附属文書

「SaMD の品質管理システム（SaMD-QMS）確立に必要なソフトウェアライフサイクルプロセスの要求事項に係るガイダンス作成と考え方に関する研究」 ガイダンスを補足する IEC 62304 附属ドキュメント

「SaMD の品質管理システム（SaMD-QMS）確立に必要なソフトウェアライフサイクルプロセスの要求事項に係るガイダンス作成と考え方に関する研究」

研究班　編

○　文書のレイアウトについて

IEC 62304 の幹である箇条 4 から箇条 9 までの内容に関して逐次的にその概要等を示したものを研究班の見解としてまとめてあります。正確な内容はガイダンスと同様、規格等に基づく解釈が適切であり、あくまでもこれらの概要を理解するツールの１つとしてご利用いただければと思います。なお、各箇条について、下記のレイアウトに基づいて作成されています。

箇条	細分箇条	概　要　解　説	簡　易　解　説	リファレンス・ポイント
箇条 4 「一般要求事項」		IEC 62304 の箇条 4「一般要求事項」は、医療機器ソフトウェアのライフサイクルプロセス全体に対して適用される基本的な要件を定めたものである。本箇条では、ソフトウェア開発及び保守において確保すべき品質、安全性、及びリスクマネジメントに関する枠組みが示されており、規格の根幹を成す部分である。具体的には、開発において品質マネジメントシステム（QMS： Quality Management System）の導入を義務付け、リスクマネジメントとの連携を求めている。また、ソフトウェアが安全に関わる度合いに基づく「ソフトウェア安全クラス」の分類方法が示され、それに応じて必要な要求事項が異なる点も特徴である。さらに、既存のソフトウェア資産、いわゆる「レガシーソフトウェア」の取り扱いに関しても明確な基準が設けられており、新規開発と同様にその安全性を保証するための適切な管理が求められている。このように、箇条 4 は以降の詳細なプロセス要求（5～9）を適用するための前提条件として機能し、全体の［illegible］体制の基礎を築くものであ	IEC 62304 の箇条 4「一般要求事項」は、医療機器ソフトウェアを安全に作って使うために、開発する会社が最低限守らなければならないルールの基本方針を示したものである。この章ではまず、「ソフトウェアの開発や管理は、決まったやり方に沿って、きちんと手順を作って進めましょう」としており、次のようなことが重要である。 ・ 品質マネジメントシステム（QMS： Quality Management System）を持つこと。これは「ミスを減らし、質の良いものを作る仕組み」のこと。 ・ リスクマネジメントを行うこと。つまり、「人に危険がないか」を考えて、それを防ぐようにすること。 ・ ソフトウェア安全クラスを正しく決めること。どれだけ人の健康に関係があるかで、作り方の厳しさが変わる。 ・ 古いソフト（レガシーソフトウェア）を使うときには、その安全性をよく確認するこ このように、［illegible］るための「基本のルール［illegible］頑張っても、製品の安全［illegible］	1） GB（※）: P56-P82 2） IEC 62304 箇条 4 3） IEC 62304 の各箇条にまたがって関係する４つの柱（細分箇条 4.1 から 4.4）を意識する。 4） 「医療機器ソフトウェア」は、本ドキュメントでは、医療機器の一部としてのソフトウェア（SiMD： Software in Medical Device）や医療機器としてのソフトウェア（SaMD： Software as Medical Device）を含む。 （※）GB：IEC 62304 実践ガイドブック（じほう）
	4.1 品質マネジメントシステム	（１）序論 IEC 62304 の細分箇条 4.1 は、医療機器ソフトウェアの開発と保守において、品質マネジメントシステム（QMS）の導入が必須であることを明記している。本項目は、開発者が自らの組織においてソフトウェアの品質を一貫して確保し、法的要求や規格への適合を果たすための管理体制を整備することを目的としている。特に、医療機器という人命に関わる製品においては、開発過程の信頼性と透明性を担保することが極めて重要であり、それを実現する仕組みが QMS である。	● はじめに 細分箇条 4.1「品質マネジメントシステム」は、医療機器ソフトウェアを［illegible］る会社が、安全で信頼できるソフトウェア［illegible］み」を持たなければならないことを定め［illegible］安定して作るためのルールブックとその［illegible］ステム（QMS）である。これがなければ［illegible］ラなやり方でソフトが作られ、命に関わ［illegible］に危険である。	1） P58-P65 EC 62304 細分箇条 4.1 ISO 13485 の QMS の概念を IEC 62304［illegible］要求を考慮して適用する。

各箇条の概要を整理

各細分箇条に要求される内容や概要を整理

概要解説は、IEC 62304 本文、附属文書やガイドブック等の内容を参考に、一般的な読み手（医療機器開発を目指す）にもわかるような方針で作成

簡易解説は、概要解説への補助的説明やイメージをわかりやすく捉えられるような方針で作成

リファレンスする規格、通知、書籍等の内容やポイント、記載箇所等を記載

【目　次】

※　IEC 62304（JIS T 2304）箇条１から３は、規格の概要、引用規格、定義等が記載されています。これらについては、IEC 62304 の内容を参照してください。

○　IEC 62304 の箇条４の概説

箇条	細分箇条	概　要　解　説	簡　易　解　説	リファレンス・ポイント
箇条 4 「一般要求事項」		IEC 62304 の箇条 4「一般要求事項」は、医療機器ソフトウェアのライフサイクルプロセス全体に対して適用される基本的な要件を定めたものである。本箇条では、医療機器ソフトウェアの開発及び保守における品質、安全性の確保、及びリスクマネジメントに関する枠組みが示されており、規格の根幹を成す部分である。具体的には、開発において品質マネジメントシステム（QMS：Quality Management System）の導入を義務付け、リスクマネジメントとの連携を求めている。また、ソフトウェアに起因する人への危害のリスクに応じた「ソフトウェア安全クラス」の分類方法が示され、それに応じて必要な要求事項が異なる点も特徴である。さらに、過去に開発されたソフトウェア資産、いわゆる「レガシーソフトウェア」の取り扱いに関しても明確な基準が設けられており、新規開発と同様にその安全性を保証するための適切な管理が求められている。このように、箇条 4 は以降の詳細なプロセス要求（箇条 5～9）を適用するための前提条件として機能し、全体のソフトウェアライフサイクルにわたる一貫した管理体制の基礎を築くものである。	IEC 62304 の箇条 4「一般要求事項」は、安全な医療機器ソフトウェアを作るために、開発する組織が最低限守らなければならないルールの基本方針を示したものである。この章ではまず、「ソフトウェアの開発は、決まったやり方に沿って、きちんと手順を作って管理して進めましょう」としており、次のようなことが重要である。 ・ 品質マネジメントシステム（QMS： Quality Management System）に基づいて手順を構築し、手順通りに実施すること。これは「ミスを減らし、質の良いものを作る仕組み」を作ること。 ・ リスクマネジメントを行うこと。つまり、「人への危害が発生しないか」及び「人への危害が発生する可能性を低減させられるか」を考えて、対応すること。 ・ ソフトウェア安全クラスを正しく決めること。どれだけ人への危害の影響の可能性に応じて、作り方の厳格性が変わる。 ・ 古いソフト（レガシーソフトウェア）を使うときには、その安全性をよく確認すること。 このように、箇条 4 は、ソフトウェアの安全な開発をスタートさせるための「基本のルールブック」である。これを守らないと、あとでどんなに頑張っても、製品の安全は保証できないとされている。 また、安全とは、人への危害の影響と可能性とによるリスクが十分に低減されていることである。	1） GB（※）：P56-P82 2） IEC 62304 の各箇条にまたがって関係する４つの柱（細分箇条 4.1 から 4.4）を意識する。 3） ISO 13485 箇条 0　序文 4） 「医療機器ソフトウェア」は、本ドキュメントでは、医療機器の一部としてのソフトウェア（SiMD： Software in Medical Device）や医療機器としてのソフトウェア（SaMD： Software as Medical Device）を含む。 （※）GB：IEC 62304 実践ガイドブック（じほう）
	4.1 品質マネジメントシステム	（１）序論 IEC 62304 の細分箇条 4.1 は、医療機器ソフトウェアの開発と保守において、品質マネジメントシステム（QMS）の導入が必須であることを明記している。本項目は、開発者が自らの組織においてソフトウェアの品質を一貫して確保し、法的要求や規格への適合を果たすための管理体制を整備することを目的としている。特に、医療機器という人命に関わる製品においては、開発過程の信頼性と透明性を担保することが極めて重要であり、それを実現する仕組みが QMS である。	●　はじめに 細分箇条 4.1「品質マネジメントシステム」は、医療機器ソフトウェアを作る会社が、安全で信頼できるソフトウェアを作るために「品質を管理する仕組み」を持たなければならないことを定めている。簡単に言えば、「良いものを安定して作るためのルールブックとその運用体制」がこの品質マネジメントシステム（QMS）である。これがなければ、バラバラなやり方でソフトが作られ、一定の品質が保たれないことや、ミスが発生しても見落とされてしまうことが起こり得、命に関わる医療機器ソフトウェアとしては非常に危険である。	1） GB（※）：P58-P65 2） ISO 13485 の QMS の概念を IEC 62304 の中の各要求を考慮して適用する。 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）品質マネジメントシステムの定義と意義 QMS とは、組織が製品やサービスの品質を一貫して確保・改善するための方	●　QMS とは何か？ QMS とは、「品質を保つために組織全体で取り組むルールや仕組み」のこと	1） 医療機器ソフトウェアも ISO 13485 と整合した QMS 省令の適用が法令上義務

		針、手順、責任、資源、記録を体系的に整備した仕組みである。ISO 13485 に準拠した QMS の導入を行い、IEC 62304 でもこの国際規格に適合することが前提となっている。このシステムにより、開発チームが属人的な判断に頼らず、組織として統一されたプロセスに基づいて製品を作り上げることが可能となる。また、QMS は設計から製造、保守、変更管理に至るまでの全ての工程をカバーする枠組みであり、ソフトウェア開発プロセスとの整合性が取れていなければならない。	である。例えば、次のようなものが含まれる。 ・ 作業の手順を文書で決めておく -だれが、なにを、いつやるかを決める -作ったものをチェックするルールを作る -ミスが起きたときに、すぐに対応する仕組みを持つ ・ 手順に則って実施し、その活動を実証するための記録を作成する ・ チェック等の結果に基づき手順、ルールを改善する つまり、ソフトウェア開発を「なんとなく」「適当に」やるのではなく、きちんとルールを決めて、ルール通りに進めることにより、顧客に有効で安全なソフトウェア提供をすることが QMS の目的である。	付けられており（薬機法、QMS 省令、QMS 関連通知等）、承認申請時の QMS 調査に耐えうる体制・プロセス、文書管理等の準備が必要とされる。
		（３）IEC 62304 における QMS の要求 IEC 62304 では、ソフトウェアのライフサイクルプロセス（企画、開発、検証、リリース、保守等）を体系的に管理するために、以下のような QMS の要素が必要とされている。 ① 文書化されたプロセスの確立 組織はソフトウェア開発におけるすべての工程について明文化されたプロセス（作業手順）を確立し、それに従って作業を実施しなければならない。これは開発の属人化を防ぎ、誰が関わっても一定の品質を保つための基本となる。 ② 責任と権限の明確化 プロジェクトに関わるすべての役割について、責任範囲と意思決定権限を明確にし、役割ごとの連携がスムーズに行えるようにする必要がある。 ③ トレーニングと教育 設計開発に携わるスタッフが業務を行うために必要な力量を明確にし、力量を取得・維持するための教育訓練を計画的に実施する必要がある。法規要求の理解と、特にソフトウェア開発者やテスト担当者に対しては、リスクベースの思考やトレーサビリティの概念についての教育が重要である。 ④ 記録の保持と管理 開発における全ての成果物や判断記録（設計レビュー、テスト結果、変更理由等）を体系的に保存し、後の検証や監査に備える。これにより、後日問題が発生した際に、原因の特定や是正処置を迅速に行うことが可能となる。	● なぜ QMS が必要なのか？ 医療機器ソフトウェアは、病気の診断、治療、命の管理等に関わる。ちょっとしたバグ（不具合）や設計ミスが、人の命を左右することさえある。そこで、以下のような理由から QMS がとても重要になる。 ・ 作業のぬけ・もれを防ぐため やるべきことを明確に決めておけば、必要な作業が忘れられることがない。 ・ 同じ失敗を繰り返さないため 一度起きたミスの原因を確認し、原因を取り除くための対策を実施し、ミスの再発を防止できる。 ・ 品質のばらつきをなくすため 誰が作業しても、一定の品質で同じ結果を出せるようにする。 ・ 規制に対応するため 国や地域の法律では、QMS の運用が義務となっていることが多い（例：ISO 13485）。 ● IEC 62304 が求める QMS の範囲 IEC 62304 では、「QMS を持ち、その QMS にソフトウェア開発の工程を含めること」を求めている。具体的には、以下のような項目が対象となる。 ・ ソフトウェア開発の計画（開発前に何をどう作るかを決める）	1) SiMD については、ISO 13485 及び IEC 62304 の要求事項への準拠、SaMD については、ISO 13485 に係る無体物に係る要求事項と IEC 62304 への準拠が必要。薬機法における、QMS 省令と基本要件基準第 12 条第 2 項の適用。

		⑤ 継続的な改善活動 品質不具合やプロセス上の問題が発生していないか定期的に評価し、必要に応じてプロセスの見直しや改善を図ることが求められる。これは製品そのものの品質向上だけでなく、開発体制の成熟にもつながる。	・ 要求事項の整理（どんな機能が必要かを明確にする） ・ 設計と実装（プログラムの中身を考え、コードを書く） ・ テスト（動作確認をする） ・ 修正と更新（バグを直したり、バージョンアップする） ・ 記録の管理（誰がいつ何をしたか、あとで確認できるようにする） ・ 構成の管理（どのバージョンのどの部品を使っているか整理） これらを、組織の QMS に組み込んで実施することで、ソフトウェアの品質を保つ仕組みができる。	
		（４）QMS と IEC 62304 の他の箇条との関係 本項の QMS 要件は、IEC 62304 の他のすべての要求事項の土台となる。例えば、箇条 5「ソフトウェア開発プロセス」や箇条 6「保守プロセス」では、それぞれの工程を管理する具体的手順が求められているが、これらは QMS によって管理・実施されることが前提となっている。また、箇条 4.2 で規定されている「リスクマネジメント」も、ISO 14971 に準拠した管理プロセスを QMS 内に組み込むことが期待されており、QMS がリスクマネジメント体制を包含する枠組みとして機能する必要がある。さらに、後述する「ソフトウェア安全クラス分類」や「レガシーソフトウェア」についても、その取り扱い方針や判断プロセスは QMS 内で文書化されていなければならない。	● ISO 13485 との関係 IEC 62304 では、具体的な QMS の仕組みをすべて書いているわけではない。かわりに、ISO 13485 という医療機器の QMS に関する国際規格を使うことを強く推奨している。ISO 13485 では、製品の設計から製造、販売、アフターサービスまで、すべての工程を管理する仕組みを定めている。IEC 62304 はその中の「ソフトウェア開発部分」を詳細に定めた規格であるとも言える。	1) ISO 13485 と IEC 62304 の箇条・細分箇条の要求事項の関係性（IEC 62304：附属書 C、C.2）を参照して、QMS 上求められる医療機器ソフトウェアへの要求事項を IEC 62304 に準拠して明確にする。
		（５）QMS の導入が不十分な場合のリスク QMS が未整備、あるいは形骸化している状態では、以下のような問題が発生しうる。 ・ 設計の一貫性が失われ、再現性のない開発となる ・ 開発・保守作業の根拠が不明確となり、第三者審査に耐えられない ・ 安全上の問題に対する初動対応が遅れ、リコールや事故につながる可能性がある ・ 変更管理が不十分となり、既知リスクの再発やソフトウェアの不整合が生じる このような状況を防ぐためにも、QMS は単なる「書類上の仕組み」ではなく、実際の業務に密接に結びついた運用体制として整備・運用される必要がある。	● ソフトウェア開発部門だけの QMS では足りない QMS は、ソフトウェア開発者だけの問題ではない。ソフトウェアが製品として実際に使用され、保守・メンテナンスからソフトウェアの廃棄に至るまでには、次のような人たちが関わっている。 ・ 設計者 ・ 品質保証担当者 ・ 製造部門 ・ 営業やサービス部門 ・ 経営者 つまり、全社的な体制で「品質を守る仕組み」を作り、それを運用することが QMS の本質である。ソフトウェア開発チームだけが頑張っても、全体の品質は保証できない。	1) SaMD であっても当然 QMS に必要な体制整備（QMS 体制省令）の導入が必要。その体制による QMS の実現（QMS 省令への遵守）が要求される。
		（６） 実運用上必要とされること ・ まずは自社 QMS が ISO 13485 の要件を満たしているかを確認すること	● 実際の運用ではどうするか？ QMS の運用では、次のような活動が日常的に行われる。	

