4.3.2.1 ソフトウェア障害リスクの分析

前述の通り、全体的なプロセスには、リスク分析が検証プランニングの段階に必要な情報を提供する2つのポイントがある。最初のリスク分析はプロセス障害の分析を目的としているのに対し、二回目のリスク分析はソフトウェア障害の潜在的なリスクの特定を目的としているため、ソフトウェアソリューションの選択後に初めて遂行可能となる。この分析の要点は、ソフトウェア障害に伴う本質的なリスクを特定・記録し、下流の予防策（プロセス及びソフトウェアの管理を含む）を特定することである。この分析を利用して現実的かつ効果的な検証アプローチに到達する。

ソフトウェア障害に起因するリスクを評価する場合、リスク予防策を構成する下流のプロセスコントロールを忘れずに考慮する。これらのリスク予防策は、ソフトウェア障害の影響を低減、すなわちソフトウェアへの依存度を減らすことで、ソフトウェアの安全な操作を確保するための試験（検査）及び文書作成（客観的証拠の収集）への依存度を低減する。また、プロセスをマニュアル管理からソフトウェアによる自動化へ移行させる行為は、そのプロセスに伴うリスクに本質的なかたちで影響を及ぼす。このような変更は、新たなリスクを生んだり、既存のリスクを予防したり、リスク発生率に影響を及ぼす可能性がある。これらを考慮することで、ソフトウェアがプロセス全体のコンテクストの中で取り扱われるようにする。

ソフトウェアのリスク分析を実行するためのモデルを付録Bに掲載した。このモデルは、包括的な基本原則を示すものではない。このような分析を実施すれば、ツールボックスからソフトウェアの検証に使用するツールを選択する際に必要な情報を得ることができる。分析のクオリティは、分析実施担当者の経験に比例すると考えられる。この活動には、ソフトウェア品質技術の豊富な経験を有する人材が必要である。

4.3.2.2 検証プランニング

検証プランニング活動では、意図する使用の定義及びソフトウェアリスク分析の結果を、リスク予防策を特定し、ツールボックスからのソフトウェア検証用のツールを選択する際に必要な情報として利用する。

どのようなツールが選択されるかは、担当者の経験及びスキルによって異なる。障害が自動化するプロセスに及ぼす影響を理解する有資格の担当者をツール選択のプロセスに参加させることが重要である。そのような担当者は、自動化するプロセス及びそのプロセスを自動化するソフトウェアの障害がもたらす本質的なリスクを理解していることが重要であり、必ずしもソフトウェアの専門家である必要はない。非常に複雑なソフトウェア、又は障害に伴う高リスクがあるソフトウェアのプランニングプロセスには、さまざまな分野（規制、品質、臨床など）の関係者が参加しなければならない。

検証のアプローチは、システムを構成するアプリケーションの段階的なリリース又は複数回にわたるリリースを実行する必要性に応じて異なる。段階的なリリース拡大の場合、セクション 4.4「保守段階」で説明する保守の方法論を利用する（検証済み機能を拡張する際の統計学的なプロセス管理システムなど）。複数回にわたるリリースの場合、システムの共通要素をカバーする基本的な検証を実施してリリースを行う。その場合、特定の意図する使用に基づくそれぞれの検証の基礎として、基本的な検証パッケージを利用する（各種生産ライン用に構成された基本的な生産実行システムなど）。

特定の個人が自分の業務に影響するという理由で障害の結果を懸念しながら、他の場所で発生した障害の影響にまったく気づいていないという事態も珍しいことではない。例えば、製造技師が製造プロセスへのリスク（ビジネスリスク）を懸念しながら、製造上の障害が臨床的な安全性のリスクとして発現することを予測するための見識を持たないこともあり得る。検証プランナーが自らのスキルと経験を踏まえ、同じソフトウェアに他のプランナーと異なるツールを選択する可能性もある。ソリューションがソフトウェアのパフォーマンスにおける信頼醸成の目標を達成している限り、いずれのソリューションも容認できると考えられる。

検証プランニング活動の成果として、選択の結果（決定事項）及び選択の理由（決定推進要因）を説明した計画書が完成する。これは、ソフトウェアが意図した通りに機能することを保証する目的で付加価値のある信頼醸成活動を選択する際に使われる根拠を示す証拠文となる。

4.3.2.3 ソフトウェアのインプリメンテーション（設計／開発／構築／試験）

このブロックには、ツールボックスから選択した各種ツールの実践的応用、及びIEEEなどの標準機関が定義した通常のインプリメンテーション活動が含まれる。前述の通り、さまざまな開発方法又はライフサイクルモデルを使って、効果的にソフトウェアのインプリメンテーションと導入を行うことができる。ここではウォーターフォール型のシーケンスを使って活動を紹介するが、これはあくまでも説明を簡単にするのが目的である。この TIR で説明しているリスクマネジメント/検証プランニング/批判的思考の概念が選択されたライフサイクルのコンテキストで応用されている限り、反復型、らせん型及びその他の有効なライフサイクルアプローチを利用することができる。

4.3.2.4 検証レポート

ソフトウェアが意図した通りに機能することを保証する上で十分な信頼醸成活動（ツールボックスからのツール選択など）が完了したら、活動及び活動の成果を最終的な検証レポートに記載しなければならない。このレポートが正式に審査・承認されれば、ソフトウェアがその意図する使用について検証されているという結論を裏付けるため

に文書化されたすべての客観的証拠をまとめた概要及びそのリファレンスがもたらされる。

4.3.2.5 ソフトウェアのリリース

ソフトウェアが意図した通りに機能し、規制プロセスに容認不能なリスクをもたらさないという結論に達したら、ソフトウェアをリリースするための正式な管理された方法が必要となる。定義された管理方法は、採用されたソフトウェアが検証レポートに記載された信頼醸成活動で評価されたソフトウェアに適合することを保証し、確認するものでなければならない。一方、リリースされたソフトウェアが（試験のシミュレーション、ハードウェアの制約、又はその他の環境的な制約などが原因で）検証済みのソフトウェアに完全適合しない場合、その根拠及び管理によって意図する環境におけるリリース済みソフトウェアの性能を十分に示せるような結果を保証・確認しなければならない。

4.4 保守段階

最終的な規制プロセスの環境で使用する目的でソフトウェアがリリースされたら、次はそのソフトウェアのライフサイクルにおける保守段階に入る。保守段階の活動では、さまざまな変更に対応し、その管理及びコントロールを行いながら、ソフトウェアが検証済みの状態を維持できるようにする。ソフトウェアを使用するプロセス内に変更が生じるだけの場合もある。

検証済みシステムを変更する場合、方針及び手順に準じて監督された方法で行わなければならない。理想としては、システムを実際の製造で使用する前に、試験環境で変更及び検証を行うことが望ましい。それが不可能で変更の試験を製造環境で行わざるを得ない場合、製造環境や製品への悪影響を最低限に抑えるために適切な対策を講じなければならない。

変更の検証用にツールボックスから選択するツールは、ソフトウェアの変更が既存のリスク予防策、新たなリスクの発生、又はその両方に及ぼす影響を分析した上で決定する。また、対策を講じても、ソフトウェアの実際の使用又はその構成は時間の経過と共に変化するため、実際の使用状況の定期的なモニタリング又はソフトウェア構成のリアルタイムのモニタリングなど、保守段階専用のツールを使用することができる。意図する使用の変更によりリスクのレベルが上昇する場合、ソフトウェアに変更がなくても、その変更がきっかけで当初行われていたものよりも大規模な検証活動が必要となる可能性もある。このような活動実行の選択及び根拠に関する決定を、検証プランニングの一環として文書に記録し、ソフトウェアが検証済みの状態を維持していることを裏付ける証拠としなければならない。

4.4.1 保守のプランニング

理想的には、開発段階のうちに保守プランニングに着手することが望ましい。変更がソフトウェアの検証にどのような影響を及ぼすかを正しく理解し、変更がリスクに及ぼす影響を調査し、検証済みの状態を維持する上で適切な活動を計画しなければならない。大規模かつ複雑なソフトウェアは、意図された任務遂行能力を損なうことなく、日常的な保守及び性能調整活動に対応可能なものでなければならない。開発段階における保守のプランニングでは、これらの業務活動のうち検証に影響を及ぼさずに遂行可能なものがどれで、検証努力を必要とする変更がどれかを定義することができる。保守段階に到達する前に、基本ソフトウェア（オペレーティングシステム、データベース管理システムなど）での変更が検証済みソフトウェアにどのような影響を及ぼすかを含め、さらにソフトウェアの検証活動を行う時期の決定方法について計画し、審議しなければならない。これらの境界を認識し、通常業務活動と検証を要する変更の違いを理解してもらえるように、ソフトウェアオペレーターを訓練するのも有用である。

トレーサビリティ分析は、保守活動を管理する上で有用なツールである。初回検証の基礎として頻繁に利用され、トレーサビリティマトリックスを使って行われることが多い。このマトリックスは、試験又は他の検証活動、リスク予防策などの要件をマッピングする。初回のインプリメンテーションで成功すれば、このマトリックスは、変更及びその適切な検証活動がもたらす影響の特定を促し、保守段階で有用なツールとなる。単純なソフトウェアの場合、これはインプリメンテーション及び検証に関する要件の単一レベルのトレースとなる。一方、複雑なソフトウェアの場合、上位レベルの機能を下位レベルの要件に分解した後、さらにインプリメンテーションと検証に分解する複数レベルのマトリックスが必要となる。それ以外の情報を組み込むこともできる。例えば、特に高リスクと考えられるソフトウェアのセクションをトレースマトリックス内で指定することができ、追加の検証活動を指示することも可能と考えられる。

4.4.2 保守段階で行われる保守のタイプ

リリース後にソフトウェアが変更される理由はさまざまである。保守による変更で特に多いタイプとして以下の例が挙げられる。

- ソフトウェアのエラー及び欠点を是正するための改良保守による変更
- ソフトウェアの性能、保守性又はその他の属性を向上させるための完全化保守による変更
- ソフトウェアの操作環境を更新するための適応保守（オペレーティングシステム又はシステムハードウェアの変更など）

