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

序文

パブリックコメント

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。

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。

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。

追加コピー

CDRH

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

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

ソフトウェアバリデーションの一般原則

本ドキュメント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。

1. 目的

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。

2. 範囲

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、本ドキュメント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。

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。

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。

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。

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。

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 適用性

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）

本ドキュメント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。本ドキュメントは、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 オーディエンス

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 最小限の負荷となるアプローチ

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、法的要件を反映し、これら要件に従うことが、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 ソフトウェアバリデーションの規制要件

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。

ソフトウェアバリデーションは品質システム規則の要件で、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）バリデーション要件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。

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）。この要件は、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件は 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。

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。この要件は、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、クレーム対応、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。

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、バリデーション要件の対象となる（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。

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件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。

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、本ドキュメント 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。

2.5　品質システム規則と市販前申請

本ドキュメントでは、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件と混同してはならない。

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件に準拠する為、同じ手順と記録を使用することもある。本ドキュメントは、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件は、本ドキュメント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。

3. ソフトウェアバリデーションの背景

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。本ドキュメント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。

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件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 定義と専門用語

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。

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。

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、要件、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 要件と仕様

品質システム規則は、設計のための要件は文書化されなければならない、そしてその特定の要件は検証されなければならないと述べている一方で、“要件”と“仕様”の違いをそれほど明確に述べていない。**要件**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。要件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、組織内部の要件でもあり得る。多種多様な要件書（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、物理的要件）が存在する。ソフトウェア要件は、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件から生成される。ソフトウェア要件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。ソフトウェア要件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。

仕様という言葉は、“要件を記述した文書”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、それによって要件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　ベリフィケーションとバリデーション

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。

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件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。

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、特定の要件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件が正確かつ完全にインプリメントされ、システム要件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。

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

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。

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、本ドキュメント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 システムデザインの一端となるソフトウェア開発

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。ソフトウェア要件は一般的に総合的なシステム要件と、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件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、これらの要件がハードウェア、ソフトウェア、もしくは両方を組み合わせたものが要件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。

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件を遵守していることを証明することである。システム要件とソフトウェア要件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、全ソフトウェア要件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 ソフトウェアはハードウェアと異なる

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：

- 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。

- 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。
- 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。
- 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。
- 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。
- 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。**
- 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。
- 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。
- 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。

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　ソフトウェアバリデーションの利点

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　設計レビュ

設計レビュは、文書化され、理解しやすく、体系的な調査であり、設計要件の妥当性や、その要件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、公式な設計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。設計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。設計レビュは、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。

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、公式設計レビュにより、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設計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、複数の設計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、公式設計レビュは要件活動の最終局面時、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。

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

- 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？
- 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件を遵守しているか？
- ✓ 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？

4. ソフトウェアバリデーションの原則

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 要件

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 欠陥の回避

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 時間と労力

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 ソフトウェア ライフサイクル

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 計画

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　手順書

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　変更後のソフトウェアバリデーション

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　バリデーション範囲

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　レビュの独立

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 柔軟性と責任

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。

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：

- 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

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。

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。

5. 活動とタスク

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 ソフトウェア ライフサイクル活動

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：

- 品質計画
- システム要件定義
- 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

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 標準的タスクサポートバリデーション

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 品質計画

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、正式な設計レビュも含むマネージメントレビュ要件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：

- 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
- 入力要件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

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。

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。

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
  - ー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
  - ー公式な設計レビュ要件
  - ーその他テクニカルレビュ要件

- 問題報告と解決手順
- その他サポート活動

5.2.2 要件書

要件の策定は、その要件の特定化、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。

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：

- 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、すべての性能要件（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件、仕様、特徴、機能

ソフトウェア安全要件は、システム要件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件の他に、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。

品質システム規則は、不完全、不明瞭、また相反する要件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件は（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、ソフトウェア要件においては、以下の点を評価する：

- 要求事項間に不整合がないこと
- システムの全性能要件が詳細に規定されている
- 耐障害性、安全性、セキュリティ要件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、ソフトウェア要件が適切である
- 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

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、公式な設計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。

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 設計

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。

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、機器設計要件、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。

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

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。)

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件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件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。

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。

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。

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件（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 構築またはコーディング

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、ソフトウェア要件の分解における最終局面である。

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。

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。

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。

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。

- 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

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 ソフトウェア開発者によるテスト

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。

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

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。

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。

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。

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

- 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

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件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。

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。

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‐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‐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‐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‐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‐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‐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‐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。

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。

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‐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‐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‐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‐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。

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。

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。

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) 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) 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) 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。

- 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

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。

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。

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。

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)

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件
- 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　ユーザによるテスト

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、指定された要件をみたし、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)

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。

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、追加的規制要件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。

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。

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。

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、安全要件の実施を含む。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。

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。

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。

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。

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　メンテナンスとソフトウェア変更

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。

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。

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。

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。

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 - 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 - 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 - 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 - 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 - 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 - 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。

6. 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

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) での規制要件である。

上記バリデーション要件に加え、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、バリデーション要件を定めたものである。これら追加的 Part11 要件は、慎重に考慮し、システムを管理する自動化記録のために、システム要件やソフトウェ要件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 要件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。

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：

- 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

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件の対象となるが、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。

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：

- 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　どの程度のバリデーション エビデンスが必要か？

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：

- 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

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　ユーザ要件定義

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：

- 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

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：

- 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

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。

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。

- 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

機器製造業者は以下の事項を含む文書を持つ：

- 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

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　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

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件、ソフトウェア要件、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。

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件をみたす、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

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件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。

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”、システム要件、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。