		・ QMS の中でソフトウェア開発に特化した手順を設けること（一般的なハードウェア開発とは異なる点に注意） ・ 内部監査、外部監査やマネジメントレビューを通じて QMS の有効性を定期的に評価し、形骸化を防ぐこと 特に、医療機器の開発・上市経験のない、中小企業やスタートアップ企業においては、QMS 導入に対する負担感が大きい場合があるが、IEC 62304 の要求に対応するためには、最小限でも開発・検証・リリース・保守・変更管理の各手順を明文化し、トレーサビリティを確保する体制が不可欠である。	・ 文書管理（手順書や設計書、テスト結果を整理） ・ 教育訓練（開発者や関係者にルールを教える） ・ 内部監査（決めたルールが守られているかを定期的に確認） ・ 是正処置（問題があったときに原因を分析し、改善する） ・ マネジメントレビュー（経営者が全体の品質状況をチェック） これらを定期的に行い、PDCA（計画→実行→確認→改善）サイクルを回すことで、品質は少しずつ向上していく。 ● 品質文化を育てること QMS があっても、それを形だけで運用していては意味がない。 本当に品質を良くするためには、「間違えたら隠さず報告する」「気づいたことは共有する」「品質を大切に思う文化」を育てることが必要である。つまり、QMS とは「紙のルール」ではなく、「会社の考え方と行動の習慣」でなければならない。細分箇条 4.1「品質マネジメントシステム」は、医療機器ソフトウェアを安全かつ確実に作るための基本的な土台を整える規定である。良い製品は、偶然ではなく、「良いプロセス」から生まれる。だからこそ、IEC 62304 ではソフトウェアを作る前提として、しっかりとした QMS の整備と運用を求めている。	
	4.2 リスクマネジメント	（１）序論 細分箇条 4.2「リスクマネジメント」は、医療機器ソフトウェアにおいて、患者や使用者に対する危害の可能性を適切に予測・評価し、必要な対策を講じる仕組みを構築することを求めた条項である。医療機器は生命や健康に直結するものであり、その制御や判断を担うソフトウェアに不具合があった場合、重大な事故や死傷に至るおそれがある。したがって、ソフトウェア単体であっても、リスクマネジメントの考え方は必須であり、本規格では ISO 14971 に準拠した実践が要求されている。	● はじめに 細分箇条 4.2「リスクマネジメント」は、医療機器ソフトウェアを開発・使用するうえで、人に危害（ケガや命の危険等）を及ぼす可能性がある問題（リスク）を見つけて、できるだけ減らすためのルールを決めることを求めている。「リスク」とは、「悪いことが起きるかもしれない可能性」のことである。例えば、心拍数を測る機器のソフトが間違った数値を表示したら、患者の状態を誤って判断してしまい、治療を遅らせる危険がある。リスクの大きさは、その「悪いこと」がどの程度深刻なものなのか(危害の大きさ)と、実際に起きる可能性(発生頻度)によって決まる。 そういったリスクを見つけ、対応策を立てて、実際に安全であることを確かめる。これがリスクマネジメントである。	1) GB（※）: P66-P73 2) ISO 14971 におけるリスクマネジメントの考え方を IEC 62304 の各箇条に要求されるリスクマネジメントの規範として適用する。 （※）GB : IEC 62304 実践ガイドブック（じほう）
		（２）リスクマネジメント ここでいう「リスクマネジメント」とは、単なる危険の洗い出しではなく、以下の一連の活動全体を指す。 ① ハザード（危険源）の特定 どのような状況・入力・動作が危険につながるかを洗い出す。例として、誤ったアラーム通知、薬剤投与量の誤表示、データの欠落等が挙げられる。	● リスクマネジメントとは何か？ リスクマネジメントとは、次のような流れで行われる活動である。 ・ リスクを見つける（リスクの特定） どんな危険が起きるかを想像し、書き出す。 ・ リスクの大きさを調べる（リスクの評価）	1) ISO 14971 の適用指針 ISO/TR 24971 : 附属書 A「ハザード及び安全に関する特質の明確化」、附属書 B「リスク分析を支援する技法」、附属書 C「方針、リスクの受容可能性の判断基準、リスクコントロール及びリスク評価の関係」、附属書 D「安全に関する情報及び残留

		② ハザードが危害に至るまでのシナリオ（危害の起こり方）を分析する 例えば、「ソフトウェアがセンサーデータを誤認識する」→「アラームが鳴らない」→「治療の遅延」→「患者が重症化する」といった因果関係を分析する。 ③ リスクの見積もりと評価 起こる可能性の大きさと結果の重大さを組み合わせてリスクの大きさを見積もり、許容できるか否かを判断する。リスクが高ければ対策が必要となる。 ④ リスクコントロール（低減策）の計画と実施 ソフトウェア設計や警告表示、使用条件の制限等により、リスクを低減または除去する。 ⑤ 残留リスクの評価と文書化 すべての対策を講じた後でも残るリスク（残留リスク）が受容可能であるかを判断し、適切な記録を残す。 このプロセスは、ソフトウェア開発の初期段階から終了後の保守まで、継続的に繰り返される必要がある。	どれだけ大きな影響があるか、どのくらいの頻度で起きるかを分析する。 ・ リスクを減らすための対策を考える（リスクコントロール） 設計の見直し、警告表示の追加、テストの強化等。 ・ リスクが本当に減ったかを確認する（検証） 実験やシミュレーションで、安全になったことを確かめる。 ・ 残ったリスクが許されるレベルかを判断する（残留リスクの評価） 完全にゼロにはできない場合でも、「このくらいなら許容できる」と判断する。	リスクの情報」、附属書 E「リスクマネジメントにおける JIS 又は国際規格の役割」、附属書 F「セキュリティに関連するリスクについての指針」が用意されており、医療機器ソフトウェアも同様にリスクマネジメントを行う。とくにサイバーセキュリティについては、この ISO/TR 24971：附属書 F において、セキュリティに含まれることが言及されている。ソフトウェアの一セキュリティについては、IEC 81001-5-1 に示されており、ISO 14971 のリスクマネジメントに基づき IEC 81001-5-1 の要求事項（箇条 7）に準拠した対応が求められる。
		（３）IEC 62304 における具体的な要求事項 IEC 62304 では、以下のようなリスクマネジメント活動の統合を要求している。 ① ISO 14971 との整合性 IEC 62304 では、リスクマネジメントの全体的なプロセスについて、ISO 14971 に従うことを求めている。これは、医療機器に特化したリスクマネジメント規格であり、設計・製造・使用すべてのフェーズにおけるリスクの特定・評価・管理に関する国際的な標準である。具体的には、ソフトウェア固有のリスク（例えば、数値演算誤差、フリーズ、通信異常等）についても、ISO 14971 の原則を適用することになる。 ② ソフトウェアのリスクとの関係性の明確化 ソフトウェアが単独で危害を生じることは少ないが、「他のシステムの動作を制御している」「不具合が間接的に危害を誘発する」といった形で危険を引き起こす可能性がある。IEC 62304 では、ソフトウェア単独の視点にとどまらず、医療機器全体のリスクの中でソフトウェアが果たす役割を明確にし、システム全体としてのリスク軽減を目指すべきであるとしている。	● IEC 62304 での位置づけ IEC 62304 では、ソフトウェアのライフサイクル（設計・開発・保守）全体にわたって、このリスクマネジメントを一貫して行うことを求めている。そして、このリスクマネジメントは、ISO 14971 という別の国際規格に沿って進めることが前提とされている。つまり、「安全なソフトウェアを作るには、ちゃんとリスクマネジメントをしましょう」というのがこの箇条の趣旨である。 ● なぜソフトウェアにリスクがあるのか？ 一見すると、ソフトウェアは目に見えず、壊れるものでもないので、「危険がない」と思われがちである。しかし、実際には次のような理由で、ソフトウェアにも大きなリスクがある。 ・ 表示が間違っていたら、人が誤って行動する ・ アラームが鳴らなかったら、緊急事態に気づけない ・ 意図せずデータを消してしまったら、正しい診断ができなくなる ・ ソフトウェアが誤った領域を指し示したら、正しい診断ができなくなる ・ 動作が止まってしまったら、治療が中断される	1) 本ドキュメント：箇条 7「ソフトウェアリスクマネジメントプロセス」を参考にする。

③ ライフサイクル全体を通じた継続的なリスクマネジメント

リスクは設計段階だけでなく、テスト、リリース、運用、保守といった各フェーズで新たに発生することがある。したがって、リスクマネジメントは一回限りのイベントではなく、ソフトウェアの変更が生じるたびに更新されるべき動的なプロセスである。例えば、ソフトウェアのアップデートにより、新たなモジュールが追加された場合、その影響を受ける他モジュールや安全機能に対してリスク再評価を行う必要がある。

④ トレーサビリティの確保

リスク識別、リスクコントロール策、検証結果等は、全て文書化され、関係する要件やテスト、実装にトレーサビリティが確保されていなければならない。これにより、後日問題が発生した際の影響範囲の特定や、是正措置の正当性を示すことが可能となる。

つまり、ソフトウェアそのものが危害を与えるのではなく、人の判断や機械の動作に影響を与えて、間接的に危害が起きる。

（４）医療機器ソフトウェアに特有のリスク

医療機器ソフトウェアは、汎用ソフトウェアと異なり、使用者や患者に対する安全性を考慮し、以下のようなリスク等に注意を払う必要がある。

- リアルタイム処理の失敗：タイミングの遅延や優先順位処理のミスにより、重大な情報が見逃される。
- 通信エラー：外部センサーや制御機器との連携ミスにより、誤った情報に基づいた制御が行われる。
- 使用環境の多様性：想定外の使用状況や誤操作により、異常状態に陥る可能性。
- ソフトウェアアップデート時の影響：一部の変更が、他の安全機能へ意図せず悪影響を及ぼすこと。

これらに限ったことではないが、リスクを網羅的に捉え、未然に対処するためにも、リスクマネジメントは必須の取り組みである。

● どんなタイミングでリスクを考えるべきか？

リスクは、以下のようなソフトウェアの活動ごとに考える必要がある。

活動	リスクを考えるべきポイントの例
要求仕様を決めるとき	使い方を誤るとどうなるか？
設計を行うとき	この構造で失敗しやすいところは？
コーディングするとき	条件を見落としたら、危ない？
テストするとき	テスト項目に抜けはないか？
保守・修正するとき	直したことで他が壊れていないか？

このように、ソフトウェアの始まりから終わりまで、ずっとリスクを考え、活動することが重要である。

● リスクの大きさをどう評価するか？

リスクは、一般的に次の 2 つを組み合わせて評価する。

- 発生の可能性（頻度）
  よく起きる？ まれに起きる？ ほとんど起きない？

- 結果の大きさ（重篤度）
  命に関わる？ 少しのケガ？ 無害？

			この 2 つをかけ合わせて、「リスクの重大度」を評価し、それに応じて対策の優先順位を決める。例えば 頻度 / 重篤度 / リスクの重大度 高い / 大きい / 非常に危険（すぐ対応） 低い / 小さい / 許容可能か検討する ● どうやってリスクを減らすか？ リスクを減らす方法にはいくつかある。例えば ・ ソフトの設計を変える（そもそも危険な操作をできないようにする） ・ 安全機能を追加する（異常を検知して止める） ・ 警告や確認画面を表示する（人が気づけるようにする） ・ マニュアルや教育を強化する（使い方を間違えないようにする） これらは、リスクコントロール（リスク対策）と呼ばれ、設計の一部として必ず検討される。 ● 残留リスクとは？ どれだけ対策しても、完全にゼロにできないリスクがある。例えば、電子機器である以上、停電したら止まる可能性はゼロにはできない。こうしたリスクを残留リスクという。この残留リスクが「受け入れられるかどうか」を、開発者だけでなく、医療者や使用者の視点でも判断する必要がある。 ● 記録がとても重要 リスクマネジメントでは、「何をどう考えて、どう対処したか」を記録に残すことが非常に大切である。この記録（リスクマネジメントファイル）は、将来のトラブルの原因を調べるとき、品質を確認するとき、規制当局に説明するとき等、さまざまな場面で役に立つ。	
		（５）結論 細分箇条 4.2「リスクマネジメント」は、単なるリスク評価の指示ではなく、ソフトウェア開発全体に「安全性を組み込む文化」を築くことを目的とした要求である。IEC 62304 では、リスクの特定・評価・制御という一連の流れを、ソフトウェアのあらゆる工程に統合することが求められており、それを支える基盤として品質マネジメントシステムとの連携が不可欠である。開発者は、「安全性は後から検査で確認するものではなく、設計時点から計画的に確保すべきものである」という意識を持ち、リスクマネジメントを単なる文書作	● まとめ 細分箇条 4.2「リスクマネジメント」は、「人の命を守るために、ソフトウェアの危険をあらかじめ見つけて、防ぐ方法を考えましょう」という、医療機器ソフトウェアにおいてもっとも重要な考え方の一つである。安全は、偶然ではなく、「考えること」と「準備すること」から生まれる。だからこそ、ソフトウェア開発では、機能よりも先にリスクを考えることが求められる。	

<table>
<tr><td></td><td></td><td>業にとどめず、実務に根差した取り組みとして定着させなければならない。</td><td></td><td></td></tr>
<tr><td></td><td>4.3 ソフトウェア安全クラス分類</td><td>（１）序論
細分箇条 4.3「ソフトウェア安全クラス分類」は、ソフトウェアが医療機器の中でどの程度の安全性に関与しているかを評価し、その結果に基づいて必要となる開発活動の厳しさ（厳格さ）を定めるための基準である。本分類は、医療機器ソフトウェアが人の生命や健康に与える影響の大きさに応じて、適切な安全性対策が講じられるように設けられたものであり、IEC 62304 全体を適切に適用するための重要な出発点となる。</td><td>● はじめに
細分箇条 4.3「ソフトウェア安全クラス分類」は、医療機器ソフトウェアがどれくらい人の命や健康に関わるかによって、「安全に作らないといけないレベル（厳しさ）」を 3 つのクラスに分けるためのルールである。簡単にいえば、「このソフトはもし失敗したらどれだけ危険なのか？」を考えて、それに応じた開発のやり方を決めましょう、という内容である。危険が少ないソフトなら簡単なルールでもよいが、命にかかわるソフトはとても厳しくチェックしないといけない。これが「安全クラス」の考え方である。</td><td>1） GB（※）：P74-P77

（※）GB：IEC 62304 実践ガイドブック（じほう）</td></tr>
<tr><td></td><td></td><td>（２）なぜ安全クラスの分類が必要なのか
医療機器ソフトウェアは、単に情報を表示し参照するものから、治療や生命維持の制御を担うものまで、その機能の重要度は多岐にわたる。すべてのソフトウェアに同一の厳しさで開発要求を課すのは非効率かつ不適切であり、ソフトウェアが果たす役割に応じて、リスクに見合った管理の水準を設定する必要がある。そこで IEC 62304 は、ソフトウェアの安全性への影響度に基づき、3 つの「安全クラス分類」を定義している。これにより、より重大なリスクを持つソフトウェアには厳格な開発・検証プロセスを要求し、逆にリスクが低いソフトウェアには簡素化された要求を適用する柔軟性を持たせている。

（３）3 つのソフトウェア安全クラスの定義
IEC 62304 では、以下の 3 つの安全クラスが定められている。

① クラス A：危害を引き起こすことが「ない」ソフトウェア
クラス A に該当するのは、ソフトウェアに不具合があったとしても、患者や使用者に対して危害を引き起こす可能性がないと合理的に判断されるものである。例えば、非臨床用の分析やログ収集、教育用シミュレーション等の機能が該当する場合がある。クラス A のソフトウェアには、他のクラスに比べて軽減された要求事項が適用される。特に、設計文書の詳細度や検証活動の厳格さにおいて、より簡略な手続きが認められている。

② クラス B：不具合が発生しても「軽度の危害」にとどまるソフトウェア
クラス B は、ソフトウェアに不具合が生じた場合に患者や使用者に危害が発生する可能性があるが、その危害が「軽度」であり、重大な健康被害に至らないと判断されるものが該当する。例としては、診断補助的なモジュールやアラーム監視が該当する可能性がある。ただし、危害の重大性と発生確率を評価した結果、重大な結果をもたらす恐れがないと明確に証明できる必要がある。このクラスでは、開発文書の整備やリスクマネジメントの適用が求められる</td><td>● クラスは 3 つに分かれている
IEC 62304 では、ソフトウェアの安全性に応じて、以下の 3 つのクラスに分類している。

<table><tr><th>クラス</th><th>意味</th></tr><tr><td>A</td><td>ソフトに問題があっても、誰にも危害がないもの</td></tr><tr><td>B</td><td>ソフトに問題があると、軽いケガ等の危険があるもの</td></tr><tr><td>C</td><td>ソフトに問題があると、命に関わるか、重大な健康被害を起こすおそれがあるもの</td></tr></table>
この分類は、作る人が勝手に決めるのではなく、ソフトの役割と、その問題が引き起こす影響に基づいて正しく判断しなければならない。そのソフトの機能ではなく、その機能がどのように使われるかで判断される。