4.4.3 プロセス変更 ― リスク予防策の変更

ソフトウェアによって全部又は一部が自動化されるプロセスが、ソフトウェアとは別に変更される場合がある。プロセスの変更が生じた場合、それがソフトウェアの検証された状態にどう影響するかを理解することが重要である。プロセスの変更は、ソフトウェアの意図する使用又はソフトウェアに関するその他のサポート情報に影響を及ぼす。また、プロセスの変更がソフトウェアのために設けられたリスク予防策に影響を及ぼすこともあり、それが検証根拠の一部にもなっている。ソフトウェアはプロセスの一部であることから、下流の管理がソフトウェアの重要なリスク予防策になると考えられる。もし、それらがソフトウェア検証原理及びプロセス定義の一部として適切に特定されれば、提案されたプロセス変更のインパクト解析を実施しやすくなる。この分析は、ソフトウェア及びそのソフトウェアが稼働するプロセスの両方で信頼を醸成するようなやり方で保守を実行する際には不可欠である。

4.4.4 緊急の変更

緊急を要する状況でソフトウェアの変更が必要になることもある。通常、このような変更が必要になるのは、ソフトウェア・オペレーティングシステム、又はデータの完全性が損なわれたり、潜在的に有害な状況の緩和を促進する場合である。

緊急の変更は、承認されたプロセスに準ずるものでなければならない。これらのプロセスは、開発及びインプリメンテーションの正当化、変更導入に対する許可の獲得及び記録のメカニズム、リスクが適切に評価・管理されていることを確実にするための準備、及び緊急の変更を実施するために必要なあらゆる活動（トレーニング、広報、製品レビュー、処分など）を要求するものでなければならない。このような状況で、リスクを適切に評価・管理するための準備を実施する行為とは、リリース前の変更検証に関する規制要件に適合する活動の最低限の組み合せを意味する。また、変更がもたらすあらゆる影響を徹底評価するために、急変更後の活動が必要になることもある。自動化されたプロセスの障害がもたらす全体的なリスクに応じて、緊急変更後の活動がすべて完了するまで、プロセスのアウトプット（データ又は製品）をさらに管理しなければならない場合もある。進行中のモニタリング

自動化されたプロセスを妨害するソフトウェアの問題には明白なものが多い。とらえどころのない潜在的な問題を見つけ出すのは、さらに困難である。エラーログ、ヘルプセンターの要請、顧客の苦情、及び他の欠陥レポートを定期的に評価することで、潜在的な問題を突き止めることができる。これらのモニタリング技術は、エラーレポートに表れるほど明白ではないが、修正可能なソフトウェアの欠陥を示唆する問題を拾い上げることができる。このようにして特定された問題に対処するためには、保守活動が必要となる。将来的なリリースのために問題を修正するだけでなく、リリース

後のソフトウェアに特定された欠陥が過去に及ぼした影響を評価し、結果を管理しなければならない。

トレーニングによるソフトウェアの正しい使い方の定着がソフトウェア検証の重要な部分を占めている場合、ユーザートレーニングの効果を定期的に評価することも、検証された状態を維持する上で有用なモニタリング技術である。

4.4.5 意図する使用の保守

意図する使用の変更は、微妙かつ発見が困難な場合もあれば、かなり明白な場合もあるため、特別な注意を要するカテゴリーである。微妙な場合、目的及び意図又はソフトウェアの使用に関する要件に変更が生じるが、必ずしもソフトウェアに関する要件の詳細が変更になるとは限らない（セクション 4.3.1.4 参照）。このタイプの変更は、意図的に生じる場合もあれば、意図する使用が影響を受けていることを知らずに新しいモードで既存のソフトウェアを単に使用した結果として生じる場合がある。意図する使用が時間の経過に伴って当初の意図を逸脱したり、ユーザーが当初の意図と異なる方法でソフトウェアを使いはじめることもある。このようなシフトが原因で、導入されたソフトウェアは検証された状態を維持できなくなる。その場合、新しい意図する使用を検証したり、新しい使用を中止しなければならない。後者の場合、リスク評価の目的は、許可されていない方法での使用期間中にリスクが発生しないようにすることである。検証済みソフトウェアに変更が加わる度に意図する使用を再検討し、それがソフトウェアの実際の使用に即したものかどうかを確認しなければならない。

4.5 廃用段階

廃用段階では、ソフトウェアの運用中止を文書に記録し、規定の記録保持期間中に関連のある電子記録にアクセスする方法を確立することが目標となる。

ソフトウェア廃用活動は、運用を中止するソフトウェアのタイプに大きく依存する。ソフトウェアの中には、特定の活動を実行するだけで、データを保存しないものもある。また、ロットトレーサビリティ又は文書管理のシステムのように、製品やコンプライアンスに関連する膨大なデータを保存する複雑なソフトウェアもある。データを保存するソフトウェアの場合、データの取り扱い方法に関する計画が必要である。考慮すべき事項として、以下のものが挙げられる。

- 廃用となるソフトウェアに替わる新しいソフトウェアはあるか。
- 新しいソフトウェアへのデータ移動は可能か。
- 長期保存用の移植可能なフォーマットでデータを移動させる必要はあるか。
- データのタイプに応じたデータ保持の要件は何か。
- データは耐久性のあるメディアに保存されるか。

- これに該当する場合、保存の指示又は手順はどのようなものであり、関連するすべてのデータ要件を含むデータ検索は可能か。
- 耐久メディアの保守方法、及びそれを読み取り可能なソフトウェアはどのようなものか。
- アーカイブ化されたハードウェアプラットフォームは、廃用になったアプリケーションを使用・検索できるように保存されるか。
- 保存されたハードウェアはどのようにして保守が行われるか。
- 苦情調査又はCAPA調査の一環として、廃用になったソフトウェアにアクセスする必要が生じる可能性はあるか。
- プラットフォーム及びアプリケーションはソフトウェアプログラムを開発し直す必要があるか。

5 文書作成

ソフトウェアのライフサイクルコントロール活動に関連するあらゆる情報を、適切な方法で確実に文書化することが不可欠である。高クオリティかつ効率的な文書作成の便益には、主に2つのタイプがある。

1. ソフトウェアの定義を明確に文書化すれば、その意図する使用、期待される性能、及びソフトウェアに加えられたあらゆる変更がもたらす影響をを徹底的に理解することができる。
2. 検証プランニング及び検証の実施を記録すれば、それが批判的思考の結果として決定された事項の証拠文書となる。実行された評価/分析及びその結果であるリスクを踏まえた有意義な信頼醸成活動のためのツール選択に関してこのような文書作成を行えば、実行された検証を簡潔に理解することができる。許容基準の適合状況をまとめて文書化すれば、遂行された活動がソフトウェアの意図する使用を確実なものにし、ソフトウェアによって自動化されるプロセスにもたらされるリスクが許容可能なレベルであることを裏付ける証拠となる。

作成する文書の範囲は、ソフトウェアの検証に適用される努力のレベルに直接関連する。努力のレベルはリスクに見合うものでなければならない。従って、このTIRで紹介するソフトウェア検証のアプローチでは、自動化されるプロセスの障害がもたらす影響に基づいて文書化の範囲を決定する。自動化されたプロセスがもたらす人体や環境への危害のリスクが大きくなるほど、文書化の範囲も大きくなると予想される。また、危害のリスクが大きくなるに従い、各種部門の同僚又は社内の上級管理職、若しくはその両方の協力によって文書精査を厳密化する必要がある。

文書化するライフサイクルコントロール情報の構成は、利用するテクノロジー及びソフトウェアのサイズ/複雑さなど、さまざまな要因によって異なる。情報の整理は、ソフトウェアライフサイクルの保守段階で検証された状態の証拠を保守する能力を助成しながら、情報の監査をサポートするような方法で行われなければならない。ライフサイクルコントロール情報の収集・文書化の方法は、検証を実施する担当者の嗜好及び既定方針によって異なる。ライフサイクルコントロールの客観的証拠をどのようにまとめ、表示するかの決定は、ソフトウェア検証担当者の裁量に委ねられる。コンプライアンス審査の観点から、検証プランニング及び報告文書作成は、ソフトウェアが意図した通りに機能することを保証するために計画及び実行された付加価値のあるすべての信頼醸成活動を編集できるように確立されなければならない。これは基本的に、規則の意図に即し、主な関係者及びそのニーズのすべてを考慮した完全なソフトウェアソリューションが開発されたことを確認するために、批判的思考のプロセスを採り入れたインプット（決定推進要因）に基づいて行われた選択（決定事項）を記録した重要な文書といえる。

備考 － “文書化（documentation）”の用語は、実際の書類又は情報をキャプチャーするツール（要件管理ツールなど）に記録された情報の本体を意味する。

6 必須プロセス

このTIRで説明した方法で最大の効果を得るためには、確実な品質システムを用意することが重要である。品質システムの側面のうち、批判的思考を採り入れた方法の成功に最良の影響を及ぼすものとして、資産/インフラ管理（人材及びハードウェア）、変更管理（構成管理を含む）及び販売業者管理が挙げられる。これらのプロセスは、業界内の他の規格及び文書に関するものであるため、このTIRの適用範囲外となる。このTIRは、特定の役割又は機能（品質保証、管理、及び製造）をここで紹介した活動と関連づけることを意図するものではない。検証活動を実施する上で許容可能な役割は、各社の理念及び人材インフラによって必然的に決まるだろう。

付録 A
ツールボックス

このツールボックスでは、検証に関する規則の意図を満たす上で利用可能な信頼醸成ツールの一覧を紹介する。この目的に利用できる活動をすべて網羅したリストではないが、最新のソフトウェアエンジニアリングに関する一連の知識に基づくツールの基本セットといえる。これらのツールの中には、重複するものや併用されるものもあるが(例えば、ノーマルケース試験はソフトウェアシステム試験に含まれることが多い)、ここではツールの価値に重点を置いている。これらのツールは、検証プランニング及び実施の基礎として使用される。ツールの選択及び利用は、ソフトウェアに関連するリスクに見合った適切なものでなければならない。

