対してバリデートされたという結論を裏付けるかどうかを判定する。

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:

機器製造業者は、自社の社員を使って、あるいは装置/ソフトウェアベンダーやコンサルタントのようなサードパーティーに依存してバリデーションを行ってもよい。どのような場合でも、機器製造業者は製品と品質システムソフトウェアを以下の条件を満たすことを保証する最終的責任を負うことになる。

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

- 意図する用途に対する手順書に沿ってバリデートされる
- 選定したアプリケーションで意図する性能をする

The device manufacturer should have documentation including:

機器製造業者は以下の事項を含む文書を持つ：

·defined user requirements;
·validation protocol used;
·acceptance criteria;
·test cases and results; and
·a validation summary

- 定義されたユーザ要求
- 使用されるバリデーションプロトコール
- 受入条件
- テストケースと結果
- バリデーションサマリー

that objectively confirms that the software is validated for its intended use.

そのことにより、ソフトウェアが意図する用途に沿ってバリデートされたことを客観的に確認できる。

6.3. VALIDATION OF OFF-THE-SHELF SOFTWARE AND AUTOMATED EQUIPMENT オフ・ザ・シェルフ・ソフトウェアと自動化装置のバリデーション

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 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.

機器製造業者により使用される自動化装置やシステムのほとんどは、サードベンダーにより提供され、オフ・ザ・シェルフ（OTS）で購入される。機器製造業者は、OTS ソフトウェア開発者により使用される製品開発方法論が、OTS ソフトウェアが機器製造業者の意図する用途に対して適切で十分であることを保証することに責任を負う。OTS ソフトウェア・装置では、機器製造業者が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求、ソフトウェア要求、バリデーションプロセスおよびバリデーション結果の情報を提供できる場合、医療機器製造者は、それら情報を彼らに要求されるバリデーションドキュメントの開始点として使用することができる。テストプロトコールや結果、ソースコード、設計仕様書、および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ていることを確定するのに役立てることができる。しかし、市販装置ベンダーからのそのような文書は利用不可能、もしくはベンダーが占有する情報の共有を拒否するであろう。

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.

機器リスクの可能性や依存性がある状況においては、機器製造業者が、OTS ソフトウェア構造で用いられるベンダー設計や開発方法論のオーディットを考慮し、OTS ソフトウェア用に作成する開発およびバリデーションドキュメントを査定する必要がある。このようなオーディットは、機器製造業者や資格ある第三者機関により行われる。オーディットは、ベンダーの手順およびソフトウェアを用いて製造した医療機器の安全性および効率性要求をみたす、OTS ソフトウェアで実施されたベリフィケーションおよびバリデーション作業の結果が、適切で十分であることを証明しなければならない

Some vendors who are not accustomed to operating in a regulated environment may not have a documented life cycle process that can support the device manufacturer's validation requirement. Other vendors may not permit an audit. Where necessary validation information is not available

from the vendor, the device manufacturer will need to perform sufficient system level “black box” testing to establish that the software meets their “user needs and intended uses.” For many applications black box testing alone is not sufficient. Depending upon the risk of the device produced, the role of the OTS software in the process, the ability to audit the vendor, and the sufficiency of vendor-supplied information, the use of OTS software or equipment may or may not be appropriate, especially if there are suitable alternatives available. The device manufacturer should also consider the implications (if any) for continued maintenance and support of the OTS software should the vendor terminate their support.

規制を受けた環境でのオペレーションに慣れていないベンダーは、機器製造業者のバリデーション要求をサポートするライフサイクルプロセスのドキュメントを持っていない場合もある。あるベンダーはオーディットを許可しないかもしれない。ベンダーからの、必須なバリデーション情報を入手できない場合、機器製造業者は、ソフトウェアが“ユーザニーズと意図する用途”を満たすと証明するため、需要なシステムレベル“ブラックボックス”テストを行う必要がでてくる。多くのアプリケーションにおいて、ブラックボックステストだけでは十分ではない。製造されたデバイスのリスクによって、プロセスでの OTS ソフトウェアの役割、ベンダーオーディット能力、ベンダー提供の情報、OTS ソフトウェアもしくは装置の用途は、適切とみなされる場合もあれば、特に、代替の選択肢が適している場合はそうでない場合もある。機器製造業者は、継続するメンテナンスに対する結果とベンダーがサポートを終了する場合の OTS ソフトウェアサポートとの関りあいをについて考慮しなければならない。

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.

ソフトウェアのコンパイラ、リンカー、エディター、オペレーションシステムなどオフ・ザ・シェルフ・ソフトウェア開発ツールに対し、機器製造業者による徹底的ブラックボックステストは、実用的でないかもしれない。そのようなテストなしに、バリデーション作業の重要な要素は、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、適切なオペレーションは、他の方法で十分に推測されるかもしれない。例えば、コンパイラは独立した第三者テストに保証され、市販ソフトウェア製品が“バグのリスト”、システム要求、そしてベンダーから入手可能なその他オペレーション情報、この常用は機器製造業者の意図する用途と比較することで、ブラックボックス”テスト作業に焦点をあてるの

に有用である。オフ・ザ・シェルフ・オペレーションシステムは個々のプログラムとしてバリデートされる必要はない。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バリデーションテストは、アプリケーションプログラムの意図する用途に適した最大読込み条件、ファイルオペレーション、システムエラー条件対応、メモリ構造など使用される全オペレーションシステムに対処しなければならない。