● なぜクラス分類が必要なのか？
同じ医療機器でも、ソフトの役割によってリスクは大きく変わる。例えば、

・ 血糖値を診断の参照のために表示するだけのアプリ：表示ミスでもすぐには命に関わらない → クラス A

・ 手術中に心拍数をモニタリングし、異常時にアラームを出すソフト：誤動作すると見逃しにつながる → クラス B または C

・ ペースメーカーの制御ソフト：止まると命が危ない → クラス C

このように、危険度に応じて「どれだけ厳格に作るか」を変えることで、安</td><td>1） SaMD 及び SiMD のソフトウェアシステムに起因する危害が、その SaMD 及び SiMD を使用するもの（患者、操作者等）に及ぼす影響に応じて、クラス A（最も低いもの）、B、C（最も高いもの）として分類して検討する。類似するものとしては医療機器のクラス分類ルール（医療機器に起因する安全性のリスクに応じて、クラス I（最も低いもの）、II、III、IV（最も高いもの）に分類される）があるが、あくまでこのクラス分類ルールは医療機器全体としての安全性リスクであり、SaMD/SiMD のソフトウェア安全クラス分類（下記 2）参照）と必ずしも一致するものではない（SaMD や、SiMD 自体が直接医療機器全体の安全性リスクに直結する場合には、一致することも考えられうる）。ソフトウェア安全クラス分類の考え方は、以降に述べる IEC 62304 の各段階（プロセス）の要求やアクティビティの厳密さ（例えば、必要とされるドキュメントの差が生じる等）に波及するため、ソフトウェア開発後に見直しを図るということは、開発期間の延長や開発作業の見直しにも関わりうるため絶対に避けたい。このソフトウェア安全クラス分類の適用については、各国</td></tr>
</table>

		が、クラスCほどの厳格さは必要ない。 ③ クラスC：不具合により「重篤な危害または死亡」が生じる可能性があるソフトウェア クラスCは、ソフトウェアの不具合によって、患者の生命を脅かす、または重大な健康被害を引き起こす可能性がある場合に該当する。例えば、薬剤注入量の制御、心拍調律、生命維持管理等、治療行為や致死的状況に直接関与する制御機能を持つソフトウェアがこれに当たる。クラスCのソフトウェアには、最も厳しい開発プロセスが適用され、設計レビュー、テストの網羅性、リスクコントロールの完全性等、すべての要求事項に最大限の適用が求められる。	全なソフトウェアの開発を実現し、危険度が高いものはより厳格に、低いものはそれほど厳格でなく進められるため、効率よく開発ができる。	の規制要求として考え方の差があるため、SaMDやSiMDを含む医療機器の企画・開発段階で、必要に応じて規制当局との摺合せをしておくことが推奨される。 2) 「高度管理医療機器、管理医療機器及び一般医療機器に係るクラス分類ルールの改正について」（薬食発0510第8号通知）
		（４）クラスの決定手順 ソフトウェア安全クラスの決定は、以下のようなプロセスで実施される。 ① 機能ごとのハザード分析を実施 ソフトウェアの各機能について、どのような危害が起こり得るかを洗い出す。 ② リスクの重大性と発生確率を評価 各ハザードがどのような健康被害をもたらすか、その可能性と影響を評価する。 ③ リスクコントロールの効果を加味して再評価 安全対策を講じた場合に、危害が回避または軽減されるかを分析する。 ④ 残留リスクに基づいてクラスを決定 最終的に残るリスクの大きさにより、該当するクラスを判定する。 ここで重要なのは、「リスクコントロールに依存しない」状態でのリスクに基づいてクラスを決めることである。つまり、「安全機能が正しく動作しなかった場合」にも危害が起こらないと確認できたときのみ、より低いクラスとすることが許される。	● クラスの決め方（分類の手順） クラスを決めるには、次のような流れで判断する。 ① ソフトウェアが何をするのかを確認する 例：「血圧の測定値を表示する」「薬を自動で投与する」等 ② そのソフトが正しく動かなかったときに起きる可能性を考える 例：「間違った値を表示してもすぐに影響はない」「薬を多く投与してしまうかもしれない」 ③ 起きた場合の影響（危害）を判断する 人にケガや命の危険があるかどうかで分ける ④ 対策が他にあるかを考える ソフトにミスがあっても、医師が確認して防げるならクラスが下がる可能性もある	
		（５）クラス判定の注意点と落とし穴 ・ 「機能」だけで判断してはならない：表示用、記録用等、一見単純な機能でも、表示ミスが誤治療につながる可能性があれば高クラスに分類される。 ・ SiMDの場合には「機器全体のリスク」ではなく「ソフトウェア単体でのリスク」に焦点を当てる必要がある。SaMDの場合には、それ自体が機器	● 判断における注意点 ・ 一番危険な機能に合わせて全体を分類する 複数の機能がある場合は、もっとも高いリスクを基準とする ・ システムや人のサポートがあるかを考慮する ソフトだけでなく、人間の判断が加わるかどうかも重要	

<table>
<tr>
<td></td>
<td></td>
<td>全体の体をなすので、「機器全体のリスク」と「ソフトウェア単体でのリスク」は同義になる。
・　クラスを下げるために過剰にリスクコントロールに依存する設計は、かえって安全性を損なう</td>
<td>・　あいまいな場合はより高いクラスにする
「たぶん大丈夫」ではなく、安全側に立って判断する</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>（６）クラスによって異なる要求事項の適用範囲
IEC 62304 の各箇条についてのクラスの適用については、IEC 62304 の細分箇条の要求事項に適用されるクラスが記載されている。クラスが上がるほど、要求の厳しさと文書化の負担が増す。開発者は、自らの製品がどのクラスに該当するかを正確に評価し、それに見合ったリソースと体制を整えることが重要である。クラス分類の運用上の考え方として、

・　文書化を怠らないこと：クラス判定の根拠を記録として残し、後の監査や審査に備える必要がある。
・　機能単位でのクラス分離も可能：複数の機能を持つソフトウェアでは、リスクが異なる機能ごとにクラスを分け、それぞれに対応した開発管理を行う手法も有効である。
・　クラス C を想定して設計することで、安全性の高い製品開発が促進される。

等が必要である。</td>
<td>●　クラスの例（イメージ）

<table><tr><th>ソフトの用途</th><th>想定クラス</th><th>理由</th></tr><tr><td>健康記録メモアプリ</td><td>A</td><td>人に直接危害を与えることがない</td></tr><tr><td>自動点滴制御ソフト</td><td>C</td><td>薬の量のミスが命に関わる</td></tr><tr><td>検査結果をグラフ表示するソフト</td><td>B</td><td>見間違えやすく、診断に影響する可能性あり</td></tr><tr><td>医師が手動で確認する補助ツール</td><td>A または B</td><td>医師の判断が入るかどうかがポイント</td></tr></table>
このように、ソフトの「使われ方」まで考えることが重要である。

●　クラスはずっと同じ？
ソフトの使われ方が変わったり、機能追加があった場合には、安全クラスも再評価する必要がある。例えば

・　元は表示だけだったソフトが、警告を出す機能を追加した
・　使う人が医師から患者本人に変わった
・　システム全体の使われ方が変わった

このようなときには、クラスが上がる可能性があるため、見直しが必要である。</td>
<td>1)　ソフトウェア安全クラスへの要求事項は IEC 62304 の各箇条の本文内に記載されているが、その概要として、IEC 62304：附属書 A、A.2「ソフトウェア安全クラス別要求事項のまとめ」、表 A.1 を参照して、各クラスに要求される IEC 62304 の各箇条、細分箇条に係る要求事項を整理、確認しておく。</td>
</tr>
<tr>
<td></td>
<td>4.4 レガシーソフトウェア</td>
<td>（１）序論
「レガシーソフトウェア（Legacy Software)」とは、現在も使用されているが、過去の開発時点で IEC 62304 等の現在の国際規格に従っていなかったソフトウェアを指す。特に医療機器分野においては、旧来の技術や設計資産を再利用する場面が多く、こうしたレガシーソフトウェアの扱いが製品の安全性や規制適合に大きな影響を及ぼす。IEC 62304 における細分箇条 4.4 は、このような既存ソフトウェアを安全かつ適切に活用するための方針を示すものである。すなわち、過去に開発されたソフトウェアであっても、それが再利用される限りは、IEC 62304 の要求事項に適合するように評価・管理されなければならないという考え方が前提である。</td>
<td>●　はじめに
細分箇条 4.4「レガシーソフトウェア」は、すでに昔から使われていて、開発時の記録やテスト結果等が十分に残っていないソフトウェアを、今後も医療機器として使い続ける場合に、どう安全性を確保するかについて定めたものである。「レガシー（Legacy)」とは「古くからの遺産」という意味で、ここでは「昔に作られて、今でも使われている古いソフトウェア」のことを指す。例えば過去において、IEC 62304 のような厳しいルールに従って作られていなかった、別のルールに従って作られていた、全くそのルールがなかった時代に作られたようなソフトウェアが該当する。</td>
<td>1)　GB（※）：P78-P82
2)　ここでいうレガシーソフトウェアは、医療機器レガシーソフトウェアを指す。ソフトウェア関連用語の定義については、GB：P78 の表 4-8 が参考になる。レガシーソフトウェアに関しては、IEC 62304 の現行版に遡って適合している立証はできない。レガシーソフトウェアを特定し、リスクマネジメ</td>
</tr>
</table>

				ントを行い、現行の要求事項と過去の要求事項に係る差分等について、ギャップ分析（IEC 62304 細分箇条 5.2、5.3、5.7 及び箇条 7 による要求事項に対し）を行い、IEC 62304 の箇条 9 による問題解決プロセスによって対応を図る。また、IEC 62304 箇条 6 の保守プロセスにのせること、レガシーソフトウェアを継続使用する根拠を明示（文書化）する必要がある（IEC 62304 細分箇条 4.4.2-4.4.5）。 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）レガシーソフトウェアの典型的な例 ・ 長年使われている医療装置に搭載されている制御ソフトウェア ・ 親会社からライセンスされたが、開発文書が不足している組込みソフト ・ 社内で開発されたが、開発プロセスの記録がほとんど残っていないモジュール ・ 医療機器の次世代モデルに再利用される旧バージョンのコード群 これらのソフトウェアは、機能的には有用であり使用継続が望まれるが、安全性や品質の観点で「ブラックボックス化」しているケースが多い。	● なぜレガシーソフトウェアが問題になるのか？ 古いソフトウェアであっても、現在の医療機器の一部として使われ続けていることがある。ところが、そのようなソフトウェアには次のような問題がある。 ・ 開発当時の設計図やテストの記録がない ・ 誰がどのように作ったのかがわからない ・ ソースコード（中身）が残っていても、コメントや説明が少なくて読めない ・ 当時に開発環境が古すぎ再現できないため、ソースコードから実行モジュールを再現することができない。 このように、安全性を確認するための情報が足りないということが問題である。	
		（３）IEC 62304 の基本的立場 IEC 62304 では、原則としてすべての医療機器ソフトウェアは、同規格に従って開発・保守されなければならないとされている。したがって、既存のソフトウェアを使用する場合には、そのソフトウェアが IEC 62304 の要件にどの程度適合しているかを確認し、不足している場合には以下のいずれかの対応が必要となる。 ・ 追加の文書化やテストを実施して要件適合を図る ・ 明確な理由とリスク評価に基づいて、適合を免除するか代替手段を講じる	● IEC 62304 はどう考えているか？ IEC 62304 では、「レガシーソフトウェアをそのまま使ってもよい」とは言っていない。むしろ、「使い続けるなら、安全性を確認するための追加対応が必要だ」としている。つまり、たとえ古いソフトウェアでも、次のような条件を満たすようにしなければならない。 ・ リスクがないこと、または小さいことを証明できる ・ 問題が起きたときに、対応できる準備がある ・ 使っている人がそのソフトの特性や制限を理解している	1） レガシーソフトウェアに関しては、IEC 62304 の現行版に遡って適合している立証はできない。レガシーソフトウェアを特定し、リスクマネジメントを行い、現行の要求事項と過去の要求事項に係る差分等について、ギャップ分析（IEC 62304 細分箇条 5.2、5.3、5.7 及び箇条 7 による要求事項に対し）を

		これにより、単に「古いからそのまま使う」ことを許さず、開発当時の不透明さを補う責任が開発者側にあることが強調されている。		行い、IEC 62304 箇条 9 による問題解決プロセスによって対応を図る。また、IEC 62304 箇条 6 の保守プロセスにのせること、レガシーソフトウェアを継続使用する根拠を明示（文書化）する必要がある（IEC 62304 細分箇条 4.4.2-4.4.5)。
		（４）適合確認のために求められる活動 IEC 62304 においてレガシーソフトウェアを使用するには、次のような活動を通じてその妥当性を確認しなければならない。 ① 文書の有無と内容の確認 ・ 要求仕様書、設計文書、テスト仕様書、バグ記録等の文書が存在するか確認する ・ 不足している場合は、ソースコードから逆解析を行い、可能な限り仕様を復元する ・ ドキュメントの信頼性や改訂履歴の整合性を評価する ② リスクマネジメントとの整合 ・ レガシーソフトウェアの使用によって生じる新たなリスク（未確認の不具合、設計ミス等）を洗い出す ・ ISO 14971 に従って、レガシー一部分の機能が安全性にどのように影響するかを評価する ・ 必要に応じて、リスク低減のための監視ロジックや独立チェックを追加する ③ テストと検証の実施 ・ ソースコードの解析やブラックボックステストにより、想定外の動作やエラーがないかを確認する ・ 特に安全関連機能については、再テストや回帰テストを徹底する ・ テスト結果を文書化し、第三者が追跡可能な形にする（トレーサビリティの確保） ④ 保守体制の整備 ・ レガシーソフトウェアを今後も使用・修正していく場合、その保守担当が仕様を理解しているかを確認する ・ 保守中にバグが発見された際の影響範囲と、修正後の再検証手順を明文化	● レガシーソフトを使い続けるために必要なこと レガシーソフトウェア含めた医療機器として安全に使うためには、次のようなことが求められる。 ① リスクマネジメントの実施 そのソフトを使うことで、患者に危険が及ぶ可能性があるかを調べ、必要な対策をとる。例えば ・ 表示ミスが命に関わるか？ ・ 操作ミスを誘発しやすいか？ ・ アラームが鳴らないことはないか？ ② 必要な文書をできる限りそろえる 設計書や試験結果がない場合は、現在のソフトの動作から調査し、同じ内容を作り直す努力をする。 ・ ソースコードがあれば読んで仕様をまとめる ・ 実際に動かして、テスト結果を新たに作成する ・ 現在の使用方法や制限事項を明文化する ③ ソフトの安全クラスを判断する 4.3 にあるように、そのソフトが人にどれだけ影響を与えるかによって、安全クラス（A/B/C）を決める。クラスが高ければ、それだけ詳しい情報や対策が必要になる。 ④ 現在の開発プロセスとつなげる レガシーソフトを使って新しいソフトを開発するときは、その部分だけ古い方法で作ったままにせず、現在のルールに取り込むようにする。例えば、 ・ 新しい QMS に対応させる	

		する ・ 修正記録やバージョン管理システムが存在しない場合は導入することが望ましい	・ 修正があった場合は、変更管理プロセスを使う ・ 不具合が出たら、問題解決プロセスに沿って対応する	
		（５）レガシーソフトウェアが与える影響 ① 安全性へのリスク ・ コードの構造が不明確であるために不具合が埋もれている可能性がある ・ 外部ライブラリや OS との互換性が失われている場合、予期せぬエラーが発生する ② 品質管理の難しさ ・ 設計思想が現在の安全要求と合致しないことがあり、レビューの判断が困難 ・ 問題発生時の影響範囲分析ができず、保守対応に時間がかかる ③ 規制対応の不備 ・ ドキュメント不備により、規制当局の審査に通らない場合がある ・ 規制当局への適合性証明で困難をきたす可能性がある	● レガシーソフトを放置してはいけない 「動いているから問題ない」として放置するのは非常に危険である。なぜなら、以下の理由から今は大丈夫でも、将来的に患者の安全を脅かす可能性があるからである。 ・ ハードウェアが故障したとき、復旧できない ・ 保守および修正を行うのが困難になる ・ 法改正や規格変更により、基準を満たさなくなる ・ 他のソフトとの互換性が失われる ・ セキュリティ上の脆弱性がある	
		（６）対応方針の例 レガシーソフトウェアの再利用が必要である場合、以下のような対応方針等が取られる場合がある。 ① 既存コードを元に設計・仕様書・テスト手順を再構築する。リバースエンジニアリングと検証の負担は大きいが、製品寿命を延ばす有効な方法である。 ② ソフトウェア本体をそのままにして、独立監視機能の追加として、安全性を監視する独立モジュールを加えることでリスク低減を図る。 ③ レガシーコードを徐々に新しい設計で再実装し、段階的置き換えによって、長期的に IEC 62304 に完全適合した構成に移行する。	● レガシーソフトにまつわる例 例 1：昔の温度管理ソフト 10 年前に作られた手術室の温度を管理するソフトがある。現在も問題なく動いているが、当時の設計書やテスト結果はもうない。ところが、新しいソフトと連携する必要が出てきた。まずはそのソフトがどんな動きをしているか確認し、リスクがないことを調べたうえで、新しいソフトとの連携仕様を明確にする必要がある。 例 2：メーカーが倒産した古い機器 製造元がなくなってしまった医療機器で、中のソフトだけがそのまま使われている。しかし、バグが見つかってしまった。修正できないなら、ソフトの使い方に制限をつけたり、注意喚起を行って対応する。場合によっては代替機器への置き換えも考える必要がある。 レガシーソフトは「管理された状態」で使い続けることが必要である。つまり、問題があればすぐ対応できるように、情報をそろえておく。	
		（７）結論 IEC 62304 の細分箇条 4.4「レガシーソフトウェア」は、旧来の開発資産の	● まとめ IEC 62304 の細分箇条 4.4「レガシーソフトウェア」は、「古いけど使い続	