自社のツールボックスをカスタマイズして社内で使用するツールを定義すれば、時間の経過と共にテクノロジーが進化し、教訓を得るに従い、新しいソフトウェアエンジニアリングのベストプラクティスを採り入れる目的で、ツールを進化させることができる。場合に応じて、活動の中には、手続き上、標準手順で動員されるものもある。

ツールボックスの構成

便宜上、ツールは主な 5 つのソフトウェアライフサイクルのプロセス活動に分類される。ソフトウェアの適用範囲及び性質に応じて、ソフトウェアライフサイクルのさまざまな段階で批判的思考を応用し、そのソフトウェアに最適なツールを特定・選択する。

リストには活動（又はツール）の名称を記載し、それぞれの活動が検証努力にもたらす価値に関する簡単な定義及び説明を紹介する。定義欄には、その活動を遂行する上で利用可能な方法の例も記載する。

ツール及びその価値

品質システム規則の設計管理に関する部分（21 CFR 820.30）は、適切な設計管理の欠如に起因する一連の現場障害に対処することを目的の一つとして、FDA によって発効されている。同様に、ソフトウェアに関しては、信頼性の向上を目的としたソフトウェアエンジニアリングの慣行で広く知られているものがいくつか存在する。これらの慣行及びそのソースは、FDA の“General Principles of Software Validation（ソフトウェア検証の一般原則）”のリファレンスセクションに記載されている。

1 開発段階 ― 定義		
活動	定義	価値
プロセス要件の定義	ソフトウェアによる部分的又は全面的な自動化について検討しながらプロセスを定義する。この場合、プロセスとしては、製造プロセス又は品質システムプロセスが考えられる。プロセス要件の定義では、プロセス又はソフトウェアのリスク分析を行う際に検討されるプロセス内における検証又は予防策についても説明する。 この活動からのアウトプットは以下のいずれか一つ又は両方で文書化される。 • プロセスフロー図 • ビジネス、製造、又は品質システムのプロセス内で行われた活動を定義する要件書	ライフサイクルの後の方で決定される事項の基礎を確立し、ソフトウェアの使用に関するコンテクストを定義する。プロセス内におけるソフトウェアと上流及び下流の活動との境界を明確にする上で役立つ。 ビジネスプロセスの要件を“ユースケース”に変換し、それを利用してソフトウェアの詳細な要件及び受入試験のための試験例を作成できる。 プロセス要件は、ソフトウェア販売業者を選定する際、業者から提供されたテストスクリプトのベースとして利用できる。
プロセス障害リスク分析	プロセス障害が機器の安全性/効率性、製造担当者、環境、又は品質システムに及ぼす影響を判定する。 プロセス障害リスクは、FMEAなどの公式な方法、又はプロセスに障害がある場合に発生する可能性がある危害のタイプを簡単なリストにまとめるなどの非公式な方法で判定できる。	プロセス障害の影響及び重度を確定し、それによりソフトウェア開発及び検証努力のレベルに関する決定を促す。
意図する使用	単純なソフトウェアの意図する使用の定義は、少ないセンテンス又はパラグラフで構成される。一方、大型で複雑なソフトウェアの意図する使用には、数ページにわたる広範な文書、及びソフトウェア要件の詳細が含まれる。リスクは、意図する使用の定義の深さを決定する上で重要な要素でもある。 意図する使用の要素は以下の通り。	意図する使用の定義は、ソフトウェアの適用範囲及び要件を理解し、そのソフトウェアの使用におけるリスクのレベルを評価するための基礎である。 意図する使用を定義することは、規制対象の活動を実行するソフトウェアを製造業者がどの程度信頼しているかを理解するための実践的な方法である。 ソフトウェアの目的及び意図は、システムの使用及びコンテクストに関する高度な視野と関連リスクに関する見識をもたらしてくれる。これにより、検証

1 開発段階 ― 定義		
活動	定義	価値
意図する使用（続き）	• ソフトウェアの目的及び意図 ○ この要素は以下のものによって定義されるソフトウェアの使用を説明する。 • ソフトウェアの使用：ソフトウェアに関する主な質問への回答を含む（セクション 4.3.1.4 参照） • 規則に準じた使用：ソフトウェアの使用によってコンプライアンスがサポートされる場合に規則への具体的なリファレンスにより、品質システムとの関連で、ソフトウェアがどのように使われるかの決定を含む（セクション 4.3.1.4 参照） • 品質システムプロセス内におけるソフトウェアと他のソフトウェア又はユーザーとの境界 ○ 既製ソフトウェアの場合、その目的及び意図する使用は、既製ソフトウェア開発業者の観点からみた一般的な使用ではなく、機器製造業者の観点からみた既製コンポーネントの具体的な使用を説明するものである。 ○ ソフトウェアの意図及び目的は、プロセスに対する高度な視点を必要とし、ソフトウェアを品質システムプロセスの要素又はコンポーネントの一つとみなす。 ○ ソフトウェアの意図及び目的は、ソフトウェアが何をするのかではなく、ソフトウェアのコンテクストを説明する。 • ソフトウェアの使用に関する要件 ○ この要素は、ユースケース、ユーザー要件、又はその他のシステム要件の形でソフトウェアの使用を説明し、ソフトウェアの目的及び意図を満たすためにユーザーがソフトウェアに求める機能を定義する。	アプローチの審査担当者のための基本的な情報がもたらされる。 ソフトウェアの使用に関する要件は、ユーザーがそのソフトウェアを使って何をするのかを読む人に正確に理解してもらうために必要なユーザーベースの視野をもたらしてくれる。 ソフトウェアに関する要件は、受入試験のための基本的な枠組みをもたらしてくれる。また、ソフトウェアに期待される機能を説明するコミュニケーションツールとしても有用である。 ソフトウェアに関する要件は、ソフトウェアの開発者が何を開発すべきかを正しく理解し、ソフトウェアの試験担当者が何をテストすべきかを理解するための詳細な透視図をもたらしてくれる。また、ソフトウェアアーキテクチャ、ソフトウェア設計、ソフトウェアコード、及びソフトウェア試験を開発するための確実な基礎をもたらしてくれる。

1 開発段階 ― 定義		
活動	定義	価値
意図する使用（続き）	○ 使用に関する要件は、詳細なソフトウェア要件の視点ではなく、プロセス及びユーザーワークフローの視点をもたらしてくれる。 ○ ソフトウェア使用に関する要件は、詳細な文書に記録された追跡可能なものである。 • ソフトウェアに関する要件 ○ この要素は、目的/問題が主にソフトウェアの意図及び目的並びに使用によって定義される場合、問題を解決したり、目標を達成するために、ソフトウェアが満たさなければならない条件又は性能を説明する。 要件要件は、正確、完全、明白、及び測定可能又は客観的に検証可能なものでなければならない。 ○ この活動のアウトプットは、特注のソフトウェア開発をソフトウェアの設計に利用可能な場合、及び既製のソフトウェアを調達の基準として利用可能な場合、ソフトウェア要件仕様書（IEEE）であると考えられる。	

1 開発段階 － 定義		
活動	定義	価値
検証プランニング	検証プランニングの定義は以下の通り。 • （GCSSDT）"プロジェクトのために採用されたアプローチを説明する管理文書。この計画では、通常、行うべき仕事、必要な資源、採用すべき方法、遵守すべき構成管理及び品質保証の手順、プログラムの組織などについて説明する。プロジェクトの中には、統合計画、セキュリティ計画、試験計画、品質保証計画などが必要なものもある。" • （NIST）"開発段階の後で実施される検証活動に関するプランニング。リスク及び/又はハザード分析活動の後に行われる。ソフトウェアのリスクが低い場合、'最低限の計画'で必要な活動が指示される場合もある。" 検証プランニングは2段階で実施される。 • 最初に、開発段階から定義段階にかけて、検証の文書化で期待される詳細及び努力のレベルを定義し、精査（経営陣の関心、部門横断的な参加、及び独立審査）のレベルを定義し、定義段階に含める活動を選択する。 • インプリメンテーション段階では、定義段階及び関連するリスク分析活動での決定に基づき、適切な検証活動を選択する。 検証プランニングからのアウトプットは、ソフトウェアが一貫して意図する使用の要件を満たしているという信頼を確立するために行われる活動を説明する計画である。	検証プランニングでは、各種の提出物に求められる厳密さ及び精査のレベルに関する決定事項、それらの提出物に求められる内容の範囲、及び使用するツール及び方法の選択を文書化する。これにより、検証プロセス全般で応用される批判的思考の証拠がもたらされる。

1 開発段階 － 定義		
活動	定義	価値
ソフトウェア要件の公式審査	（IEEE）"ソフトウェア又はハードウェア品目に関する要件がプロジェクト担当者、管理職、ユーザー、顧客、又はその他の関係者に提示され、コメント又は承認が得られるまでの間のプロセス又はミーティング。例として、ソフトウェア要件の審査が挙げられる。 要件審査は、意図する使用の定義のいずれか又はすべてのレベル（ソフトウェアの目的及び意図する使用、ソフトウェアの使用に関する要件、又はソフトウェアに関する要件）で個別又は複数同時に行われる。	ソフトウェア要件仕様書は開発及びインプリメンテーションプロセスの主要なインプットであることから、ソフトウェア要件仕様書が正確、完全、明白、及び測定可能又は客観的に検証可能なものであることを保証するために公式審査が行われることが多い。
ソフトウェア開発ライフサイクルモデルの選択	ソフトウェアライフサイクル全体の開発に関する部分で使用するライフサイクルの方法及びコントロールを定義する。 IEC 62304は一部のソフトウェアのプロセス標準として最適なものと考えられる。	これまでの結果、適切なライフサイクル開発モデルを選択することで、ソフトウェアのクオリティ及び信頼性が向上している。