		活用に対して現代的な安全性・品質保証をどう適用するかという課題に対応するものであり、現場では極めて実践的なテーマである。開発者は「古い＝使えない」ではなく、「古い＝再評価すべき資産」として捉え、安全性を維持した上での活用を検討すべきである。	けたいソフトウェア」を安全に管理するための考え方を示している。レガシーソフトウェアを「使ってはいけない」とは言っていない。むしろ、「使うなら責任を持って管理しなさい」としている。開発等に関わる記録がないからといって放置するのではなく、今のルールに合わせて評価し直し、安全が確保されているかをしっかり確認することが求められる。過去の資産を、現在と未来に安全につなげていくこと。これは、技術の継続と安全性のバランスをとるための現実的な考え方である。すべてをゼロから作り直すのは現実的ではないこともある。だからこそ、安全を損なわずに「うまく引き継ぐ」ための知恵として、この条項が存在する。それが、レガシーソフトウェアと向き合ううえで、もっとも大切な姿勢である。	

○　IEC 62304 の箇条５の概説

箇条	細分箇条	概　要　解　説	簡　易　解　説	リファレンス・ポイント
箇条 5 「ソフトウェア開発プロセス」		（１）序論 IEC 62304 における箇条 5「ソフトウェア開発プロセス」は、医療機器ソフトウェア開発において実施すべき一連のプロセスを体系的に定めた中核的な規定である。このプロセスは、医療機器ソフトウェアが設計意図通りに、安全かつ効果的に機能することを保証するための枠組みとして位置付けられている。ソフトウェアの企画段階からリリースに至るまでの工程を明確に定義し、それぞれの工程において実施すべき活動や作成すべき成果物を規定している。	●　はじめに 箇条 5「ソフトウェア開発プロセス」は、医療機器のソフトウェアを安全に、正しく、効率よく作るための手順を決めたものである。これは「ただ動けばよい」ではなく、人の命に関わるようなソフトウェアだからこそ、ミスが起きないように開発のやり方をルールとして整えておくことが目的である。 この章では、ソフトウェアを開発するときにどんな順番で、何を考え、何を確認して、どのように完成させるかを、8 つのステップに分けて説明している。	1）GB（※）: P84-P147 （※）GB : IEC 62304 実践ガイドブック（じほう）
		（２）ソフトウェアライフサイクルにおける役割 IEC 62304 では、ソフトウェア開発を「ライフサイクル」という観点から捉えている。すなわち、ソフトウェアは開発されて終わりではなく、使用中の保守、バージョンアップ、変更管理を含む長期的な運用が想定されており、これらを見越した計画的な設計・実装が求められる。その出発点となるのが、本箇条で定められた開発プロセスである。このプロセスには、要求定義、設計、実装、テスト、リリースといった一般的なソフトウェア開発工程が含まれているが、医療機器ソフトウェアにおいて特有の要件、すなわち医療機器として必要とされるリスクマネジメント、トレーサビリティや検証の徹底等の医療機器としての安全性への配慮について、ライフサイクルを通じて求めている。		1）IEC 62304 : 附属書 C、C.6 において、ISO/IEC 12207「ソフトウェアライフサイクルプロセス」（一般的なソフトウェアのライフサイクルプロセス規格）との視点の違い、各箇条の対比が示されている。
		（３）開発プロセスの 8 つの構成要素 箇条 5 では、開発プロセスを以下の 8 つの細分箇条に分け、それぞれの工程における活動と要求事項を明確にしている。 5.1 ソフトウェア開発計画 ソフトウェア開発の全体計画を策定し、体制・スケジュール・レビュー体制・安全クラスへの対応方針を明確化する。 5.2 ソフトウェア要求事項分析 医療機器として必要な機能や制約条件をソフトウェア要求事項として整理・文書化する。 5.3 ソフトウェアアーキテクチャの設計 システム構成、モジュールの分割、安全関連機能の独立性等の観点から全	●　ソフトウェア開発の 8 つのステップ IEC 62304 の箇条 5 では、次のような 8 つの細分箇条（ステップ）に分かれている。 5.1 ソフトウェア開発計画 まずは「どうやって開発を進めるか」の全体的な予定を立てる。 5.2 ソフトウェア要求事項分析 ソフトに「何をさせたいか」「どんな機能が必要か」をはっきりさせる。 5.3 ソフトウェアアーキテクチャの設計 ソフトの全体構造を設計する。「どこに何の機能を持たせるか」を考える。 5.4 ソフトウェア詳細設計	1）「開発プロセス」に係るセキュリティについては、IEC 81005-1 箇条 5、細分箇条 5.1-5.8 として、IEC 62304 箇条 5、細分箇条 5.1-5.8 に対応して追加する要求事項の考え方について示されている。

		体の構造を設計する。 5.4 ソフトウェア詳細設計 　各モジュールやユニットがどのように動作するかを具体的に記述し、後の実装・試験に備える。 5.5 ソフトウェアユニットの実装 　詳細設計に基づき、実際のソースコードを記述し、静的解析や単体試験を通じて品質を確認する。 5.6 ソフトウェアの結合及び結合試験 　複数のユニットを統合し、インターフェースや相互作用の動作確認を行う。 5.7 ソフトウェアシステム試験 　ソフトウェアが仕様通りに動作するか、統合された状態での機能試験を実施する。 5.8 システムレベルで使用するためのソフトウェアリリース 　テスト結果やリスクマネジメントの結果を踏まえ、最終的に医療機器として使用できる状態でリリースする。	モジュールごとの細かい動作内容を詳しく決める。 5.5 ソフトウェアユニットの実装 　実際にプログラム（コード）を書く。 5.6 ソフトウェアの結合および結合試験 　書いた部品（モジュール）をつなげて、正しく連携するかテストする。 5.7 ソフトウェアシステム試験 　ソフト全体が、最初に決めた通りに動いているかをまとめて確認する。 5.8 システムレベルで使用するためのソフトウェアリリース 　最終的に「これで安全に使える」と判断し、正式にリリースする。 ●　なぜ順番と手順が大事なのか？ 　ソフトウェア開発では、最初の設計があいまいだと、後で間違いに気づいても直すのが大変になる。とくに医療機器では「あとでバグを見つけた」では遅いので、最初からしっかりとした順序で確認しながら進めることが重要である。また、1つのミスが人の命に関わるような医療機器では、「テストした結果を記録に残す」「設計と実装のズレをなくす」等、見えない部分のチェックや記録も開発の一部とされる。	
		（４）安全クラスによる要求事項の違い 　この開発プロセスは、前述の箇条 4.3 で定義されたソフトウェアの安全クラス（A, B, C）によって、要求される活動の厳格さや文書化の度合いが異なる。例えば、クラス C（重大な危害の可能性があるソフトウェア）においては、各工程でのレビューや検証が厳密に求められる。一方で、クラス A では要求事項の一部が省略可能とされている。このように、安全性に応じて適用の度合いが変化することは、過剰な負担を回避しつつ、リスクに見合った対策を講じる合理的な構成となっている。		1）　ソフトウェア安全クラスへの要求事項は IEC 62304 の各箇条の本文内に記載されているが、その概要として、IEC 62304：附属書 A、A.2「ソフトウェア安全クラス別要求事項のまとめ」、表 A.1 を参照して、各クラスに要求される IEC 62304 の各箇条、細分箇条に係る要求事項を整理、確認しておく。
		（５）開発プロセスと他プロセスとの連携 　この箇条 5 のプロセスは、箇条 4 で規定された品質マネジメントシステムおよびリスクマネジメント活動と密接に連携する。また、箇条 6（保守プロセス）、箇条 7（リスクマネジメントとのインターフェース）、箇条 8（構成管理）、箇条 9（問題解決）とも有機的に結びついており、ライフサイクル全体で一貫性のある開発が可能となる。	●　ソフトウェア開発は「チーム戦」 　これらの作業は、1 人のプログラマーだけでやるものではない。設計者、品質管理者、テスト担当者、プロジェクトマネージャー等、いろいろな役割の人が協力して進める。例えば、 ・　計画を立てるのはマネージャー ・　設計を考えるのは設計者 ・　コードを書くのはプログラマー	1）　IEC 62304、図 1,図 2 参照 2）　GB（※）：P62-P63、図 4-2 参照 （※）GB：IEC 62304 実践ガイドブック（じほう）

			・　テストを実行するのは試験担当 というように、それぞれの担当が連携して、安全で確かなソフトウェアを作る。だからこそ、共通のルール（＝IEC 62304）が必要になる。	
		（６）結論 IEC 62304 の箇条 5 は、医療機器ソフトウェアの安全で信頼性の高い開発を実現するための技術的・組織的な基盤を提供するものである。このプロセスを正しく実施することで、ソフトウェアの設計ミスや仕様誤解によるリスクを未然に防ぎ、最終的な医療機器の安全性と有効性を確保することが可能となる。	●　まとめ 箇条 5「ソフトウェア開発プロセス」は、医療機器のソフトウェアを安全に、確実に、効率よく作るための 8 つのステップを示している。これは単にコードを書く作業ではなく、設計からテスト、記録、リリースまでを一つの流れとしてしっかり管理することを意味している。例えば、演劇をやるときに、脚本・配役・稽古・衣装・照明・リハーサル・本番という順番があるように、ソフトウェア開発にもちゃんとした流れがあり、それを守ることで「安全で使えるソフトウェア」ができあがる。	
	5.1 ソフトウェア開発計画	（１）序論 IEC 62304 における細分箇条 5.1「ソフトウェア開発計画」は、ソフトウェア開発に先立って、開発の全体像を明確にし、関係者間で共通の理解を得るために必要な計画書を作成することを求めている。これは、医療機器ソフトウェアの開発という複雑で責任の重い作業において、無駄や手戻り、誤解を防ぐための「地図」のようなものである。ソフトウェア開発計画は単なる日程表ではない。それは、安全性、品質、リスクマネジメント、レビュー、テストといった多面的な観点から、開発全体を見通すための構造化された文書であり、プロジェクトの開始から終了までを統制する「統合的管理の骨格」である。	●　はじめに 細分箇条 5.1「ソフトウェア開発計画」は、医療機器のソフトウェアを開発するにあたって、どのように作業を進めていくのかをあらかじめ計画としてまとめておくことを求める規定である。例えるなら、「遠足のしおり」のようなものである。行き先（目的）、集合時間（スケジュール）、持ち物（リソース）、係（役割分担）、危険時の対応（リスクマネジメント）等を前もって考えておけば、当日あわてずに行動できる。それと同じように、ソフトウェア開発でも「計画」がなければ、途中で迷ったり、事故が起きたりしてしまう。	1）GB（※）：P86-P102 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）ソフトウェア開発計画の目的 ソフトウェア開発計画の目的は以下のようになる。 ・　プロジェクト全体の方向性を明確にすること ・　開発工程ごとの責任と役割分担を可視化すること ・　開発における活動と成果物の順序と依存関係を整理すること ・　安全性に関わる判断（例：安全クラス）を文書化すること ・　リスクマネジメントや構成管理等、他プロセスとの連携計画を明確にすること これらにより、誰が、何を、いつ、どのように行うのかを明文化し、チーム内の連携と品質管理を確実なものとする。	●　なぜ「開発計画」が必要なのか？ ソフトウェアの開発は、多くの工程と人が関わる複雑なチーム作業である。とくに医療機器のソフトウェアは、人の命や健康に関わるため、次のような理由から開発計画が重要とされている。 ・　どんな手順で進めるのかをはっきりさせるため あいまいなまま始めると、やり直しやミスが増える。 ・　誰が何をするか決めておくため チーム内で役割が分かっていないと、抜けや重複が起きやすい。 ・　問題が起きたときにすぐ対応できるようにするため スケジュールに余裕があるか、代替案があるかを計画段階で確認する。	1）IEC 62304:附属書 B、B5.1「ソフトウェア開発計画」

			・　規制や安全性のルールを守るため 医療機器では法律や規格に従う必要があるため、計画にそれらを反映することが重要である。	
		（３）開発計画に含めるべき要素 ソフトウェア開発計画に以下のような内容を含めることを求めている。 ① ソフトウェア安全クラスの定義 最初に、対象となるソフトウェアまたはその構成要素（ソフトウェアシステムやアイテム等も含めて）が、どの「安全クラス（A, B, C)」に該当するかを明確にする必要がある。これは、以降のすべての工程（要求、設計、試験等）で適用すべき要求レベルを定める基準となる。また、ソフトウェア全体に一律のクラスを適用するのではなく、構成モジュールごとに異なるクラスを設定することも可能であり、その判断根拠を明記することが求められる。 ② 開発活動のスケジュールとマイルストーン 開発の各フェーズ（要求分析、設計、実装、検証等）における開始日・完了日、主要な成果物（ドキュメントやテスト結果）を時系列で整理し、マイルストーンごとにチェックポイントを設けることが望ましい。 このスケジュールは、後のレビューや進捗確認における基準としても使用される。 ③ 開発チームと責任分担 各工程において誰が何を担当するか、役割分担を明確にすることも重要である。例えば、以下のような記載が必要となる。 ・　要求仕様書の作成者とレビュー責任者 ・　テスト計画の作成者と承認者 ・　リスク分析の責任者と関連部署 また、責任の独立性の観点から、設計と検証を同一人物が行うことを避けること等も考慮する。 ④ 使用する標準・手順・ツール 開発で使用する業界標準（例：IEC 62304、ISO 14971、ISO 13485）や、社内の開発ガイドライン、レビュー手順、テストツール、構成管理ツールの一覧も計画書に明記すべきである。これにより、開発プロセスの一貫性と再現性が担保される。	● 開発計画に含めるべきこと ソフトウェア開発計画には以下のような情報を含める必要がある。 ① 開発活動の範囲（何を開発するのか） ・　ソフトウェアの目的や機能 ・　対象となる製品やバージョン ・　開発する部分、再利用する部分、除外する部分の明記 ② 適用する開発プロセスと規格 ・　IEC 62304 をはじめとする開発ルール ・　使用する QMS（品質マネジメントシステム） ・　ソフトウェア安全クラス（A/B/C）とその根拠 ③ 作業の手順と順番（プロセスの流れ） ・　要求分析、設計、実装、テスト、リリースまでの各ステップ ・　それぞれのステップで必要な作業や記録 ④ 作業を行う人とその役割 ・　プロジェクトマネージャー、開発者、テスト担当、品質保証等 ・　外部委託先が関与する場合は、その管理方法 ⑤ スケジュールとマイルストーン ・　いつまでに、何を終える予定か ・　各フェーズの終わりにおける会議のタイミング（チーム全体レビュー等） ⑥ 使用するツールや設備 ・　開発用 PC、ソフトウェア開発環境、テストツール等 ・　バージョン管理や文書管理システムも含まれる ⑦ リスク対策の方針 ・　リスクマネジメントとどう連携するか ・　問題が起きたときの連絡体制と緊急対応ルール ● 開発計画は「1 回で終わり」ではない 開発計画は、一度立てたら終わりではない。プロジェクトが進むにつれて、	1） ソフトウェア安全クラスへの要求事項は IEC 62304 の各箇条の本文内に記載されているが、その概要として、IEC 62304：附属書 A、A.2「ソフトウェア安全クラス別要求事項のまとめ」、表 A.1 を参照して、各クラスに要求される IEC 62304 の各箇条、細分箇条に係る要求事項を整理、確認しておく。

		⑤ 他プロセスとの連携計画 IEC 62304 では、開発プロセスと以下のプロセス群の相互連携が重要視されている。 ・ リスクマネジメントプロセス（箇条 4.2、7） ・ 構成管理プロセス（箇条 8） ・ 問題解決プロセス（箇条 9） それぞれのプロセスが開発のどのタイミングで介入し、どのように成果物を共有するかを整理しておく必要がある。 ⑥ レビュー・確認・検証のタイミング 各成果物（要求仕様、設計文書、テスト仕様等）に対するレビューや検証活動の実施予定と方法も明記する。レビューの記録を残すことで、規格や規制当局の監査に耐えうるエビデンスを準備することができる。 ⑦ 変更管理・中断対応 開発中に設計変更や予期せぬ中断が発生した場合の対応方針（例：変更管理手順、影響評価の実施条件）をあらかじめ盛り込むことも望ましい。	状況は変化する。例えば ・ 新しい要求が追加された ・ スケジュールが遅れた ・ 問題が発生して対応が必要になった このようなときには、開発計画を見直し、修正し、最新の状態に保つこと（更新）が求められる。これを「生きた計画書」と言い、IEC 62304 ではこのような柔軟な対応も重視している。 ● 開発計画を作るとは？ 開発計画は、医療機器ソフトウェアの開発を安全に、効率的に、ルールに従って進めるための出発点である。どんな良いチームでも、計画がなければバラバラになり、ミスや混乱が起きやすくなる。だからこそ、IEC 62304 では最初に「しっかりとした開発計画を立てましょう」と強く求めている。「演劇」をすることを考えてみよう。いきなり「じゃあ明日から練習ね！」ではうまくいかない。 ・ いつどこで練習する？ ・ 台本は誰が書く？ ・ 役者と裏方はどう分担する？ ・ 衣装や道具は誰が用意する？ ・ 何日までにどこまで終わらせる？ こういったことを最初に考えて、紙にまとめる（＝計画書を作る）ことで、スムーズに、トラブルなく本番を迎えることができる。それと同じように、ソフトウェア開発でも「計画」がとても大事である。	