1 開発段階 ー 定義		
活動	定義	価値
リスクマネジメントプランニング	リスクマネジメントのプランニングには、以下のサブ活動のいずれか又はすべての準備が含まれる。 • 高リスクを伴う機能の特定 • 規制又はリスクの影響から見たソフトウェア機能の特定 • 機能分割及びリスクのトレーサビリティ • ソフトウェアが危害につながる確率を確定する目的で"分割度"（直接または何らかのプロセス方法による）を利用した潜在的な危害に関するソフトウェアの機会/近似性の特定 • リスク評価、リスク審査、リスク分類の活動 • ライフサイクルプロセスを通じたリスク分析の反復 リスクマネジメントプランニングのアウトプットとして、リスクに関係するこのソフトウェアのさまざまな問題エリアを分析するためのアプローチ、及び故障モード影響解析（FMEA）や故障樹解析（FTA）などのリスク分析法の選択が挙げられる。	リスクマネジメントのプランニングは、ソフトウェアの潜在的な危害に関する見識をもたらし、検証努力におけるツール及び努力レベルの選択を促してくれる。
製造/業務プロセス内におけるリスク予防策の特定	リスク/ハザードの予防策（手順管理など）を特定するメカニズム。予防策が用意され、適切に機能していることを確認するための継続的なモニタリングを含む。	リスク要因がプロセスに入り込むのを予防する。

2 開発段階 ー インプリメンテーション		
活動	定義	価値
ソフトウェア障害の分析（リスク分析）	自動化するプロセス及びプロセス障害の分析で特定された問題エリアに関係するソフトウェア障害の影響を判定する。安全なソフトウェア慣行を確立するにはAAMI TIR32の採用が最適と考えられる。リスク評価及びリスク評価法の詳細については付録Bを併せて参照すること。 詳細なソフトウェア障害分析の具体的なテクニック： • ソフトウェア故障樹解析：ソフトウェアが原因でそれ自体が危険な状態に陥る可能性がないことを証明したり、ソフトウェアが危険な状態に陥る可能性のある環境条件を明らかにするために使用する。 • 故障モード影響解析（FMEA）：特定の故障モードがソフトウェアの挙動に及ぼす影響を判定するために使用する。	この活動により、ソフトウェア内にある高リスクエリアの試験に重点を置き、設計に採用すべきリスク予防策の特定を推進することができる。ソフトウェアの要件の脆弱な部分又は欠落した部分を明らかにし、ハザード又はリスクの潜在的な可能性が高いエリアでのロバスト性を向上させることができる。
ソフトウェアアーキテクチャの文書化及び審査	ソフトウェアアーキテクチャは、ソフトウェア要因の上部構造及びそれらの要因の関係定義する。 この活動では、アーキテクチャを文書化し、正確性、完全性及びソフトウェアの性能について審査（検証）を行う。	コミュニケーションを改善し、要件をサポートする上でのアーキテクチャの妥当性及び適正を検証する。この段階でソフトウェアのリスクを正しく評価することで、予防策を単純化することができる。
設計仕様書	ソフトウェアの要件がどのようにして履行されるかを正確に記載した文書。通常含まれるものとして、ソフトウェア又はコンポーネントの構成、アルゴリズム、制御論理、データ構造、データセットの使用に関する情報、インプット/アウトプットのフォーマット、インターフェースの説明などが挙げられる。 ソフトウェアの設計は、サイジングなど、さまざまな形式の分析によって推進される。初期段階で必要な機能の特定を容易にするために、プロトタイプを設計してもよい。	ソフトウェアがどのように機能するかを理解するために、アーキテクチャとコードの間にある抽象的な層を明らかにしてくれる。具体的なインプリメンテーションの要件を遂行する上で必要な構造を明らかにしてくれる。

2 開発段階 ー インプリメンテーション		
活動	定義	価値
開発/設計審査	審査を実施し、一つまたは複数の構成アイテム用に選択された設計アプローチの進行状況、技術的な妥当性、及びリスク解決策を評価する。この審査には以下の目的がある。 • 構成アイテムに関する要件の適合性を設計ごとに判定する。 • 定義のレベルを評価し、選択されたインプリメンテーションの方法及びプロセスに関連する技術的なリスクを測定する。 • 構成アイテムと機器を構成する他のアイテム、施設、ソフトウェアと人の間にある物理的及び機能的なインターフェースの存在及び互換性を確定する。さらに、場合に応じて、 • 予備的な業務文書及び補足文書を評価する。	設計をコードに変換できるように、ソフトウェアの要件及び設計の仕様書が正しく作成されていることを確定する。 ソフトウェアが複雑で高リスクな場合、又は重要な環境で動作する場合は特に有用である。
ソフトウェア設計におけるリスク予防策の特定	リスク評価で特定されたリスク/ハザードの予防策を特定する。継続的なモニタリングを実施して予防策が用意されて正しく機能していることを確認できる反復的なプロセスでなければならない（手順管理、ハードウェア冗長性など）。	ソフトウェア使用の残留リスクを制限し、それを意図する使用に準じた許容可能なものにする。
コードレビュー/コード検証	欠陥の発見・除去及びコード全体のクオリティ向上を目的としたソフトウェアソースコードのピアレビュー。コードレビューには以下の種類がある。 • 単一ピアによる非公式なレビュー • 非公式なグループレビュー • 公式ミーティングでのウォークスルー • 役割及び責任が割り当てられた公式な検査 コードレビュー及びコード全体のクオリティは、共通コード化基準を確定し、それを遵守することで向上する。	ソフトウェア設計のコードへのインプリメンテーションが適切なことを確認する。プロセスの初期にエラーを発見・除去する最後の機会。

2 開発段階 ー インプリメンテーション		
活動	定義	価値
トレーサビリティ分析	設計、コード、試験、リスク/ハザード分析及びリスク予防策に関する要件のトレーサビリティ。プロセス要件へのトレーサビリティを含めることもできる。	トレーサビリティ分析は、未定義の要件を特定し、すべての要件が検証済みであることを確認するためのカバレージ解析を可能にする。 トレーサビリティは、回帰試験及び保守活動でも有用である。設計又はコードに変更が生じた場合、それを要件に準じてトレースバックすることができる。

2 開発段階 ー インプリメンテーション		
活動	定義	価値
販売業者の監査	ソフトウェア販売業者のシステムについて、その業者が安全かつ有用なソフトウェアを供給する上で十分な能力を備えていることを購買者に保証するために必要なレベルに達しているかどうかを評価する。販売業者の監査には、以下をはじめとするさまざまな方法を利用できる。 • 開発及びサポートの慣行に関する質問を記載した簡単なアンケートを業者に送付する。 • 業者が供給するソフトウェアに対するユーザーの要求を調査する。 • 業者を選定し、試験又は欠陥追跡などの問題エリアについて現場監査を行う。 • ソフトウェアの仕様、開発、試験、導入及びサポートに関連する全業者のシステムについて、品質システムの徹底的な現場監査を行う。販売業者監査を実施することで、同等又はより重要な意図する使用について業者システムの使用状況を確認できる。 チェックすべき具体的な要素として、以下の例が挙げられる。 • 業者が作成した"既知の問題リスト"のチェック • 業者が作成した基本システム検証資料のチェック • "Out-of-the-box"ソフトウェア作業フロープロセス図のチェック • "Out-of-the-box"標準レポートライブラリーのチェック • 標準作業フロー及び業務規則に加えられた構成変更のギャップ分析 • 業者から供給された導入検査及び認定用自動化試験ツールの説明及び転嫁	販売業者のプロセスを知れば、ソフトウェア製品に対する自信を強めたり、弱いエリアを認識できるほか、ユーザーが必要に応じて受入試験をカスタマイズすることができる。問題の報告手順及び問題解決プロセスをきちんと定義していれば、問題が発生した場合にそれを解決する上で役に立つ。

3 開発段階 ー 試験		
活動	定義	価値
試験プランニング	試験プランニングでは、ソフトウェアがその意図する使用に準じていることを裏付ける信頼醸成をサポートする試験活動の全体的なアプローチを定義しなければならない。しかし、ソフトウェア試験だけでは、ソフトウェアがその意図する使用に準じていることを裏付ける信頼醸成には不十分である。試験と他の検証テクニックを併用して、包括的な検証アプローチを実現する必要がある。 リスク推進要因などの要素を踏まえて試験のレベルを決定し、開発者試験やユニット試験、統合試験、ユーザー試験、負荷試験、操作試験などの適切な試験法に基づき、ソフトウェアが要件及び設計の仕様に準じていることを裏付ける上で適切なレベルの信頼を獲得しなければならない。 試験で確認された不具合を記録し、それに対処するためのプロセスを定義する。 （IEEE）意図する試験活動の適用範囲、アプローチ、リソース、及びスケジュールを明記した文書。それにより、試験項目、試験すべき機能、試験作業、責任、必要なリソース、及び不測事態対応計画を特定する。	価値の低い活動に費やされる時間を短縮して検証努力を節約する。 試験活動の役割及び責任を明らかにする。 これにより、試験に採用するアプローチ及び方法の根拠がもたらされる。監査に利用可能な以下の証拠を得ることができる。 • 全体的な検証努力に必要な適切なボリュームの試験が実施されている。 • すべての不具合への対処が適切に行われている。

3 開発段階 ― 試験		
活動	定義	価値
ユニット試験	(1) (NIST) 誤字、統語誤用、及び論理エラー用モジュール、ユニット設計の正しいインプリメンテーションのためのモジュール、及びユニット要件適合のためのモジュールの試験。 (2) (IEEE) 単一のソフトウェア要素（ユニット又はモジュール）又はソフトウェア要素の集合のための設計のインプリメンテーションを確認する目的で実施する試験。	オーナーの管理下でコーディングが行われる場合にのみ該当（改良、インターフェースなど）。 ユニットごとに検証のための試験を行う代わりに、開発業者のユニット試験を監査/モニタリングすることができる。 システムのレベルからは通常到達できないレベルでのソフトウェア試験に価値をもたせる。その好例として、管理された試験環境で"絶対に起きてはならない状況"及びなかなか起こり得ない状況について、エラー状態及びエラー回復の試験を行う場合が挙げられる。
データ検証	データの正確性を確認するために遂行される活動。データの移送、変換、又は試験の一環として、若しくは独立して行われ、状況に応じて統計学的サンプリングを含めることもできる。	データの正確性における信頼を醸成する。
統合試験	(IEEE) ソフトウェア要素、ハードウェア要素、又はその両方を組み合わせて試験を行い、ソフトウェア全体が統合されるまで、それらの連携を評価する通常の試験進行。	この活動の価値は、データ構造を経由して通信しなければならない主要なソフトウェアコンポーネントの試験、又は複数のハードウェア要素をカバーする機能を備えたソフトウェア、及び通信手段が電子的で通信プロトコルによって制御されるソフトウェアの試験を行うところにある。ユニット試験と同様、このタイプの試験は、タイミングやエラー回復、デッドロックなど、システム又はユニットのレベルでシミュレーションが難しい連携を調査する上で必要となる。