<table>
<tr>
<td></td>
<td></td>
<td>（４）文書化とトレーサビリティ
　ソフトウェア開発計画書は、計画段階で一度作って終わりではなく、開発中も更新・維持されるべき「生きた文書」である。実際の作業と乖離しないよう、変更時には版管理し、関係者に通知・承認を得ることが求められる。また、開発計画と実際の成果物（テスト記録、設計書等）との対応が取れているかどうか、トレーサビリティ（追跡可能性）の確保も重要である。これは、品質保証だけでなく、製品回収や修正時の迅速な対応にも直結する。</td>
<td>● 開発計画と他の文書との関係
　ソフトウェア開発計画は、プロジェクト全体の「上位文書」として、以下のような様々等キュメントと関係を持つ。
<table><tr><th>関連文書</th><th>内容</th></tr><tr><td>要求仕様書</td><td>ソフトウェアに求められる機能や性能をまとめたもの</td></tr><tr><td>設計文書</td><td>ソフトウェアの構造や動作を詳細に説明したもの</td></tr><tr><td>テスト計画書</td><td>どんな試験をどのタイミングで行うかを示す</td></tr><tr><td>リスクマネジメント計画</td><td>危険への対応方針を定めたもの</td></tr></table>　これらの文書は、開発計画の中で「いつ」「誰が」「どう作成・確認するか」が決められているため、全体の管理がしやすくなる。</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>（５）ソフトウェア開発計画と規制対応
　計画文書は、規制当局への申請や審査において、審査員が開発の妥当性を評価するための最初の資料となる。したがって、曖昧な表現や口頭の運用に頼らず、明確な記述と責任体制が示されていることが必要である。規制当局からの典型的な指摘例としては、「開発計画に基づくエビデンスが不十分」「安全クラスの根拠が不明」「レビューが文書化されていない」等が挙げられる。</td>
<td></td>
<td>1)　「医療機器の基本要件基準第 12 条第 2 項の適用について」（平成 29 年 5 月 17 日付け薬生機審発 0517 第 1 号通知）別添参照。
2)　「医療機器に係る基本要件適合性チェックリストについて」（令和 3 年 8 月 18 日付け薬生機審発 0517 第 1 号通知）</td>
</tr>
<tr>
<td></td>
<td>5.2 ソフトウェア要求事項分析</td>
<td>（１）序論
　「ソフトウェア要求事項分析」は、ソフトウェアが「何をすべきか」を明確に定める工程である。細分箇条 5.2 では、ソフトウェアが安全かつ意図した通りに動作するために、必要な機能や制約を正しく洗い出し、要求事項として明文化することを求めている。この要求分析の工程は、ソフトウェア開発プロセスの全体的な成否を左右する極めて重要な段階である。なぜなら、誤った要求定義に基づいて設計・実装が行われた場合、後工程でいくら努力しても意図と異なる製品ができあがってしまうからである。IEC 62304 では、医療機器としての使用を前提に、一般的な機能要件だけでなく、安全性・信頼性・規制への適合といった特有の観点も含めて要求分析を行うことを重視している。</td>
<td>● はじめに
　細分箇条 5.2「ソフトウェア要求事項分析」は、医療機器ソフトウェアを作るときに、「このソフトに何をさせるのか？」「どんな動きをする必要があるのか？」を明確にする作業を指している。例えるなら、家を建てるときの「設計図の前に、お客さんがどんな家に住みたいかを聞く作業」にあたる。間取り、部屋数、コンセントの場所等、要望を聞かずに作ってしまったら、住みにくい家になってしまう。それと同じで、ソフトウェアも「使う人が必要とする機能」や「守るべきルール」をあらかじめしっかり理解しておくことが、良いソフトづくりの第一歩である。</td>
<td>1)　GB（※）：P103-P113

（※）GB：IEC 62304 実践ガイドブック（じほう）</td>
</tr>
<tr>
<td></td>
<td></td>
<td>（２）ソフトウェア要求事項について
　ソフトウェア要求事項には以下のようなものがある。</td>
<td>● 要求事項って何？
　ここでいう「要求事項」とは、ソフトウェアに求められることすべてを指す。大きく分けて以下の 2 つがある。</td>
<td>1)　IEC 62304：付属書 B、B5.2「ソフトウェア要求事項分析」</td>
</tr>
</table>

		① 機能的要求 ソフトウェアがどのような処理や動作をすべきかを示す要求である。例えば、 ・ 「心拍数が 100 を超えたらアラームを鳴らす」 ・ 「投薬量の設定値を記録する」 といった具体的な機能の指示が該当する。 ② 性能要求 処理速度、応答時間、同時処理能力、精度等、性能に関する条件である。 ・ 「測定値の表示はセンサー取得後 1 秒以内に行う」 ・ 「誤差は±0.2℃以内とする」 等がこれに該当する。 ③ 制約条件 設計上の制限や技術的・法的制約を含む。例えば、 ・ 特定の OS 環境上で動作させる ・ 医療機器規制によりログを一定期間保存する 等の外部的要因である。 ④ 安全性に関わる要求 医療機器としての不具合による危害を防ぐための仕様である。 ・ データ喪失時のリカバリ動作 ・ 使用者への誤操作防止設計 等、リスク低減を目的とした要求がここに含まれる。 ⑤ ユーザーインターフェース要件 表示や操作性に関する要件であり、誤使用を避けるための視認性、操作の一貫性等が含まれる。	・ 機能的要求：何をするのか？（例：心拍数を表示する、温度が高いときに警報を出す） ・ 非機能的要求：どうあるべきか？（例：1 秒以内に反応する、10 時間連続で動作できる） この「要求事項」をきちんと分析することが、「何を作るか」をはっきりさせる作業であり、これが要求事項分析となる。 ● なぜ要求事項分析が大事なのか？ ソフトウェアを作る前に、何を作ればいいのかがはっきりしていなければ、途中で迷ったり、完成しても使い物にならなかったりすることがある。とくに医療機器では、間違った動作が人の命に関わるので、以下の理由から要求分析はとても大事である。 ・ 必要な機能の抜けや漏れを防ぐ ・ 間違った機能を実装しないようにする ・ 後から変更になることを減らす ・ 設計やテストの土台となる情報を作る つまり、要求事項分析は「地盤固め」のような作業で、ここがしっかりしていないと、その上に作るソフトもぐらついてしまう。	
		（３）要求事項分析の基本的な手順	● IEC 62304 で求められる作業内容	1） IEC 62304 細分箇条 5.2.1-5.2.6

次のような手順に従って要求事項分析を行うことが求められる。

① ソースの特定
要求事項は様々な出所から発生するため、まずその情報源を明確にする必要がある。主な情報源としては、

・ 顧客やユーザーのニーズ
・ 法令・規格（例：ISO 14971、IEC 60601）
・ 他のシステム要件（ハードウェアやクラウド連携）
・ 過去の問題やバグ履歴

等が挙げられる。

② 要求事項の収集と文書化
得られた要求事項を一覧化し、漏れや重複を防ぐ。あいまいな表現は明確に言い換え、「何を」「どのように」「どの程度まで」行うかを具体的に記載することが重要である。

③ 優先順位付けと分類
要求事項ごとに優先順位を決め、安全性や規制への影響度に応じて分類して管理する。

④ 安全性に関わる要求事項の識別
IEC 62304 では、要求事項のうち安全性に関連するもの（リスク制御策に対応するもの）を明確に区別してラベルをつけることが求められている。

⑤ レビューと合意形成
要求事項の定義は開発者だけでなく、品質保証担当者や安全性評価者と共にレビューを行い、関係者全員の合意を得ることが重要である。

IEC 62304 では、ソフトウェア要求事項分析において以下のような活動を行うことが求められている。

① 要求事項の収集
・ 製品の使い方（使用者、使用場所）を考える
・ 関係者（医師、看護師、技術者等）からヒアリングする
・ 使う人が間違えないように、どう動作すべきかを考える

② 要求事項の文書化
・ 口頭やメモではなく、正式な文書としてまとめる
・ 各項目に番号や ID をつけ、後で追跡できるようにする

③ 要求事項の分類と優先順位づけ
・ 安全に関わるもの、便利さに関わるもの、法律に関わるものを分ける
・ どれが重要で、どれが「あるとよい」かを整理する

④ リスクマネジメントとの連携
・ 要求事項の中で安全性に関わるもの（例えばアラーム機能等）については、リスクマネジメント（IEC 62304 の 7 章）と結びつける
・ 「この要求がないと、何が危険になるか？」を考える

⑤ テストの準備
・ この要求事項が、本当に実現されているかどうかを確認できる方法（テスト可能性）を意識して書く
・ あいまいな言葉（「すぐに」「できるだけ早く」等）は使わず、数値や条件で表す

● 具体的な要求事項の例

ID	要求事項（例）
REQ-001	心拍数を測定後 1 秒以内で表示すること
REQ-002	心拍数が 180 を超えたとき、10 秒以内に警報を鳴らすこと
REQ-003	測定データは 7 日間保存できること
REQ-004	操作画面は英語・日本語の切り替えができること
REQ-005	測定ミスが起きた場合、再測定を促すメッセージを表示すること

			このように、誰が読んでも同じ意味に理解できるように「明確な言葉」で書かれることが重要である。	
		（４）トレーサビリティの確保 IEC 62304 では、ソフトウェア要求事項が後の設計、実装、テストにどのように反映されたかを追跡可能にする「トレーサビリティ」が強く求められている。これは、以下の目的を果たすためである。 ・ 要求事項に対するテストが実施されていることの証明 ・ 問題発生時に原因を迅速に特定するための手がかり ・ 規制当局の審査において、安全性が担保されていることの根拠 具体的には、各要求事項に対して一意の ID を割り当て、設計仕様書やテスト仕様書にその ID を紐付けて記録する。		1） IEC 62304：付属書 B、B5.2
		（５）要求事項の変更管理 ソフトウェア開発の過程で、新しい要求事項が追加されたり、既存の要求事項が変更されたりすることは珍しくない。IEC 62304 では、こうした変更に対して以下のような管理が求められる。 ・ 変更の理由と内容の記録 ・ 影響を受ける設計・テストの再評価 ・ リスクへの影響分析 ・ 変更後の再レビューと承認 変更管理が適切に行われないと、設計との齟齬が発生し、意図しない動作やリスクの増大を招くことになる。		
		（６）結論 ソフトウェア要求事項分析は、医療機器ソフトウェアの品質と安全性を確保するうえで最も基盤となる活動である。IEC 62304 におけるこの工程は、単なる仕様記述ではなく、「安全で確実な医療行為を支える土台」を構築する行為に等しい。明確かつ網羅的な要求事項の定義と、要求事項に基づいた設計・実装・検証の一貫した流れが、最終的に信頼性の高い製品へとつながる。	● まとめ 細分箇条 5.2「ソフトウェア要求事項分析」は、ソフトウェアに何をさせるかを明確にする「設計前の最重要ステップ」である。ここでの抜けや曖昧さは、後工程のすべてに悪影響を及ぼすため、丁寧で体系的な作業が必要とされる。しっかりとした要求事項分析ができていれば、その後の設計・開発・テストのすべてがスムーズになり、結果として安全で信頼できるソフトウェアが完成する。だからこそ、IEC 62304 ではこのステップを強く重視している。例えば、「屋台を出す」と決まったとき、いきなり調理を始めることはない。 ・ 何を売るか（焼きそば？クレープ？）	

			・　材料は何が必要か？ ・　販売価格は？ ・　何人でまわすのか？ 　こうした「やるべきこと」を最初に考え、一覧にしておくのが「要求分析」にあたる。これがないと、準備不足で当日うまくいかない。ソフトウェア開発でもそれは同じである。	
	5.3 ソフトウェアアーキテクチャの設計	（１）序論 　「ソフトウェアアーキテクチャの設計」とは、細分箇条 5.2 で定めたソフトウェア要求事項をもとに、ソフトウェア全体の骨組みや構造を定めることである。細分箇条 5.3 では、医療機器ソフトウェアにおいて、各機能がどのように分割され、どのように相互に連携し、どの部分が安全性に影響を及ぼすのかを体系的に示すアーキテクチャ設計が必要である。アーキテクチャ設計は、家を建てる際の「設計図」に相当する。設計図がなければ、どこに壁を作り、どこに扉を設置するかが曖昧になり、住居としての機能や安全性が確保できないのと同様に、アーキテクチャが不十分なソフトウェアは、保守性や安全性、信頼性に問題を抱えることとなる。	●　はじめに 　細分箇条 5.3「ソフトウェアアーキテクチャの設計」は、ソフトウェア全体の構造や、部品どうしのつながり方、働きの分担を大きな視点で考える作業である。例えば、家を建てるときに「キッチンはここ」「お風呂はここ」「この部屋は子ども用」といった間取り図を考えるように、ソフトウェアでも「どの機能をどこに置くか」「どうやって情報をやり取りするか」といった全体の形（＝アーキテクチャ）を最初に設計する。この段階では、まだコードを書いたりはしないが、この設計が良くできていれば、あとからの作業がずっとスムーズに進み、安全でメンテナンスしやすいソフトができる。	1） GB（※）: P114-P120 （※）GB : IEC 62304 実践ガイドブック（じほう）
		（２）ソフトウェアアーキテクチャ設計の目的 　IEC 62304 がソフトウェアアーキテクチャの設計を明文化して求めている理由は、以下の 3 点に集約される。 ①　安全関連機能の特定と隔離 　ソフトウェアの中で、患者に危害を及ぼす可能性のある機能を明確にし、それを他の機能から独立させることで、不具合が伝播しないようにする。 ②　機能ごとの責務と分離の明確化 　モジュールやサブシステムごとに役割を明確に分け、責務を限定することで、テストや修正の影響範囲を狭め、保守性を向上させる。 ③　トレーサビリティとリスクマネジメントのための基盤構築 　要求→設計→実装→テストの一貫したトレースを可能にし、安全性を説明可能な構造とする。	●　アーキテクチャとは何か？ 　「アーキテクチャ」とは、もともと「建築様式」や「構造」という意味の言葉である。ソフトウェアにおいては、ソフトを構成する部品（モジュール）がどう配置され、どうつながっているかを表す全体の設計図のようなものである。ソフトウェアアーキテクチャの設計では、次のようなことを考える。 ・　機能をいくつの部品に分けるか？ ・　どの部品がどの役割を持つか？ ・　どのような順番で動くか？ ・　情報（データ）はどこからどこへ流れるか？ ・　外部の機器（センサー等）とのやり取りはどうするか？ 　つまり、「動かす中身」ではなく、「動かす構造と仕組み」を考える段階である。 ●　なぜアーキテクチャの設計が重要なのか？ 　ソフトウェアアーキテクチャの設計は、次のような理由から非常に重要である。 ・　大きなミスを防げる	

			最初に全体を見ておけば、後から「構造が合わなかった」といったトラブルを防げる。 ・　役割分担がしやすい チーム開発では、部品ごとに作業を分けられる。 ・　安全性を考慮しやすい 危険な機能（アラーム等）を別のモジュールに分けて、確実に動くようにできる。 ・　テストしやすい構造になる モジュールごとにテストをして、全体が正しく動くことを確かめやすくなる。 ・　将来の変更に強くなる アップデートや修正があっても、一部を変えるだけで済む構造になる。 このように、アーキテクチャはソフトの「骨組み」であり、この設計がしっかりしていないと、安全性も信頼性も損なわれることになる。	
		（３）アーキテクチャ設計に含まれるべきソフトウェアアイテム（ソフトウェアシステムの構成要素） IEC 62304 では、ソフトウェアアーキテクチャ設計で明確にすべき情報として、以下のような項目が挙げられている。 ① ソフトウェアアイテムの分割 ソフトウェア全体を、複数の機能単位やモジュールに分解し、それぞれに固有の機能と責務を持たせる。例えば、 ・ ユーザーインターフェース ・ センサーデータ取得 ・ アラーム制御 ・ データ記録 ・ 設定管理 等のように、論理的に独立した要素へと分類する。 ② ソフトウェアアイテム間の関係と通信手段 各アイテムがどのように連携するかを明示する。例えば、データが「関数	● IEC 62304 が求めていること IEC 62304 では、アーキテクチャ設計において以下のようなことを明確にするよう求めている。 ① ソフトウェアモジュールの定義 ・ どんなモジュール（部品）があるか ・ それぞれのモジュールがどんな機能を持つか ② モジュール間のインターフェース（接続） ・ どのモジュールがどこに情報を渡すか ・ 接続はどんな形式か（例：ファイル、関数呼び出し、データベース等） ③ 外部とのやりとり ・ 機器本体、センサー、ネットワーク等、ソフトの外側とどうつながっているか ④ リスクコントロールに関わる機能の位置づけ ・ アラームや安全装置等、リスクを下げるための機能がどこにあるか ・ 他の機能から切り離されているか、安全性が確保されているか	1) ソフトウェアアイテムの分離粒度（どの程度まで細分化してアイテムとするか）について、IEC 62304 は規定しておらず、分離の考え方を示している。分離粒度は、医療機器ソフトウェアを設計開発（企業）側が検討することである。どの程度までソフトウェアアイテムを分離するかは、固有の医療機器ソフトウェアの違い、設計開発や上市後の維持管理における企業の QMS の違いによってもそれぞれのあり方がある。