3 開発段階 ― 試験		
活動	定義	価値
ユースケース試験	ユースケース試験は、システム又はコンポーネントの内部機構又は構造を無視した機能試験の一形態であり、選択されたインプット及び実施条件に反応して生じたアウトプットに重点を置く。ユースケースごとに関連のあるインプットパラメータを用意し、実際の使用条件をシミュレーションすることが特定された数値のセットをそれぞれのパラメータに割り当てることができる。目標を達成の手順を示す所定のフローを利用して、一連のユースケースを関連づけることができる。	システム全体が正しく機能し、そのシステムが現実的な使用シナリオ及び作業量を処理し、実際の生産に近い環境においてユーザーに受入可能な方法でデータを効率的に処理できることを保証する。
インターフェース試験	アウトプットからインプットに至るまでの全体的なデータ移送経路を考慮しながら、ソフトウェアアプリケーション間のインターフェースを確認する。これは直接的なインターフェース試験又は100%データ検証により遂行可能である。試験活動には、インターフェースが正常と異常の両ケースで要件に沿って仕様限界又は境界条件で機能することを確認するための戦略が含まれなければならない。	データフロー検証は、高リスクアプリケーションの既知のリスクを低減するための試験法の一つである。 この活動の価値は、異種のソフトウェアコンポーネント又はアプリケーションをカバーする機能を備えたソフトウェアの試験を行うところにある。経験上、コード又はデータをわずかに変更しただけでも、境界条件で適切な試験を実施しなければ、ソフトウェアインターフェースは故障点になりやすいことが判明している。
回帰試験	(NIST) ソフトウェアの開発及び保守段階で行われた変更又は修正に起因するエラーを発見するために、それ以前にプログラムが正しく機能していたときの試験例を再実施すること。 ソフトウェアがその意図する使用を満たすために実行しなければならない基本試験例を定義・記録する必要がある。回帰試験活動を正しく実行するために、保存可能な構成パラメータセット又はデータセットの定義又はベースライン化が必要となる場合もある。	修正後も引き続きソフトウェアが当初の要件仕様に適合していることを確認する。 他のモジュール、アプリケーション、オペレーションシステム、データベース構造に小規模又は大規模な変更が加えられた後も、ソフトウェア又はソフトウェアモジュールがその意図する使用に適合していることを証明する際に使える非常に有用なツールである。

3 開発段階 ― 試験		
活動	定義	価値
業者供給の試験スイート	これらのスイートは、ソフトウェアソリューションの最大能力をテストし、エンドユース環境でのソフトウェアの性能に対する大きな信頼を獲得することができる。しかし、講じられているリスク予防策の試験を含め、定義された意図する使用及び試験の完全性に対する妥当性についても評価が行われなければならない。ソフトウェアの使用期間中、販売業者に試験スイートの保守を要請する契約が必要になる場合もある。	"通常使用"又は基本構成におけるソフトウェアの精度に対する信頼を確立する上で有用なツール。ビジネスでの"ユースケース"を補強し、エンドユース環境での正しい機能を保証しなければならない。
ソフトウェアシステム試験	(IEEE) 統合されたハードウェア及びソフトウェアの試験を実施し、ソフトウェアが所定の要件に適合していることを確認するプロセス。このような試験は、開発環境とターゲット環境の両方で実施される。この試験には、サブセットとして、ソフトウェアのインストール及び構成検証活動が含まれなければならない。 ソフトウェア検証は、ソフトウェアがその意図する環境で意図するユーザーに使われる場合の適合性を確認するものであり、ソフトウェアシステムの試験とは異なる。ソフトウェアシステムの試験では、ソフトウェアに関する要件が適切に履行されていることだけを確認する。 自動化製造システムの場合、プロセス検証試験で、これらの試験の一部又は全部をカバーすることができる。品質システムアプリケーションの場合、ソフトウェアの作業指示書に規定された手順をすべて実施すれば、ソフトウェア試験の要件をカバーすることができる。	この試験は、ユーザー試験に最も類似している。このような試験の中には、手順が細かく定められていて再現性が高く、専門知識を有する担当者によって実施されるものもあれば、意図する使用環境で意図するユーザーによって行われるものもある。使用エラーで発見不能なソフトウェア設計及びインプリメンテーションのエラーを発見できるところに価値がある。 この試験では、適切なバージョンのソフトウェア及び関連ハードウェアとソフトウェアの構成が、ターゲット環境で仕様書又は設計に準じて正しく設置されていることを確認する。
ユースケース試験	ユースケースに基づいて実施される試験。それらのユースケースで定義されている代替フロー及びエラー状態を含む。	ユースケース試験は、ソフトウェアと対話するユーザーの観点から行われる。ソフトウェアとの通常及び異常な対話の際にユーザーが遭遇するであろう欠陥を発見できる。

3 開発段階 ― 試験		
活動	定義	価値
ノーマルケース試験	(GPSV) 通常のインプットによる試験。 (IEEE) (1) 結果を観察又は記録し、ソフトウェア又はコンポーネントの何らかの側面を評価しながら、所定の条件下でソフトウェアまたはコンポーネントを操作するプロセス。(2) 既存の状態と要求された状態の差 (バグなど) を発見し、ソフトウェアの機能を評価する目的でソフトウェアを分析するプロセス。	要件の妥当性を検証する。 備考：期待される有効なインプットだけでソフトウェア製品の試験を実施しても、そのソフトウェア製品を徹底的にテストしたことにはならない。従って、ノーマルケース試験は、それ自体ではソフトウェア製品の信頼性に対する十分な信用を獲得することはできない。
ロバスト性試験 (ストレス試験)	(GPSV) 予想外の無効なインプットが与えられた場合、ソフトウェア製品が適切に動作することを実証するソフトウェア試験。 そのようなテストケースを特定する方法として、同値類分割、境界値分析、及び特殊例特定 (エラー推測) などが挙げられる。 (IEEE) 所定要件の限界又はそれを超えた状態でシステム又はコンポーネントを評価するための試験。 このタイプの試験では、ソフトウェアが障害を起こす負荷及びその障害発生の仕組みを判定できる。性能試験と同じプロセスを採用しながら、非常に高レベルの負荷をシミュレーションした状態でストレス試験が実施される場合が多い。	ソフトウェア内の弱点を特定し、ソフトウェアが通常の生産作業負荷を受けた状態で機能することを証明できる。 リソースの問題を特定し、システムアーキテクチャの弱点を判定するのに役立つ。 負荷を受けた状態のグレースフルデグラデーションが結果として破局的な障害に至らないことが望ましい。
アウトプット強制試験	(GPSV) 所定 (又はすべて) のアウトプットがシステムによって適切に生成されていることを確認するためのテストインプットを選択すること。 アウトプット強制では、システムから特定のアウトプットを生成する目的でいくつかのテストケースを設計する。ここでは、システムの反応を誘発するインプットではなく、希望するアウトプットを生成することに重点が置かれる。	要件及びシステムアウトプットの妥当性を確認する。

3 開発段階 － 試験		
活動	定義	価値
インプットの組合せ試験	ソフトウェアユニット又はシステムが操作の実行中に遭遇する可能性があるインプットの組合せを利用した試験法。 エラー推測では、システムの特定エリアで発生が予想されるエラーの項目別リストを作成し、それらのエラーをチェックするためのテストケースを設計する。 因果関係のグラフ化は、テストケースに採用するソフトウェア製品へのインプットの組合せを体系的に特定するための機能的ソフトウェア試験法の一つである。	ここで特定される機能的試験法は、個別又は単一の試験インプットに重点を置く。ソフトウェア製品の大半は、それぞれの使用条件下で複数のインプットを使って操作が行われる。 エラー推測を拡充してインプットの組合せを特定できるが、これは特殊なテクニックである。 エラー推測は、科学的な試験法というより技能的な試験法といえるが、システムの履歴に精通した試験官が実施すれば非常に効果的である。
ベータ試験	少数のクライアントを対象に販売業者がライブ環境で実施する試験。 （プレスマン）一ヵ所又はそれ以上のエンドユーザー施設で、開発者による監督が行われない環境において、顧客がソフトウェアのライブアプリケーションで実施する受入試験。	ベータ試験は通常、所定の手順や監督のない状態で行われる。意図する使用の分析や試験に最適なシミュレーションである。一般的にラボで実施される試験ほど厳密ではないが、予測不能な使用状態をテストするのに適した方法といえる。 しかし、ベータ試験はライブ環境で行われるため、検証が完了するまで製品を保留の状態にしたり、独立した代替的な対象を設けて、試験中のソフトウェアに依存することなく、プロセスのアウトプットのクオリティを検証しなければならない。
性能試験	（NIST 500-234）反応時間、CPU使用率、及び操作で数量化された他の機能の要件に準じて、ソフトウェアシステムがその程度作業を実行するかを測定すること。 性能試験では通常、自動化された試験スイートが使われる。それにより、通常、最大、及び例外のさまざまな負荷条件を簡単にシミュレーションすることができる。	システムが生産現場の環境で要求された反応を達成できるかどうかを確認する。 この種の試験は、使用頻度の高いアプリケーションにおける性能上の弱点を特定する上で特に有用である。 性能試験は、購入検討中のサーバやミドルウェアなど、サードパーティ製品の環境でのベンチマーキングにも応用できる。

4 開発段階 － 導入		
活動	定義	価値
ユーザー手順のレビュー（ソフトウェアの使用に関する手順を含む。ソフトウェアの使用が適切に定義されていることを保証する）	ソフトウェアの使用に関連するユーザー手順/指示のレビュー。指示は完全、正確、かつ明確か。ソフトウェアの使用に関連するユーザー文書の正確性を審査/検証する。	ソフトウェアが"正確"でもユーザー指示が不正確だと、ソフトウェアは正常に機能しない。不正確な場合、ソフトウェアの誤用につながる可能性がある。
アプリケーションの社内トレーニング	文書に規定され、ソフトウェアに特化したトレーニング活動。	ソフトウェアの使用又は習得が"困難"な場合に重要性を増す。ソフトウェアのリスクプロフィールにあまり依存しないソフトウェア特性の一つである。
据付時適格性確認	ソフトウェアが、インストール指示書の規定に準じてインストールされ、機能していることを裏付ける信頼を確立する。	ハードウェア及びソフトウェアのバージョン/構成、すべてのコンポーネントの連携機能が正常であることを確認する。
運転時及び性能適格性確認（プロセス検証が行われる場合）	運転時適格性確認 － 製造プロセス及び関連システムが所定の限度及び許容範囲内で正常に動作可能なことを裏付ける信頼を確立する。 性能適格性確認 － プロセスの有効性及び再現性を確立する。	これらの活動は、プロセス検証に必要なものであり、エンドユース環境でソフトウェア開発試験を実施する機会をもたらしてくれる。ソフトウェアが正常に機能することを裏付ける信頼も確立する。
最終受入試験	最終的な導入の直前に行うシステムの試験。通称"稼働試験"と呼ばれる。	ソフトウェアがその意図する使用に適合し、正常に機能していることを裏付ける信頼を確立する。
オペレータ検定試験	トレーニング受講者がトレーニングで示す適格性を確認すること。	製造アセンブリオペレーションで要求されることが多い。オペレータエラーを減らす手段。

5 保守段階		
活動	定義	価値
保守プランニング	保守プランニングに関連する方法として、以下の例が挙げられる。 • フォワードプランニング – この方法は、フォワードプランニング及びソフトウェアへの変更予測をカバーする。定義されたプロセスとソフトウェアインプリメンテーションに特化した活動又はツールの標準セットを利用し、ソフトウェア変更を分類・評価するための全体的な枠組みを提供する。セキュリティのアップグレード又はパッチに関するプランニング及び戦略を含む場合もある。この方法は、保守段階に入る前、最初のソフトウェアインプリメンテーションで利用できるほか、保守段階では随時利用可能である。 • 審議中の変更プランニング – この方法は、ソフトウェアの変更が審議中に行われるプランニングをカバーする。このプランニングは通常、審議中の変更に特化した活動に重点を置く。このプランニングは、ソフトウェアの*保守段階*に行われる。	このツールは、ソフトウェアの検証された状態を維持する活動に構造をもたらしてくれるほか、初回導入時から保守活動時にかけて重要なソフトウェアに関する重大な情報を収集すると考えられる特定ソフトウェアの変更評価及びインプリメンテーションの統一アプローチを提供する。例えば、ソフトウェアに特注のコンポーネントがある場合、このコンポーネントは、既製の基本ソフトをアップグレードする際、取り扱いに関する特別なニーズを有する。特注コンポーネントは、アップグレードの際に、再構成又はメインアプリケーションからの隔離が必要となる場合もある。
既知の問題分析	購入したソフトを使って規制プロセスを自動化する場合、使用中の検証済みソフトウェアに適用される定期的なアップグレード及び/又はパッチを受け取ることがある。既知の問題分析は、インストールされたソフトウェアの使用及び/又は検証された状態への影響について、販売業者が認知するソフトウェアのあらゆる問題を評価するプロセスである。	この分析を行えば、悪影響の原因となり得る変更が加えられたソフトウェアのインストールを回避することができる。
互換性試験	（GCSSDT）複数のソフトウェアシステムが情報を交換する能力を判定するプロセス。	どの外部ソフトウェアがバージョンの不一致により現行ソフトウェアに影響を及ぼすかを特定する。 アウトプットは、ソフトウェアがリソースをインターフェースで連結したり、共有する他のソフトウェアに有用な場合もある。
インフラ互換性分析	ソフトウェアインフラへの変更がインストール済みソフトウェアにどのような影響を及ぼすかを判定するプロセス。これには、ハードウェア又はシステムのロケーションへの変更が含まれる。	インフラの変更がソフトウェアに関する予測不能な機能の問題の原因になるのを回避する。

5 保守段階		
活動	定義	価値
システムモニタリング	以下の方法を利用して、次に紹介するさまざまな観点からシステム全体の"健康"を判定できる。 • 意図する使用が変更されているかどうかの定期的なチェック – ソフトウェアが現行の意図する使用に引き続き対応しているかどうか、意図する使用又はソフトウェア、若しくは両方に変更が求められているかどうかを確定する。 • エンドユーザーによる実際の使用 – ユーザーによるシステムの使用が保守の際にモニタリング可能な情報を生み出す。エラーログ、ヘルプデスクコール、障害レポートを使用できる。 • トレーニング効果の評価 – エラーログ及びヘルプデスクコールは、トレーニングプラン又はプログラムの効果判定のソースになり得る。 • 欠陥分析 – この方法を利用してシステム内のバグの状態をチェックする。バグは社内で報告される場合と販売業者から報告される場合がある。この方法は、社内で開発されたものと既製品、両方システムに応用できる。 • データ監査 – この方法を利用して、データの保守がシステム内で正しく行われていることを確認する。	ソフトウェア又はシステムの継続的な適合性及び応用性に関する情報を提供する。 保守活動（改良、適応又は完全化）のきっかけとなる傾向を特定する。 トレーニングの評価に適したツール。 ソフトウェアが所定の要件を満たしていることを裏付ける信頼をさらに醸成する。 ソフトウェアの意図する使用又はリスク予防策に影響を及ぼす可能性がある既知のバグに対して、問題回避又はシステムの追加変更を行うことができる。 これは、検証プロセスの有効性を評価するツールのほか、新規又は修正要件のソースになり得る。
バックアップ及びリカバリープロセス	バックアップ及びリカバリープロセスには、システムバックアップ、バックアップメディアの保存及び保持、及びバックアップメディアからデータを復元するためのリカバリー手順などが含まれる。	このガイダンスを利用して、データ消失を回復可能なものにする、データ保持の要件を満たすメカニズムを提供できる。

5 保守段階		
活動	定義	価値
運用管理	バックアップとリカバリーのプロセス、モニタリング及びレポート作成のほかにも、ソフトウェアが意図した通りに動作することを保証するための運用管理がある。一般的な方法として、以下の例が挙げられる。 • セキュリティ － この方法は、データの消失、汚染、悪用、及びシステムの意図しない使用を予防するための管理策を利用する。 • アクセス権管理 － この方法は、システムへのアクセスを許可したり、システム内でアクセスを管理することで、ユーザーによるシステムの使用を管理する。 • データベース管理 － データベースを管理し、所定のプロセスを利用して効率的な運用状態でデータベースの保守を行う。 • アーカイブ化 － アーカイブ化の方法を利用して、システムの日常業務に必要とされないデータを管理する。 • 不測事態対応計画 － コンピュータ化されたシステムの障害発生時も運転を継続することが目的である。	意図する環境でシステムを適切に使用することが、検証された状態を維持する上で重要なポイントである。
回帰分析	回帰分析には、トレーサビリティ分析又はインパクト分析など、システムの検証された状態を維持するための活動を決定する作業が含まれる。	この分析を適切に実施すれば、変更案の影響を大きく受けるエリアに検証努力を集中させ、ソフトウェアの変更されない部分が変更の影響を受けないことを確認するための試験など、適切な試験を決定することができる。

付録 B
リスクマネジメント

ISO 14971 の概念を生産及び品質システムソフトウェアに応用する。

ISO 14971 の“Medical devices － Application of risk management to medical devices（医療機器 － リスクマネジメントの医療機器への応用）” は、生産又は品質システムではなく、医療機器のために作成された規格である。しかし、この規格の最高レベルに位置づけられたリスクマネジメントの概念及びプロセスは、生産・品質システムにも応用できる。

このレポートの主題は生産・品質システムソフトウェアでの検証及びリスクマネジメントだが、ソフトウェアが全体的なプロセスやシステムを構成するコンポーネントの一つであり、故障する可能性があるということを考慮せずに、ソフトウェアのリスクを徹底的に分析することはできない。医療機器規格の ISO 14971 は、患者又はその他の医療機器ユーザーへの危害のリスク低減に関するものである。生産又は品質システムのコンテクストにおける危害とは大きく異なる。この場合の危害とは、生産される機器への影響（通常、患者への直接的な危害ではない）、全体的な生産プロセスへの危害、品質記録の完全性への危害、規則遵守への危害、又は事業への危害を意味する。

ここで紹介するリスクマネジメントのプロセスは、ソフトウェアで駆動するシステムに固有なタイプの危害を個々に検討するものであり、それがこのレポートのテーマとなっている。そこで、14971 のリスクマネジメントプロセスを、これらの危害発生源のリスクの管理及び予防に応用する。このコンテクストにおける 14971 の参照は、この規格が生産・品質システムソフトウェアに直接適用するという意味ではない。しかし、リスク分析、リスク評価、及びリスクマネジメント（“危害”の語法に多少修正を加えている）の一般化されたプロセスは、生産・品質システムソフトウェアに適用できるほど一般的なものといえる。新しいプロセスを考案したり、同じプロセスを言い表す言語を変えたりするのではなく、我々は目的のために 14971 からリスクマネジメントプロセスを“借りる”ことにした。

リスクマネジメントプロセスのフローチャート

下記のソフトウェアリスクモデルのフローチャートは、以下の目的を適切に遂行するために必要なプロセス及びプロセス要素を示している。

- 規制された生産環境でのソフトウェアの使用がもたらす潜在的な影響を特定又は発見する。
- ソフトウェアの故障又は誤用がもたらす潜在的なリスクを評価する。

- ソフトウェアの故障に起因する危害の本質的な重大性を減らすための適切な予防策を通じてリスク低減を適用する。
- 適切なレベルの検証が適用されるようにする。

ソフトウェアの使用がもたらす潜在的な影響を特定又は発見するには、まず最初にソフトウェアの意図する使用を定義し、最初のリスク分析を行う必要がある。この分析の一環として、“もしシステムが故障したら、何が機器の安全性や有効性に影響を及ぼし、生産担当者への潜在的な危害や環境へのダメージを生み出し、生産プロセスや品質システム、又はその両方に悪影響を及ぼすのか”という疑問について考えなければならない。単純な低リスクのソフトウェアの場合（このタイプのソフトウェアを含む例については付録 C を参照）、リスクマネジメント活動を文書化するには、この評価だけでほぼ十分といえるが、高リスクを伴うシステムの場合、リストに記載したすべてのエリアの影響を及ぼす可能性があるため、分野横断的なチームによる非常に綿密な評価が必要となる。