		呼び出し」「メッセージキュー」「共有メモリ」等でやり取りされるのか、それぞれの接点と通信手段を図示する必要がある。 ③ 安全性関連構成要素の識別 ソフトウェアの中で安全性に直接関与するモジュール（例えば、誤投薬を防ぐ投薬制御ロジック等）を明示し、他のアイテムとは隔離または冗長化された設計とすることが望ましい。 ④ 外部インターフェースとの接続 ハードウェアや他のソフトウェアシステムとの接続点も重要な設計情報である。センサーやアクチュエーター、クラウドサービス、ユーザー操作パネルとの接点を明記し、エラー伝播の可能性やデータ整合性への配慮を行う。 ⑤ アーキテクチャ設計の正当性の記録 設計した構成とその構造が、なぜ安全性を確保できるのか、その判断根拠を説明する文書（アーキテクチャ設計説明書等）を残す。	⑤ トレーサビリティの確保 ・ この設計が、前の段階（要求分析）とどうつながっているか ・ あとで誰が見ても追跡できるようにする	
		（４）安全性に係るアイテムの分離の考え方 IEC 62304 では、安全性に影響を及ぼすアイテム（例えば投薬制御）と、影響を及ぼさないアイテム（例えば画面色の設定等）を、設計段階で分離することが求められている。このような考え方により、ある部分に不具合があっても、その影響が安全性に関係する機能に波及するのを防ぐことができる。これには、以下のような設計方針がある。 ・ 安全性関連アイテムは独立したプロセスとして実行 ・ 異常検出ロジックを別アイテムで構築し、二重化 ・ ソフトウェアウォッチドッグでハングアップを検知		1) IEC 62304：附属文書 B、B4.3 を参考に、安全クラスを考慮して、システム全体の安全性の確保を検討することを検討。図 B.1 のように、上位のソフトウェアアイテムの安全性の確保によって、それに接続された下位のアイテムの安全性クラスを低減化し、過度な安全性ではなく、開発コストや作業量も含めた効果的なアイテムの分離を検討することも必要。
		（５）SOUP の特定とシステムへの要求 IEC 62304 では、SOUP（Software Of Unknown Provenance）と呼ばれる、開発元や品質保証が明確でないソフトウェアアイテムを使用する際の注意事項が定められている。SOUP の機能や使用条件、既知の制限事項や問題点を明らかにし、必要に応じて安全性を確保するための追加のソフトウェア要求事項を定義する必要があるとされている。これは、SOUP が予期しない動作を起こす可能性があるため、その影響を最小限に抑えるための対策である。また、SOUP が安全性に関わる部分に使用されている場合、既知の不具合や問題の有無を調査し、それらの情報を文書化することが要求されている。加えて、それらの問題が医療機器の安全性にどのような影響を与えるかを評価しなければならない。これにより、未知のリスクによって患者や使用者に危害	● SOUP とはなにか？ IEC 62304 では、「SOUP」という特別なソフトウェアの使い方について説明している。「SOUP」とは、もともと医療機器用に作られたわけではなく、どのように開発されたか明確になっていないソフトウェアで、安全性についての情報がはっきりしていないソフトウェアのことである。例えば、インターネットで手に入れた無料のプログラムや、市販のソフト等がこれにあたる。SOUP を使うときには、そのソフトが何をするのか、どんな制限があるのか、どんな問題が知られているのかをきちんと調べておく必要があると決められている。そして、もしそのソフトに問題があった場合に備えて、必要なルールや条件（要求事項）を追加で決めておくことも大事である。	1) 迅速なシステム開発において、SOUP を使うなという意図ではなく、使用するうえでの SOUP の仕様の特定やリスクについて明確にすることを求めていることに注意。

		が及ぶことを防止することができる。つまり、SOUP を使用する際には、その信頼性や安全性に対する十分な理解と管理が不可欠である。		
		（６）アーキテクチャ設計の記述方法と図示 　アーキテクチャは、単なる文章ではなく、構成要素同士の関係性を視覚的に示す方法を用いることが望ましい。代表的な表記方法としては、 ・　ブロック図（Block Diagram） ・　クラス図（Class Diagram） ・　コンポーネント図（Component Diagram） ・　シーケンス図（Sequence Diagram） 等がある。特に IEC 62304 では、第三者が簡単に理解できることが重視されている。これは、医療機器ソフトウェアは、安全性に直結するためであり、設計やシステムの理解が不十分な場合、重大なリスクを引き起こす可能性があり、規制当局への提出資としても明瞭で理解しやすい図示が望まれる。	●　図を使って表現することも大事 　アーキテクチャの設計では、文章だけでなく、図（ブロック図やフローチャート）を使ってわかりやすく表現することが多い。	
		（７）テストとの関係 　アーキテクチャ設計は、以降の試験計画（ユニットテスト、結合テスト、システムテスト）の基礎となる。ソフトウェアアイテムごとの責任が明確であることにより、 ・　単体テストの対象が特定できる ・　モジュール間インターフェースのテストが可能になる ・　結合テストで重点的に確認すべき経路がわかる といったメリットが得られる。特に安全クラスが B または C のソフトウェアでは、この段階でテスト可能性まで考慮して設計されていることが望ましい。		
		（８）アーキテクチャ設計レビューの実施 　設計が完了したら、関係者による設計レビューを行う必要がある。レビューでは次の観点からの確認が行われる。 ・　ソフトウェア要求をすべて満たしているか ・　構成要素の分離が実現されているか ・　モジュールの依存関係に循環や過度な結合がないか ・　トレーサビリティが取れているか（要求←→構成要素）		

		レビューの結果は記録として残し、設計の妥当性を文書で証明できるようにする。		
		（９）結論 細分箇条 5.3「ソフトウェアアーキテクチャの設計」は、ソフトウェア全体の構造を安全性、保守性、テスト容易性の観点から最適化するための極めて重要なプロセスである。特に医療機器においては、予測不能な障害や誤動作が致命的な結果を引き起こす可能性があるため、ソフトウェアアイテムの明確な分割と安全性機能の隔離が不可欠である。本工程は、「どこに何があり、どう連携し、どこが危険か」を見える化する作業であり、それを通じて設計全体の透明性と信頼性を確保することができる。アーキテクチャ設計は、開発の要であり、後続工程の品質もここで決まるといっても過言ではない。	● まとめ 細分箇条 5.3「ソフトウェアアーキテクチャの設計」は、ソフトウェアの「全体像」を考え、安全で正確に機能するように部品を分け、つなぎ方を定める重要なステップである。この設計がしっかりしていれば、プログラムを書く人もテストする人も迷わずに作業でき、結果として品質の高いソフトウェアが完成する。医療機器という人の命に関わる分野では、こうした「目に見えない設計」の品質こそが、製品の安全を守る基盤となる。例えば、「お化け屋敷」を作ることになったとする。いきなり道具を作り始めても、うまくいかない。まずは、「入口はここ」「1 つ目の部屋はこう」「次はこうやって進む」といった全体の構成（アーキテクチャ）を考える必要がある。 ・ どの係がどこを担当するか？ ・ おどかし役と案内役をどう分けるか？ ・ スピーカーやライトはどこに配置するか？ こういったことを事前に考えるからこそ、当日の運営がスムーズになる。それと同じで、ソフトウェア開発でも、全体の設計（アーキテクチャ）を最初にしっかり決めておくことが大切である。	
	5.4 ソフトウェア詳細設計	（１）序論 ソフトウェア詳細設計は、前工程である「アーキテクチャ設計」に基づいて、各ソフトウェアアイテム（モジュールやユニット）の中身を具体的に定義する工程である。簡単にいえば、「どのように作るか」を仕様レベルで記述することであり、ソースコードを書く直前のステップにあたる。IEC 62304 の細分箇条 5.4 では、この詳細設計を通じて、安全性と品質を確保したうえで、実装や試験が行えるような「明確で再現性のある設計書」を作成することを求めている。これは、設計ミスを減らし、機能漏れや誤動作を防ぐとともに、開発後のトレーサビリティや変更管理にも大きく貢献するものである。	● はじめに 細分箇条 5.4「ソフトウェア詳細設計」は、前の段階で作成したソフトウェアアーキテクチャ（全体の構成）に基づき、それぞれの部品（モジュール）がどう動くかを、より細かく具体的に設計する作業である。例えば、家の「間取り図」まではアーキテクチャ設計だったとすれば、この詳細設計では「この部屋には何の家具を置くか」「コンセントは何個必要か」といった中身の具体的な計画を立てるイメージである。プログラムを書く前の「一つひとつの機能の設計図」と言ってもよい。この設計がしっかりしていれば、次のステップである実装（コードを書く作業）がスムーズに進み、バグや間違いも減る。	1） GB（※）：P121-P124 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）詳細設計とは何をするのか ソフトウェア詳細設計で行うべき主な内容は以下の通りである。 ・ 各ユニット（モジュール）の機能仕様の定義 ・ アルゴリズムや処理手順の明文化 ・ 入出力データの定義と制限	● 詳細設計で行うこと IEC 62304 における詳細設計では、以下のような内容をそれぞれのソフトウェアユニット（小さな部品）ごとに明確にしていく。 ① 処理内容の記述 ・ どんな入力を受け取るか？	

		・　エラーハンドリングの記述 ・　内部変数・データ構造の設計 ・　インターフェース仕様（呼び出し元・呼び出し先）の明確化 ・　安全関連動作（フェイルセーフ等）の記載 ・　テスト容易性や保守性を考慮した分割設計 　これらを設計書として文書化し、誰が読んでも仕様通りに実装が可能な状態にすることが目標である。	・　どんな処理を行うか？ ・　どんな出力を返すか？ 　例えば、「測定値を受け取り、上限値を超えたらアラームを出す」といった処理を、順番や条件を明確に記述する。 ②　データの構造や使い方 ・　どんなデータを扱うか（数値、文字列等） ・　データの形式（例：int 型、float 型等） ・　データの保存場所（一時記憶か、ファイルか） ③　エラー処理や異常時の対応 ・　入力が間違っていた場合はどうするか？ ・　通信が失敗したときはどう対応するか？ ④　他の部品（モジュール）との関係 ・　どことつながっているか？ ・　どんなタイミングで呼び出されるか？ 　これらをすべて文書にして記録しておくことで、後の作業者やテスト担当者も同じ理解を持てるようにする。 ●　なぜ詳細設計が重要なのか？ 　ソフトウェアの開発は、たくさんの人が関わるチーム作業である。そして、医療機器ソフトウェアのように人の命に関わるものは、「なんとなく動けばよい」という作り方は許されない。詳細設計をしっかり行うことで、次のような利点がある。 ・　ミスが減る：処理内容をはっきり決めておくので、迷わずコードが書ける ・　統一感が出る：複数人が作業しても、ばらばらにならない ・　テストしやすくなる：設計と結果を比較して、合っているか確認できる ・　保守しやすくなる：あとで修正する人も、構造を理解しやすい 　つまり、「安全で、効率よく、ミスの少ないソフトを作るための橋渡し」がこの詳細設計である。	
		（３）詳細設計で取り扱う要素の具体例 ①　関数やサブルーチンの仕様定義（例）	●　書き方の例 　以下は、あるソフトウェアユニットの簡単な詳細設計の例である。	

・ 関数名：calculateDosage()
・ 引数：patient_weight（float：浮動小数点型), drug_concentration（float：浮動小数点型）
・ 戻り値：float（投与量）
・ 処理概要：体重と薬濃度から推奨投与量を計算し返す。
・ 異常時処理：異常な値（マイナス値等）入力時は、マイナスの値を返す。

② 状態遷移図
医療機器によっては、複雑な動作モード（待機、計測、警報中、異常停止等）があるため、状態遷移を図として明確にすることが望ましい。

③ 入出力仕様
・ 入力：センサー値（10ms 間隔で更新）
・ 出力：アラーム信号（1 秒保持)、画面表示値（10ms 間隔更新）

このように、時間的要件や同期条件も合わせて設計に含める必要がある。

項目	内容
ユニット名	心拍数監視モジュール
入力	心拍数（整数）
処理	心拍数が 180 を超えたら、アラーム信号を出力する
出力	アラーム ON（TRUE）/アラーム OFF(FALSE)
エラー処理	心拍数が 0 またはマイナスの場合は、エラー表示し、アラームを OFF にする。

こうした内容をユニットごとに明確にしておくことが、詳細設計の基本である。

（４）安全性に関する設計配慮
詳細設計では、特に安全性に関わる以下の設計判断が重視される。

・ 入力データの検証とフィルタリング
外部からのデータ（センサー等）は、そのまま使用せず、値の範囲チェックや異常データの検出ロジックを記述する必要がある。これにより、不正確なデータが誤処理につながるのを防ぐ。

・ エラーハンドリング
異常発生時に、ログを記録する、フェイルセーフに移行する、使用者に警告を出す等、適切な対処方法を設計段階で決定しておく。例えば、「通信不能が 2 秒続いたら警報を発する」といった条件の記載が求められる。

・ 安全機能の二重化・独立化
高い安全性を要求される場合（安全クラス C 等）には、重要な処理のチェック処理や、別経路による監視（例：CRC チェック）を組み込む設計が必要となる。

（５）設計レビューと文書化
詳細設計書は、開発チーム内の他のメンバーや品質管理担当者によるレビューを受け、以下の観点から妥当性を確認する。

		・　要求をすべて満たしているか ・　エラー処理や異常系への配慮がなされているか ・　インターフェース仕様が一貫しているか ・　テスト可能な設計になっているか 　レビュー結果とその対応内容も文書として残し、設計根拠の一部とする。また、設計書のバージョン管理や変更履歴の記録も重要であり、開発中や保守時の追跡可能性を担保する。		
		（６）詳細設計とテストの関係 　IEC 62304 では、詳細設計と後工程である「ユニット実装」「ユニットテスト」の間に明確なトレーサビリティを求めている。 例えば、「関数 A は要求 X に対応し、テストケース TC-01 で検証される」といった関連づけが必要である。これにより、 ・　要求がすべて実装されていることの証明 ・　実装された機能が意図通りに動作するかを検証 ・　問題発生時の原因特定と対応範囲の把握 といった、開発の確実性が担保される。	●　詳細設計とテストのつながり 　詳細設計で決めた処理は、次のテスト工程（5.5～5.7）で「本当にその通り動いているか？」を確認される。そのため、 ・　設計内容が曖昧ではいけない ・　数値や動作条件がはっきりしていなければいけない ・　記録として誰でも読める形式になっていなければいけない という点が求められる。	
		（７）結論 　細分箇条 5.4「ソフトウェア詳細設計」は、設計の最終段階として、実装可能な具体的仕様を文書化する重要な工程である。この段階での設計ミスや不備は、後のバグや安全性問題の原因となり得るため、慎重かつ体系的な設計が必要である。また、詳細設計はコードを書く前の「正確な青写真」であり、それがあることで開発者は迷いなく実装を進められ、テスト担当者も仕様通りの検証ができる。品質・安全・保守性を担保するうえで、この工程の価値は極めて高いといえる。	●　まとめ 　細分箇条 5.4「ソフトウェア詳細設計」は、ソフトウェアの動きを一つひとつの部品ごとに、わかりやすく、明確に設計する工程である。この作業がしっかりしていれば、次のプログラミングやテストも順調に進み、結果として安全で信頼できる医療機器ソフトウェアが完成する。安全なソフトは、しっかりとした設計からしか生まれない。だからこそ、詳細設計は「見えないところで支える、安全の基盤」と言える。例えば、たこ焼き屋をやるとき、ただ「焼く」と言っても、 ・　タコを何グラム入れる？ ・　火加減はどれくらい？ ・　ひっくり返すタイミングは？ ・　ソースはどの順番でかける？ 等の細かい手順がある。これらを最初から決めておけば、誰でも同じ品質で作れるし、トラブルも減る。ソフトウェアでも同じで、細かいルール（詳細設計）を事前に決めておくことが、安全で正確な動作につながる。	