リスク予防策のアプリケーションには、ソフトウェアの設計変更、手順変更、ハードウェア冗長性、セキュリティコントロール、コードレビューなどの静的試験、負荷試験やパス試験などの動的試験、バックアップシステム、モニタリングシステム、アウトプットのマニュアル検査、又は販売業者の検査などが含まれる。これらのリスク予防策は、すぐれたエンジニアリングの問題解決テクニックを応用したものである。予防策を評価・応用するこのプロセスは、システムの危害発生能力が容認可能なレベルにまで低減されたという確信が得られるまで継続される。この TIR では、リスクを低減するこれらの方法又は理念をリスク予防ツールとして考えることを推奨する。これらのツールを文書化することは、リスク分析及び上記のリスク予防ツールを利用したリスク低減計画の背後にある思考プロセスの共有を促す良い方法の一つといえる。

リスク予防策の適用により新たな意図しないリスクが全体的なシステムに入り込む可能性を評価するためには、上記の質問プロセスを繰り返し実施しなければならない。リスク評価に基づき、検証努力に適切なレベルの厳密性を割り当てる。

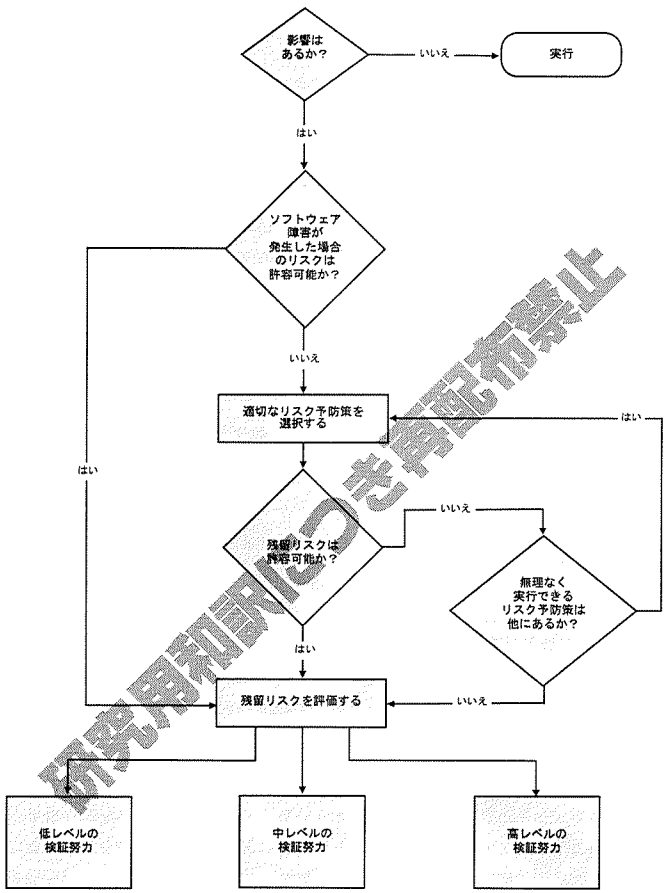

図 B1 － リスクマネジメントモデル（ISO 14971 に基づき翻案）

リスクマネジメント用語の生産・品質システムへの応用

ISO 14971 にあるようなリスクマネジメントの用語及び概念は、手を加えないまま、その大部分を生産・品質システムソフトウェアに応用できる。生産・品質システム（具体的にはそれらのシステム内のソフトウェア）に応用する場合、特殊な意味を持つ用語と概念もいくつかある。

リスク分析及びリスクマネジメントに関する 14971 などの文書によれば、リスクはハザード、原因及び間接要因の階層構造になっている。このレベルでリスクを取り扱うことは、検証をテーマとするこの TIR の適用範囲外であるが、ソフトウェア故障に伴うリスクのメカニズムとそれを予防する最善策を理解するためにシステムを解体する上で役に立つことは間違いない。

リスク（Risk）

リスクとは、14971 の定義によれば、特定された危険が発生する確率と結果的に生じた危害の重大性を組み合わせたものである。生産・品質システムソフトウェアの障害に起因する危害の重大性は、医療機器そのものに起因する危害の重大性とは微妙に異なる。

危害（具体的にはソフトウェアの障害に起因する危害）が発生する確率というのは難解な概念だが、それは単に、ソフトウェアの障害発生率が予測困難であるという理由による。

まず最初に、自動化プロセスソフトウェア、つまり、コンプライアンスをサポートするために使われるソフトウェア、又は機器の製造や検査プロセスの自動化に使われるソフトウェアを例に挙げ、ソフトウェアという用語を定義してみよう。

自動化プロセスソフトウェアの検証は、通常、患者又は医療機器ユーザーへの危害の直接的な原因にならないという点で、医療機器ソフトウェアと異なる。安全性に関わる（有害な）自動化プロセスソフトウェアの障害は、大抵、使用時に機器が故障した場合にのみ有害となる。自動化プロセスソフトウェアは、通常、患者又は機器ユーザーへ*間接的な*危害をもたらすだけである。

生産・品質システムの障害が間接的な危害を及ぼす例として、以下のものが挙げられる。

- 生産滅菌システムの障害が有害又は致命的な感染症を引き起こす。
- 最終試験システムの障害により潜在的な機器の欠陥を発見できない。
- 医療機器にトレーサビリティ機能をもたらす MRP システムの障害により、機器の潜在的なユーザーに安全性に関するリコールを通知できない。

生産・品質システムの障害が直接的な危害を及ぼす例として、以下のものが挙げられる。

- 生産安全性システムの障害がオペレータに危害をもたらす。
- 滅菌システムの障害が原因で環境に有害物質を放出する。

リスクを予防するためのオプションとして、障害の重大性を低減すること、傷害発生の確率を減らすこと、又はその両方が考えられる。

重大性（Severity）

生産・品質システムソフトウェアの障害に起因する危害の重大性が、患者若しくはソフトウェアで製造や品質を管理する医療機器のユーザーに直接的な危害を及ぼすことは滅多にない。この場合の危害は間接的なものである。最終的に患者又は機器ユーザーへの危害の原因となるのは、機器にもたらされる危害である。言うまでもなく、間接的な危害の方が重大性は低い。実際、生産・品質システム障害の方が重大性は高いと考えられる場合もあるが、その理由は単に、それらのシステムに生じた 1 回の障害が多数の機器の障害につながり、障害が発見される前に、結果として多くの患者が影響を受けるからである。一つの機器に生じたソフトウェアの欠陥が、一人の患者に 1 度だけ危害をもたらすこともある。

生産・品質システムの障害が、結果的に直接的な危害と間接的な危害の両方をもたらすこともあり得る。下記に掲載した危害は、相互排他的なものではなく、それぞれが患者及び/又は医療機器ユーザーに間接的な危害を及ぼす可能性を持っている。その例をいくつか紹介する。

- 医療機器への悪影響
  - マシンツールが臨界許容を生じない。
  - キャリブレーションシステムが薬剤送達装置を正確に較正できない。
  - 滅菌コントローラーが故障して無菌でないコンポーネントが発生する。
- 生産プロセスへの悪影響
  - 自動化プロセスの障害に伴ってマニュアルの問題回避策が講じられ、生産が減速する。
  - ソフトウェア制御のプロセス障害により規格外部品の割合が増加する。
- 規則遵守への悪影響
  - 苦情処理システムが間違った故障統計データを出力し、現場報告された欠陥が未確認のまま放置される。
  - 機器サービス/修理システムが故障し、それまで見つけられなかった欠陥を指摘できたかも知れない問題の動向を浮き彫りにすることができない。
  - インプラントに関するデータベースの完全性が損なわれる。
  - 製品の安全性チェックに関連する品質管理記録の消失。
  - コンプライアンスデータの消失。
  - 機器検証データの消失。

- 製造された機器のソフトウェア構成を管理・報告することができない。
- トレーサビリティ機能をもたらす MRP システムの障害により、機器の潜在的なユーザーに機器の安全性に関するリコールを通知できない。

- 生産担当者又は環境への危害。
  - オペレーターの傷害又は有害物質の漏出。

生産・品質システムを自動化するソフトウェアに関連するリスクを分析する際、あらゆるカテゴリーの危害を考慮しなければならない。それぞれの危害の重大性は、具体的なアプリケーション、及びその危害に対する機器製造業者の許容力に依存する。製造業者が採点法を利用して危害の重大性をランク付けしたり、許容可能/許容不可能の二者択一で判定を行うことも多い。危害そのものが特定されるまで、重大性を予測できない場合が多い。そのような場合、最も重大な危害をリストの最上部、あまり重大でないものをリストの最下部に配置した相対的なスケールを使って危害の重大性をランク付けすると役に立つこともある。ランク付けしたリストをチェックすれば、どのレベルが許容可能で、どれが不可能かを明確にすることもできる。

尤度（Likelihood）

自動化プロセスソフトウェアの障害に起因する潜在的な危害発生の尤度に影響を及ぼす可能性がある要因として、以下の例が挙げられる。

- 障害の結果を察知する能力。
- 障害を改善又は障害に起因する危害を緩和する下流のリスク予防策の有無及び有効性。
- 二次的または同時に発生して危害を生じるイベントの数（相乗的蓋然性 ）。
- 二次的なイベントの尤度。
- 危害をもたらす可能性がある二次的なイベントの平行パスの数（相加的因果性 ）

障害の尤度は、障害の確率に関連するか、それと同じである。学術研究の多くは、ソフトウェア障害の確率を予測するための数量モデルや、残された欠陥数の数量化に重点を置いてきた。これらの推定法に関わるパラメータの数は多く、その大半は主観的な尺度に基づいている。