	5.5 ソフトウェアユニットの実装	（１）序論 細分箇条 5.5「ソフトウェアユニットの実装」は、詳細設計に基づいて実際のプログラムコード（ソースコード）を作成し、それぞれのユニットが正しく機能することを確認する工程である。ソフトウェア開発において「コーディング」と呼ばれる作業に相当し、設計された仕様を形にする段階である。このプロセスでは、単に動くコードを作るだけでは不十分である。特に医療機器に使用されるソフトウェアにおいては、安全性・信頼性・保守性を確保するために、厳格なコーディング基準、文書化、テストを伴うことが求められる。	● はじめに 細分箇条 5.5「ソフトウェアユニットの実装」は、ソフトウェアの部品（ユニット）ごとに、実際にプログラム（コード）を書く工程である。ここまでの段階で、ソフトウェア全体の構造（アーキテクチャ）や、ユニットごとの詳細な動作（詳細設計）がすでに決まっている。実装では、その設計どおりにコンピュータが理解できる言語（プログラミング言語）で命令を作る作業を行う。例えば、料理のレシピ（設計）をもとに、実際に材料を使って調理するのがこの「実装」である。設計がよくできていても、調理が雑であれば、美味しい料理はできない。それと同じく、プログラムの品質は、実装の正確さに大きく左右される。	1） GB（※）：P125-P129 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）ソフトウェアユニットとは何か IEC 62304 において「ソフトウェアユニット」とは、ソフトウェア構成要素の中でも最小の機能単位として分割された部分を指す。具体的には、１つの関数、クラス、モジュール、またはライブラリ等がこれに該当する。ユニットは単独で機能を持ち、個別に設計・実装・テストが可能であるべきとされている。例えば、 ・ 心拍数を計算する関数 ・ 入力値のチェック処理 ・ 記録データをファイルに保存するモジュール といった小さな構成要素がユニットである。	● ソフトウェアユニットとは？ ソフトウェアユニットとは、ソフトの中のひとつの小さなまとまりの部品である。これは、関数やクラス、モジュール等と呼ばれることもある。例えば、以下のようなユニットが考えられる。 ・ 温度センサーの数値を読み込むユニット ・ データを画面に表示するユニット ・ 異常値を検出してアラームを出すユニット １つ１つのユニットは、「入力を受けて、何らかの処理をして、出力を出す」という動作を持つ。そして、こうしたユニットを組み合わせることで、全体のソフトが完成する。	
		（３）実装における要求事項 IEC 62304 の 5.5 では、実装段階において以下の事項を満たすことが求められている。 ① 詳細設計に基づいたコードの記述 設計書に記された仕様・アルゴリズム・インターフェースに従って、逸脱なくコードを記述すること。開発者の独断による仕様変更は許されず、変更が必要な場合は正式な変更手順に従う。 ② コーディング規約の適用 読みやすく、保守しやすく、バグの入りにくいコードを書くために、組織として統一されたコーディング規約（記述言語の仕様等）を定め、それに従う必要がある。例えば以下のような項目が含まれる。 ・ 命名規則（変数名、関数名）	● 実装で求められること IEC 62304 では、ソフトウェアユニットを実装する際、以下のような点に気をつけることを求めている。例えば、たこ焼き屋で「何をどう焼くか」は詳細設計にあたる。「実際に焼く」ことが実装である。 ・ レシピどおりに粉を混ぜる ・ 温度や焼き時間を守る ・ 具を忘れずに入れる ・ 焦がさないように見ながら返す これらを忠実に行うことで、おいしいたこ焼きができる。勝手なアレンジをすると、焦げたり、中が生だったりしてしまう。ソフトウェアの実装も同じで、設計どおりに、丁寧に、ルールを守って作ることが重要である。	

		・ インデントとフォーマット ・ コメントの記述方法 ・ エラー処理の方法 ・ 禁止関数の使用制限（例：unsafe なメモリ操作） ③ 安全性を考慮したコーディング 特に安全クラスを高く設定したユニットについては、厳格な安全性への配慮が求められる。例えば、以下のような開発環境における特有な事象の回避については、ソフトウェアにその回避を考慮したコーディングまで必要となる場合がある。 ・ 境界チェックの徹底（バッファオーバーフロー防止） ・ null ポインタ参照の回避 ・ 例外発生時の制御フロー維持	① 設計どおりにコードを書く 第一に、詳細設計で決めた通りの処理を、正しく実装することが最重要である。勝手に設計を変更したり、自己流に変えたりせず、決められた内容を忠実にコードに反映させる必要がある。 ② コーディング規約に従う ソフトウェアを安全に作るには、統一された書き方のルール（コーディング規約）に従ってコードを書く必要がある。これにより、誰が書いても読みやすく、理解しやすいコードとなり、ミスも減る。例えば ・ 変数名を分かりやすく付ける（tempSensor、alarmFlag 等） ・ if 文やループの書き方をそろえる ・ コメントをつけて処理の意味を説明する ③ トレーサビリティを保つ 「このコードは、どの要求や設計に基づいて書かれているのか？」が後から追跡できるようにすること（＝トレーサビリティ）も重要である。コード中に「この処理は REQ-003 に対応」と記述しておくとよい。 ④ セーフティ機能を適切に実装する 特に医療機器では、安全に関わる処理（例：アラーム発報、異常値のブロック等）を確実に実装しなければならない。これらは「リスクコントロール」と呼ばれ、間違えると人命に関わることもあるため、細心の注意が必要である。 ⑤ 実装中によくあるミスとその対策 ミスの例 | 対策 入力チェックを忘れる | 入力データの正当性を必ず確認するコードを書く if 文の条件を間違える | 単体テストで条件分岐ごとに動作を確認する 単位（℃、ms 等）の取り違え | 設計書に単位を明記し、コード内でも統一する コメントが不十分でわかりにくい | なぜその処理をしているかを書く習慣をつける	
		（４）ソフトウェアユニットテスト	● 実装のあとの確認作業	

		実装されたユニットに対しては、それぞれが正しく動作するかを確認する「ユニットテスト（単体テスト）」を実施しなければならない。これは、コードが設計通りに機能しているかを検証する唯一の手段であり、品質確保の出発点となる。 ① ユニットテストの目的 ・ 各関数やモジュールの機能が正しく実装されているかを確認 ・ 異常値・境界値入力に対する耐性を検証 ・ エラー時の処理分岐が期待通りに動くかを確認 ・ リスク低減策がコードに組み込まれているかを検証 ② テストの文書化 IEC 62304 では、テストの実施だけでなく、テスト仕様書と結果の文書化が義務付けられている。記載すべき情報には以下が含まれる。 ・ テスト対象ユニットの ID ・ 入力値と期待結果の一覧（テストケース） ・ 実行環境（コンパイラ、OS、ツールバージョン） ・ テスト実施者と実施日 ・ 合否判定と証跡（ログ、スクリーンショット等） この文書は、将来の不具合対応や規制当局の審査時に、非常に重要な証拠資料となる。	ソフトウェアユニットの実装が終わったら、その後は次のステップである「ユニットテスト」に進む。このテストでは、書いたコードがちゃんと動いているかどうかを、ユニット単位で確認する。だからこそ、テストしやすいように書くこと（テスト容易性）も実装段階から意識しておくとよい。	
		（５）静的解析とコードレビュー ユニットの品質を高める方法として、静的解析ツールの活用も IEC 62304 では推奨されている。これは、コードを実行することなく構文・構造・バグの傾向を解析する技術である。例えば、 ・ 未使用変数の検出 ・ メモリリークの可能性 ・ コーディング規約違反の警告 等を自動的に発見できる。また、コードレビュー（他の開発者による確認）も併用することで、人的ミスや設計の誤解を早期に防ぐことができる。		
		（６）トレーサビリティの確保 IEC 62304 では、ソフトウェアユニットと、それに対応する以下の要素間の「追跡可能性（トレーサビリティ）」を確保することが求められている。		

		・　要求事項（どの要求に対する実装か） ・　詳細設計（どの設計仕様に基づいているか） ・　テストケース（どのテストで検証されるか） 　この追跡性が確保されていなければ、設計漏れや不十分な検証の見逃しにつながる。また、変更時に影響範囲がわからなくなり、重大な不具合のリスクが高まる。		
		（７）結論 　細分箇条 5.5「ソフトウェアユニットの実装」は、医療機器ソフトウェアにおける「ものづくり」の核心であり、信頼性と安全性を確実に実現するための実装と試験の実行が求められる段階である。設計との整合性、厳格なコーディング規約、静的解析やレビュー、トレーサビリティの確保を通じて、単なる「動作するコード」ではなく「医療現場で命を預けられるコード」を作ることが本条項の目的である。	●　まとめ 　細分箇条 5.5「ソフトウェアユニットの実装」は、医療機器ソフトウェアを作るうえで、設計どおりに、正確に、安全にコードを書く工程である。この段階では、ただ動くコードを作るのではなく、「読みやすく、ミスがなく、あとで確認や修正がしやすいコード」を目指すことが重要である。そして、安全性が命よりも優先される医療の現場では、ソフトウェアの１行１行が患者の命とつながっていることを意識する必要がある。	
	5.6 ソフトウェアの結合および結合試験	（１）序論 　細分箇条 5.6「ソフトウェアの結合および結合試験」は、複数のソフトウェアユニットを統合（結合）し、それらが正しく連携して機能するかを検証する工程である。ここでの目的は、個別に正常であっても、組み合わせることで不具合が発生しないかを確認することである。例えるなら、ユニット実装とは「部品を作ること」であり、結合試験とは「それらを組み立てて、正しく動作するかを確認すること」に相当する。医療機器においては、一つひとつの部品（ソフトウェアユニット）が正常でも、それらが適切に連携しなければ誤作動や重大事故につながるため、結合試験は極めて重要な品質保証活動である。	●　はじめに 　細分箇条 5.6「ソフトウェアの結合および結合試験」は、個別に作ったソフトウェアの部品（ユニット）同士を組み合わせ（結合）て、それらが正しく一緒に動くかどうかを確認（試験）する工程である。例えば、テレビのリモコン、画面、電源装置等、単体で動く部品を組み立てて、実際にチャンネルが変わるか、音が出るかを確認するような作業である。それぞれの部品がどんなに完璧でも、組み合わせてうまく動かないなら、それは「完成品」とは言えない。ソフトウェアも同様で、ユニットが連携して正しく機能するかを確かめることが非常に重要である。	1）GB（※）：P130-P135 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）結合とは何か 　IEC 62304 における「結合」とは、2 つ以上のソフトウェアユニット、またはモジュールを統合し、それらが 1 つの大きな構成要素として動作するようにする作業を指す。例えば、 ・　データ取得ユニットとアラーム発生ユニット ・　設定管理ユニットと保存ユニット ・　ユーザーインターフェースと制御ロジック 等、機能的に分離されていたコード同士を接続し、相互作用させる段階である。この結合においては、インターフェースの不一致、想定外のデータのやりとり、タイミングのずれ等によるエラーが発生しやすく、それらを発見・是正するために結合試験が実施される。	●　「結合」とは何か？ 　ここでいう「結合」とは、ソフトウェアの中の複数のユニットを 1 つのまとまりとして接続し、連携させて動かすことを意味する。例えば、 ・　「心拍数を計測するユニット」と「アラームを出すユニット」をつなぐ ・　「測定データを記録するユニット」と「表示するユニット」を連携させる 　このように、ユニット同士が正しい順番で情報を渡し、意図した通りに連携できるかを確認するのが「結合」である。	

		（３）結合試験とは何か 「結合試験」は、結合されたソフトウェアユニットが仕様通りに連携して動作することを検証する工程である。主に以下の点を確認する。 ・　インターフェース（関数の呼び出し、データの受け渡し）が正しく機能するか ・　ユニット間のシーケンス（処理順序）が正しいか ・　例外時の挙動が一貫しているか（例：通信エラー時のアラーム動作）	●　「結合試験」とは何か？ 結合試験は、結合したユニットどうしが連携して正しく動いているかをテストする工程である。以下のようなことを確認する。 ・　情報の受け渡しが正しく行われているか？ ・　処理の順番に間違いはないか？ ・　想定外の入力が来たとき、どう対応するか？ ・　１つのユニットのエラーが他に影響していないか？ つまり、「ユニット単体では問題なくても、つないだらバグが起きる」といった「つなぎ目のトラブル」を見つける試験が結合試験である。例えば、お化け屋敷の照明係と音響係がいて、「お化けが出たら音を鳴らす」と決めていたとする。照明係がライトを当てたときに、音響係がタイミングよく音を鳴らさないと、盛り上がらない。 ・　どのタイミングで誰が動くのか？ ・　合図は何か？（目線？ボタン？） ・　誤作動が起きたらどう止めるか？ これを事前にテストしておくのが「結合試験」であり、ソフトでもそれと同じ考えが必要である。 ●　なぜこの工程が重要なのか？ ソフトウェアの開発では、「動かない原因の多くが『部品のつなぎ方』にある」と言われている。次のような問題が起こりやすい。 ・　データの形式が違う（例：文字列と数値） ・　呼び出す順番が違う ・　片方のユニットは準備ができていないのに、もう一方が先に動いてしまう こうした「誤解」や「タイミングのズレ」は、結合して初めて見えてくる。したがって、バグの多くは結合試験で見つかる。	
		（４）試験の記録とトレーサビリティ IEC 62304 では、結合試験に対して以下のような記録を残すことが求められている。 ・　結合試験計画書（試験の目的、対象、方法、使用するツール）	●　IEC 62304 が求める結合試験の内容 IEC 62304 では、次のような試験内容を実施・文書化することを求めている。 ①　試験計画の作成	

・　テストケース一覧（入力、期待出力、実行条件）
・　実施記録（合否、ログ、スクリーンショット、担当者名、日時）
・　結果のレビュー・承認記録

　さらに、各テストケースがどの要求・設計・安全性機能に対応しているかを示すトレーサビリティマトリクスが必要である。これにより、要求に対応しない部分の見逃しや、未試験のインターフェースが存在することを防ぐ。

・　どのユニットとどのユニットを結合するか
・　何を試験するか（目的）
・　どんな方法で行うか（手順）
・　いつ、誰が、どこで行うか

② 試験ケースの作成
・　入力と出力の条件を具体的に記述する
・　正常動作だけでなく、異常動作も確認する
・　すべてのインターフェース（つなぎ方）をカバーする

③ 試験の実施と記録
・　実際に試験を行い、その結果を記録する
・　異常があれば、どこで何が起こったかを明記する
・　試験の証拠として、ログや画面キャプチャを添付する

④ 合否の判定
・　期待した動作がすべてできていれば「合格」
・　ひとつでも間違いがあれば「不合格」とし、原因を調査・修正する

●　結合試験の記録の例

試験 ID	ユニット A	ユニット B	入力	期待される結果	実際の結果	合否
TC-INT-001	心拍センサー	アラーム制御	190bpm	アラーム ON	アラーム ON	合格
TC-INT-002	温度測定	表示モジュール	37.5℃	「平熱」表示	「高温」表示	不合格

このように、何を試験して、どうなったかをきちんと残すことが求められる。

（６）エラーの検出と処置
　結合試験で発見されるバグは、以下のようなものが多い。

・　データ型の不一致（整数と小数等）
・　引数の順序ミス
・　コールバック関数の未実装
・　イベント順序のずれ
・　初期化のタイミング不一致

		・　例外処理の取りこぼし これらのバグは、「仕様書通りに書いたつもり」のコードでも発生しうるため、結合試験では「想定外の動作がないか」に重点を置いたシナリオが重要となる。発見されたエラーについては、是正処置（修正）、再試験、リスク影響評価が順に行われ、必要に応じて開発計画・設計文書の改訂が行われる。		
		（７）結論 細分箇条 5.6「ソフトウェアの結合および結合試験」は、単なるユニットの集積ではなく、「連携して安全に機能するソフトウェアシステム」の構築を目的とした工程である。個別には問題のないモジュール同士でも、連携部分に潜在するバグが大事故の引き金になることが医療機器では十分に想定される。したがって、結合試験は開発の最終盤ではなく、アーキテクチャ設計段階からその対象・観点を意識して計画されるべきである。結合の質こそが、製品全体の信頼性を左右する鍵であり、それを支えるのが本工程の本質である。	●　まとめ 細分箇条 5.6「ソフトウェアの結合および結合試験」は、個別に作ったソフトウェアの部品をつなぎ合わせ、全体として正しく動くかを確認する重要な工程である。個々のユニットが正しくても、結合部分での誤動作があれば、医療機器としては大きな問題となる。だからこそ、結合試験は「見えないところの継ぎ目」を確かめる作業であり、安全で信頼できるソフトウェアを作るうえで決して欠かせない試験である。	
	5.7 ソフトウェアシステム試験	（１）序論 細分箇条 5.7「ソフトウェアシステム試験」は、ソフトウェアのすべての構成要素が統合された状態において、仕様通りに機能するか、安全性が確保されているかを検証（Validate）する最終段階の試験である。これは、すべての設計・実装・結合を経て完成した「ソフトウェアシステム」全体を対象に行われるものであり、いわば「完成検査」にあたる工程である。この試験により、開発されたソフトウェアが医療機器としての要件を満たしているか、また、リスクマネジメントで定義されたリスク低減策が実装され有効であるかを確認する。したがって、システム試験は単なる「動作確認」にとどまらず、安全性・有効性・規制適合性を保証するための最終的な証明行為である。	●　はじめに 細分箇条 5.7「ソフトウェアシステム試験」は、ソフトウェアの開発が一通り完了したあとに、全体としてきちんと正しく動くかどうかを確認する最終的な試験である。これまでの開発工程では、ユニット単位、あるいは結合単位での確認を段階的に進めてきた。ソフトウェアシステム試験では、それらすべてをひとつにまとめて、「実際に使われる状態に近い環境で」、「全部の機能が正しく働いているか」を確認する。これは、製品として世に出す直前の、いわば「卒業試験」のようなものである。例えば、お化け屋敷で、個別の係（音響、照明、案内）がそれぞれ準備できていても、本番で全体がうまく動くとは限らない。だから事前に「リハーサル」を行い、 ・　お化け役のタイミングは合っているか？ ・　照明と音は連動しているか？ ・　お客さんが迷わずゴールできるか？ を確認する。それが「システム試験」に相当する。細かいチェックの最後に、全体として「本当にうまく動くか」を確かめる最終試験である。	1）GB（※）：P136-P142 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）ソフトウェアシステム試験の位置づけ ソフトウェア開発プロセス全体の中で、システム試験は以下の位置にある。		