ソフトウェア検証のリスク評価に利用できる比較的単純な方法として、相対語でのみ尤度を考える方法が挙げられる。例えば、ほとんどの人は、複雑なソフトウェアの方が単純なソフトウェアよりも障害の発生率が高く、試験を経たソフトウェアの方が未試験のソフトウェアよりも障害の発生率が低いと考えるだろう。尤度又は確率がどの程度減少するかを数値化する試みがどれほど有意義なものかは、議論の余地がある。

これを数学的に判定してくれる絶対確実な科学は存在しない。ソフトウェア検証及び設計・開発の優良実施の作業は、障害の確率を上昇させるソフトウェアの属性を減らし、障害の確率を低下させる属性を増やすことに重点を置かなければならない。

つまり、この文書のコンテクストでは、障害が結果的に直接的又は間接的な危害をもたらす場合に限り、障害の蓋然性は興味深いものになる。ビジネスのリスク、規制のリスク、又はその両方について検討が行われていれば、それらのカテゴリーで危害を生み出す障害も興味深いものになる。

自動化プロセスソフトウェアの障害尤度を取り扱うアプローチの一つとして、ソフトウェアが障害を起こすのを当然とみなすことが挙げられる。自動化プロセスソフトウェアの障害に起因する危害は、医療機器を介して間接的な影響を及ぼすことが多いため、下流で障害を見つけたり、リスクを予防する機会は数多く存在する。それらの機会をリスク予防策に利用することもできるし、障害に起因する危害のポテンシャルを減らす目的で既に利用している例もある。

また、ソフトウェア障害の後又はそれと同時に他の何らかのイベントが発生した場合、危害はソフトウェア障害に起因するものだけになる。因果連鎖におけるこれらのイベントの尤度は、ソフトウェアと障害が危害をもたらす尤度に影響を及ぼす。

リスク低減努力の優先順位を決定するには、患者、ユーザー又は環境への潜在的な危害の重大性とその危害の原因が発生する尤度を組み合わせて検討しなければならない。

リスク又は残留リスクの容認性

おそらくリスクマネジメントで最も困難な活動は、リスクの許容可能レベルを決定することだろう。これは、潜在的な危害の重大性に大きく依存する。各製造業者は、リスクの容認性を定義・文書化するための基準を設定し、それらの基準に適合した評価を可能にするようなフォーマットであらゆるリスクを特定する必要がある。一般的に、ある人が同僚や管理職、監査担当者に無理なく妥当性を証明できるレベルにまでリスクの許容可能レベルを下げることができれば、それは適切なレベルと考えられる。

容認性の閾値を提案するのはこの文書の守備範囲外だが、そのレベル設定プロセスについて、いくつかの案を紹介するのは構わないだろう。

- 具体的にする。“できる限り低く”とか、“他の製品と同じくらい安全”などといった許容基準は役に立たない。許容基準は、許容基準を満たしていることを客観的に判断できるように、試験用の仕様書と同様に文書化されなければならない。
- 早い段階で許容基準を特定する。危害の潜在的なリスクが特定されたら、早急に目標又は使用を決定する。リスク予防策を講じる前に、許容性に関する

目標を設定することが重要である。リスク予防策を講じた後、容認性の認識はより高レベルのリスクへと向かう場合が多い。許容基準を事前に文書化し、プロセスが逸脱しないようにすること。

- リスクの許容決定を裏付ける根拠を文書化する。これは、将来的に自動化プロセスの保守を行い、思考プロセスを規制当局の調査官に伝達する上で有用である。

リスク予防策

リスク予防策とは、システム内で特定された危害に起因するリスクを低減する対策のことである。このレポートの主旨に関連があるのは、ソフトウェア障害に起因する危害のリスク予防策である。

リスク予防策の形式はさまざまだが、常に全体的な生産又は品質システムのコンテクストで考慮されなければならない。すべてのリスク予防策が同じようにリスク低減に効果的とは限らない。結果的に生じた危害の重大性、又は危害の発生率を低下させることで、リスクを低減できることを忘れてはならない。

ソフトウェア障害に対するリスク予防策は、ソフトウェア本体の外側で履行されることが多い（ウォッチドッグタイマー、スレーブプロセッサー、又はトランザクションロギング）。生産システムの場合、ソフトウェア障害に起因する生産上の欠陥リスクは、生産された部品の下流での適切な検査を要求することで予防できる場合が多い。

下流の検証

組み込まれた生産プロセスソフトウェアは、アクセスが困難で、製造業者から詳細な情報を入手できない場合が多い。よくある例として、マシンツールに組み込まれたソフトウェアが医療機器の製造に使われることがある。ソフトウェアをスタンドアロンの単位で検証する場合、このタイプのソフトウェアを意図する使用について検証するのは困難なこともある。

このような状況で特に効果的なリスク予防策は、ソフトウェアのアウトプット、又はそのソフトウェアが制御する機器のアウトプットを下流で検証する方法である。つまり、ソフトウェアによって自動化されたプロセスのアウトプットを監視して潜在的に有害な欠陥をチェックすれば、ソフトウェアが意図する使用に適合していることを直接判定できる。こうすれば、ライフサイクルコントロールの手法を採用して、ソフトウェアが意図する使用に適合していることを証明する代わりとなる。この手法は、パーツ単位、又は統計的に決定されたパーツのサンプリングごとにチェック可能で、ごく少数の重要な操作を自動化しているプロセスにのみ有効である。検証技師は、下流

での検証を代用した根拠、及び継続的な検証ではなく、サンプリングによる検証を選択したことを正当化するために利用した前提について詳しく説明し、それらの前提を検証しなければならない。

下流での検証は、他のリスク予防策と同様に文書化しなければならない。後のコスト削減措置で排除されることがないよう、検証プロセスがリスク予防策の一つである旨を文書化することが特に重要である。また、規則によって検証の“客観的な証拠の用意”が求められており、その確認が検証の大部分の代わりになることから、下流での検証の結果を文書化する必要もある。製品の進化に伴い、ソフトウェアによって自動化されるプロセスの意図する使用も進化する。その一例として、当初は医療機器のコンポーネントで一つの重要なオペレーションを実行していたマシンツールについて考えてみよう。後に、医療機器の設計がわずかに変更され、ソフトウェアで動作するそのマシンツールに二つの重要なオペレーションが要求されるようになった。このマシンツールの意図する使用が変更された結果（安全上重要なオペレーションが一つから二つに増加）、下流での検証も両方のオペレーションをチェックするように変更されなければならない。

下流での検証プロセスを確認する

下流での検証は、マニュアル又はその他の人的なオペレーションによって遂行される。その例として、エッジの仕上げや機械的整合性の目視検査、又は機械的誤差や電気的導通のマニュアル測定などが挙げられる試験のタイプに関係なく、もしそれがソフトウェアによって自動化されたプロセスの下流検証で、その自動化プロセスのためのリスク予防策として利用されているのなら、検証試験を文書化しなければならない。試験官のための試験手順を明記し、試験項目ごとに結果の許容範囲を明確に定義する。試験官もまた、自動化プロセスのアウトプットをテストするための手順を実行したことを証明する文書を提出しなければならない。

リスク予防策としてのソフトウェア検証

ソフトウェア検証は、ソフトウェア障害の尤度を低減させることから、リスク予防策の一つといえる。残念ながら、その尤度がどの程度下がるのかは、今のところ判明していない。従って、ソフトウェア検証はリスク予防の最後の手段と考えるべきである。なぜなら、ソフトウェアコンポーネント以外のリスク予防策は、重大性のみならず、確率さえも軽減すると考えられ、それによって危害の尤度を大幅に低減できるからである。

より付加価値の高い検証活動を利用して開発と試験が行われたソフトウェアは、ほとんど検証活動が行われずに開発と試験が行われたソフトウェアよりも、故障する確率

が少ない。もしリスクマネジメントがソフトウェア検証活動に大きく依存するのであれば、リスク（重大性に重点を置く）が高いと認識してもらえるように、より深く詳細な客観的証拠を用意することがますます重要になっているといえる。客観的証拠は、ソフトウェアの開発と試験が優良実施規則に基づいて行われたことを明記した文書で構成される。このアプローチは、ソフトウェア障害に起因する危害の重大性又は尤度をコントロールするプロセス又はシステムに計画したリスク予防策を採用する方法ほど望ましいものではない。

リスクモデルの例

下記のリスクモデルは、14971 のリスクモデル（図 B1）で使用したものと同じように、アンケートの結果に基づいてソフトウェアに起因する危害の重大性を高レベル、中レベル及び低レベルに分類したものである。

これはリスクモデルの一例に過ぎない。プロセスやソフトウェアのリスクを、高中低のレベルに分類する方法は他にもたくさんある。これはあくまでも一つのモデルの一例である。

リスク評価

	リスク評価アンケート	Yes または No で回答すること Yes の場合はリスク番号を記入すること （例：リスク#1、リスク#2、…..リスク#n）
1.1 **製品の安全性** （危害）	ソフトウェアが故障した際に懸念される製品安全性への潜在的なリスクは存在するか？ • 患者への危害 • オペレーターへの危害 • 傍観者への危害 • サービス担当者への危害 • 環境への危害	
1.2 **製品の安全性** （危害）	ソフトウェアのユーザーがミスを犯した際に懸念される製品安全性への潜在的なリスクは存在するか？ • 患者への危害 • オペレーターへの危害 • 傍観者への危害 • サービス担当者への危害 • 環境への危害	
2.1 **製品クオリティ**	ソフトウェアが故障した際に懸念される製品クオリティへの潜在的なリスク（安全性のリスクを除く）は存在するか？	
2.2 **製品クオリティ**	ソフトウェアのユーザーがミスを犯した際に懸念される製品クオリティへの潜在的なリスク（安全性のリスクを除く）は存在するか？	
3.1 **記録の完全性**	記録を保管するシステムに記録の完全性への潜在的なリスクは存在するか？ • 記録消失 • 記録損傷	
4.1 **FDA/ISO 規格の遵守証明**	規格遵守を証明する能力に関する潜在的なリスクは存在するか？ • 記録消失 • 記録損傷 • 規制プロセス要件（マネジメントコントロール、CAPA、サービス及びサポートなど）に対する自動化プロセスの不適合	