		① 要求分析（5.2）で「何をすべきか」が定義され、 ② 設計（5.3, 5.4）で「どう作るか」が示され、 ③ 実装（5.5）、結合（5.6）で「構築された」ソフトウェアに対し、 ④ システム試験（5.7）で「正しく動作するか」 を検証する。この流れの中で、システム試験は「要求事項との整合性」「リスクマネジメント措置の有効性」を最終的に検証する唯一の工程である。		
		（３）システム試験の目的 IEC 62304 におけるソフトウェアシステム試験の主な目的は以下の通りである。 ・ ソフトウェア要求仕様に基づく動作確認 ・ ユーザー視点での動作の一貫性と正当性の検証 ・ リスクコントロールの実効性の確認 ・ 未検出の結合エラー・システム異常の発見 ・ リリース判定のための品質証明 この工程により、開発の全過程を通じた成果が安全かつ有効であることが初めて立証される。	● システム試験の目的 ソフトウェアシステム試験の目的は、以下の 3 つにまとめられる。 ① 要求通りの機能が実現されているかの確認 最初に決めた「やるべきこと」がすべて実現されているかを確認する。 ② 全体の機能が組み合わさって問題なく動いているかの確認 部品同士の連携が正しく、全体として安定して動いているかを見る。 ③ 安全に関わる部分が特に確実に動いているかの確認 アラームや制御装置等、人の命や健康に関わる機能は特に慎重に確認する。 ● システム試験と結合試験の違い 前のステップ（5.6）では、部品同士の「つながり」を重点的に確認した。一方でこの 5.7 のシステム試験では、製品全体として、ソフトウェアが正しく動くかを総合的に確認することがポイントである。結合試験は「部品同士が仲良くできるか」を見るのに対し、システム試験は「製品としてちゃんと役割を果たせるか」を見る、と言える。	
		（４）試験の準備と計画 IEC 62304 では、システム試験において以下の準備を文書として整えること等を求めている。 ① システム試験計画 ・ 試験の対象範囲と目的 ・ 使用するソフトウェアバージョン ・ 試験環境（OS、ハードウェア、通信条件等） ・ テスト実施体制と担当者の明確化 ・ 実施スケジュール ② 試験項目と手順	● IEC 62304 が求める試験の内容 IEC 62304 では、システム試験を行うにあたって、以下のことを明確に行うこと等を求めている。 ① 試験計画の作成 ・ どの機能を試験するのか ・ どのような条件で試験するのか ・ どのような方法（テスト手順、テストツール）で実施するのか ② 試験ケース（テストケース）の作成 ・ 入力：どんな情報をソフトに与えるか ・ 期待される出力：ソフトはどう反応すべきか	

・　各要求事項に対応したテストケース
・　入力条件、期待結果、判定基準
・　リスク低減策のテスト（例：異常入力、電源喪失等）
・　UI の操作性・表示確認

③　トレーサビリティマトリクス
・　すべての要求がどの試験で確認されたかを示す対応表

・　判定条件：正しいといえる基準は何か

③　異常時の動作確認
・　正常な動作だけでなく、間違った入力や外部トラブルがあったときにどう動くかも試験する
・　例えば、「心拍数が異常に高かったときにアラームが鳴るか」等

④　リスクマネジメントとの整合
・　安全性に関わる部分（例：危険を回避する処理）は、リスクマネジメント文書との対応関係を明確にする

⑤　試験結果の記録と報告
・　誰が、いつ、何を試験して、結果がどうだったかを記録に残す
・　合否の理由も明記する（なぜ合格と判断したか、なぜ失敗したか）

（５）試験実施の具体例
例えば、以下のようなシステム試験項目が設定される。

試験項目	入力	期待結果
心拍アラームの作動確認	心拍数 120 を入力	「高心拍数アラーム」が鳴る
測定データの保存機能	測定を 3 回実施	履歴に 3 件分のデータが記録される
通信途絶時のエラーメッセージ	通信切断	画面に「通信エラー」と表示される
電源断からの復帰動作	電源を一度切断し再投入	前回の状態から再起動される
誤設定防止ロック	禁止値を手動で設定	入力できず、警告メッセージ表示

（６）結果の評価と記録
システム試験では、すべてのテストケースに対して次のような情報が記録される。

・　テスト実施日と担当者名
・　ソフトウェアバージョン
・　実行環境

●　試験記録の例
以下は、ソフトウェアシステム試験の記録の一例である。

試験 ID	要求 ID	入力条件	期待される動作	実際の結果	合否
ST-001	REQ-101	心拍数 180	アラーム作動	アラーム作動	合格

		・　結果（合格／不合格） ・　エビデンス（ログ、画面、写真等） ・　不合格時の対応（原因分析、修正、再試験） これらは開発記録として保管され、外部監査（例：PMDA 審査、ISO 監査）にも提示可能であることが必要である。	<table><tr><td>ST-002</td><td>REQ-102</td><td>測定エラー</td><td>「再測定してください」と表示</td><td>何も表示されず</td><td>不合格</td></tr></table> 　このように、試験ケースごとに記録を残すことで、後からでも「このソフトは安全に動く」と証明できる。	
		（７）不具合への対応 システム試験で発見された問題は、以下の手順で管理される。 ・　問題の記録（バグトラッキング） ・　影響評価（他機能への波及） ・　修正内容の明確化 ・　該当範囲の再試験 ・　変更管理記録とのリンク		
		（８）結論 　細分箇条 5.7「ソフトウェアシステム試験」は、単なる最終チェックではなく、「このソフトウェアが患者の安全を守るために本当に信頼できるか」を最終的に判断する検証工程である。全体の要求仕様とリスクマネジメント方針が、実際の動作として具現化されているかを確認し、そのエビデンスをもって「使用可能」と判断することが本工程の目的である。開発者にとってはこの試験をもって製品が完成したと思いがちであるが、実際にはここでの結果が製品の信頼性を社会に証明する最も重要な根拠となる。その意味で、システム試験は製品の「出口管理」である。	●　まとめ 　細分箇条 5.7「ソフトウェアシステム試験」は、ソフトウェアが製品として本当に使える状態にあるかを、最終的に確認する非常に重要な工程である。どれだけ設計や実装がうまくいっていても、システム全体としてうまく動かなければ、医療機器としての信頼性は得られない。この試験を正しく行い、記録として残すことで、「このソフトは安全である」と胸を張って言える状態にする。それこそが、医療の現場で安心して使えるソフトウェアを生み出すための最後の関門になる。	
	5.8 システムレベルで使用するためのソフトウェアリリース	（１）序論 　細分箇条 5.8「システムレベルで使用するためのソフトウェアリリース」は、ソフトウェアを開発完了後に実際の医療機器へ組み込み、使用可能な状態としてリリースする際の最終確認と承認に関する工程である。ここでいう「リリース」とは、ソフトウェアが必要なテスト・レビュー・文書化・リスクマネジメントをすべて完了し、規格や社内手順に基づく最終判断を経て「使用してよい」と判断された状態を指す。単にソフトウェアのファイルを出力することではなく、製品として「責任をもって提供する段階」へ移行する極めて重要なステップである。	●　はじめに 　細分箇条 5.8「システムレベルで使用するためのソフトウェアリリース」は、ソフトウェアの開発・試験がすべて終わったあとに、「このソフトは医療機器の一部として安全に使える」と判断して、正式にリリース（出荷）するための最終準備をする工程である。ここまでで作ってきたソフトがどれだけうまく動いていても、最後に「問題ない」ときちんと確認されなければ、製品としては使ってはいけない。だからこのリリース工程では、「間違いがないか」「必要な書類はそろっているか」「安全な状態で保管・管理されているか」といったチェックが求められる。いわば「卒業式の前の最終チェック」や、「製品として世の中に出す前の品質確認」である。	1）GB（※）：P143-P147 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）リリースの位置づけと意義 　ソフトウェア開発プロセスの流れのなかで、リリースは以下のような「出口」にあたる。	●　ソフトウェアリリースとは？ 　「リリース」とは、ソフトウェアを正式な製品として、実際の医療機器に組み込んで使えるようにすることを意味する。もうこれ以上の修正は加えない完	

		① 要求分析（5.2）～実装（5.5）で設計・構築されたソフトウェアを ② 結合試験（5.6）、システム試験（5.7）で検証し、 ③ すべての要件が満たされたことを確認したうえで ④ 「正式な医療機器の一部として市場に供給できる」と判断する工程 この判断を誤れば、安全でないソフトウェアが製品に組み込まれ、患者に重大な危害を及ぼす可能性がある。そのため、IEC 62304 では、文書に基づく客観的証拠をもとに、正式に承認された手順でリリースを行うことを求めている。	成状態であり、以下のような準備が必要である。 ・ ソフトウェアが正しく完成していることの証明（試験合格、文書完了） ・ ソフトのバージョンをはっきりさせる（例：Ver. 1.0.3） ・ リリースされたものが保存・配布できるようになっている ・ 今後の保守や修正のときに参照できるように、記録が整っている これらが整って初めて、「このソフトはシステムの一部として安心して使える」と判断できる。	
		（３）リリースのために必要な条件 IEC 62304 は、リリースを行うためには以下のような項目がすべて満たされていることを求めている。 ① 必要なソフトウェア開発活動が完了していること ・ すべての要求仕様が実装されている ・ 設計レビュー・テスト・問題処理が完了している ・ 保留中の重大な不具合が存在しないこと ② 検証（Validation）・妥当性確認（Verification）が完了していること ・ 結合試験、システム試験において、すべてのテストが合格していること ・ リスクコントロール手段が実際に有効であることが確認されていること ③ 成果物が整備・文書化されていること ・ 要求仕様書、設計書、テスト仕様書、リスクマネジメントファイル ・ ソースコード、バージョン管理情報、ユーザーマニュアル ・ 使用上の注意、制限事項、安全警告文 ④ バージョン識別と構成管理が明確になっていること ・ ソフトウェアが一意に識別できるバージョン番号を持っている ・ どのソースコード、どのテスト結果がどのリリースに対応しているかが明確 ・ 外部ライブラリや依存コンポーネントのバージョンも記録されている これらすべての文書がレビューされ、承認を受けていなければならない。	● IEC 62304 で求められること IEC 62304 では、ソフトウェアリリースにあたって、以下のことが求められている。 ① リリースの手順を文書で定める ソフトウェアをリリースする際には、「どのような手順で、誰が確認し、どのように記録するか」といった手続きが決められていなければならない。これにより、担当者によって対応が変わるといったムラを防ぐ。 ② 完成状態のソフトを識別できるようにする 例えば、 ・ ファイル名、フォルダ名にバージョン情報を含める ・ 日付、作成者、目的（リリース用、本番用等）を明示する ・ リリース済みのファイルと開発中のファイルが混ざらないように管理する という工夫が求められる。 ③ リリース時の構成情報を記録する 「このソフトは、どのプログラム、どの設定ファイル、どのライブラリを使ってできているか」といった構成情報（ソフトウェア構成）を明確にしておく必要がある。これにより、後でバグが出たときにも、どのバージョンに影響があるかを追跡できる。 ④ すべての試験が完了していることの確認 ・ 単体テスト ・ 結合テスト	

			・　システムテスト 　これらがすべて「合格」していることを確認し、その記録を添付する必要がある。試験が未完了のままリリースしてはいけない。 ⑤ 承認の記録を残す 　リリースには、責任者の承認が必須である。「〇〇部長が、2025 年 5 月 10 日に確認し、リリースを許可した」といった記録を残すことが、トレーサビリティの確保につながる。 ●　ソフトウェアリリースの流れ（例） ① ソフトウェア開発完了（コード・設計・テスト済） ② 構成ファイル・バージョン番号の確定 ③ 試験記録や文書の確認 ④ リリース文書の作成（リリースノート等） ⑤ 承認サイン取得 ⑥ ファイルを指定のフォルダに格納し、「リリース済」と表示 ⑦ 必要な関係者（製造、保守担当）へ通知 　このような流れで、人の手による「うっかりミス」や「勝手な修正」を防ぐようになっている。	
		（４）リリース判断のプロセス 　IEC 62304 では、リリースの最終判断は「適切に権限を持つ者（通常は品質保証部門や開発責任者）」が、以下のような文書を確認し、承認を行うことで完了すると定義している。主な確認項目として、 ・　開発完了報告書 ・　システムテスト報告書と合否一覧 ・　残存リスク評価 ・　構成管理レポート ・　ユーザーマニュアルの完成とレビュー記録 　このプロセスを文書化し、社内手順として確立しておくことが、規制対応でも求められる。		
		（５）リリース後に必要な対応 　リリースはゴールではなく、新たな責任の始まりでもある。以下のような活動が直ちに必要となる。	●　リリース後の管理 　リリースは終わりではない。リリースされたソフトは、保守対象となり、後のアップデートや不具合対応の基礎資料として使われる。そのため、	

		・　リリース構成のアーカイブと保管（ソースコード、ドキュメント含む） ・　保守フェーズへの移行（IEC 62304 の箇条 6 に従う） ・　市場への出荷記録とバージョン管理の徹底 ・　顧客／現場からのフィードバックループの構築（苦情、バグ情報）	・　どのソフトが、いつ、どこでリリースされたか ・　それが今も使われているかどうか を追跡できるように、構成管理（箇条 8）と密接に関係している。	
		（６）結論 　細分箇条 5.8「システムレベルで使用するためのソフトウェアリリース」は、ソフトウェア開発プロセスの集大成とも言える工程であり、製品が「市場に出るか否か」を決める重要な判断点である。そのため、すべての試験や文書化、リスクマネジメント、承認手続きが滞りなく完了していることを、証拠とともに示す必要がある。医療機器におけるソフトウェアは、単なる技術成果物ではなく、「命にかかわる製品の一部」である。その責任を背負って「使ってもよい」と判断することが、リリースという行為の本質である。	●　まとめ 　細分箇条 5.8「システムレベルで使用するためのソフトウェアリリース」は、開発を終えたソフトウェアを医療機器として安心して使える状態にするための最終確認工程である。ここでは、動作の正しさだけでなく、記録や管理の正確さ、将来の追跡性までが問われる。つまり、「人に安心して渡せるソフトであるか」を、書類と手順でしっかり証明することが、この工程の核心である。例えば、パン屋で、焼きたてのパンを出すには最後の確認が必要である。 ・　ちゃんと焼けているか？ ・　表示価格や成分表示は合っているか？ ・　ラッピングは清潔か？ ・　アレルギー表示等はついているか？ 　こうした確認を行って初めて、「このパンをお客さんに出せる」となる。ソフトウェアでも同じで、製品として世の中に出すには、最終確認が不可欠である。	

○　IEC 62304 の箇条６の概説

箇条	細分箇条	概　要　解　説	簡　易　解　説	リファレンス・ポイント
箇条 6 「ソフトウェア保守プロセス」		IEC 62304 の箇条 6「ソフトウェア保守プロセス」は、医療機器に搭載されたソフトウェアが市場にリリースされた後も、安全かつ有効に動作し続けることを保証するために必要な活動を定義している。本プロセスの主な目的は、使用中に発見された問題やバグ、変更要求に対して組織的・計画的に対応し、製品の安全性・信頼性を継続的に維持することである。ソフトウェアはリリースされた後も、実際の使用環境における想定外の状況、ユーザーからの苦情、法規制の変更、新たなセキュリティ脅威等により、保守対応が求められる場面が多い。したがって、リリース後も規格に従った管理体制を維持し、開発時と同等の品質で保守を行うことが必要とされる。IEC 62304 では、保守プロセスを以下の 3 つの細分箇条に分けて規定している。 6.1 ソフトウェア保守計画の確立 保守の方針、体制、手順を計画として明文化し、準備する段階である。 6.2 問題及び修正の分析 市場や社内から報告された問題を収集・分析し、対応要否を判断する段階である。 6.3 修正の実装 必要な修正を設計・実装し、適切な検証と文書化を経て再リリースする段階である。 この一連の保守活動においても、リスクマネジメントとの連携が不可欠であり、ソフトウェアの変更が新たな危害を引き起こさないよう注意深く管理される必要がある。また、ソフトウェア変更によっては安全クラスが再評価されることもあるため、初回開発と同様の厳格さが求められる。	箇条 6「ソフトウェア保守プロセス」は、医療機器ソフトウェアが世の中に出たあとで、何か問題が見つかったときに、どうやってそれを直すか、その手順を定めたルールである。ソフトウェアというものは、一度完成して終わりではない。実際に使ってみると、「思った通りに動かない」「使う人が誤解する」「まれにバグが起きる」といったことがどうしても起きる。それに対してきちんと対応し、安全性を保ち続けるために、この「保守プロセス」が必要になる。ソフトウェアの保守とは、問題を見つけて、内容を調べて、必要な修正を加え、安全に動くように保ち続けることである。例えば以下のようなケースがある。 ・　バグが見つかってしまった ・　利用者から「この画面がわかりにくい」と指摘があった ・　外部の装置がバージョンアップされ、今のままだと接続できなくなった これらに適切に対応することで、ソフトを「育てながら使っていく」ことができる。IEC 62304 では、保守プロセスを次の 3 つの細分箇条に分けて整理している。 6.1 ソフトウェア保守計画の確立 → 保守をどのように行うか、あらかじめルールとして決めておく。 6.2 問題および修正の分析 → 実際に問題が起きたとき、その内容や原因をしっかり調べる。 6.3 修正の実装 → 調べた結果に基づいて、ソフトを安全に修正し、再びリリースする。 つまり、「決める → 調べる → 直す」という流れで、問題に対応していくわけである。医療機器ソフトウェアは、人の命や健康に関わるため、問題が起きたときの対応が非常に重要である。バグが見逃されてしまえば、大きな事故につながることもある。だからこそ、 ・　問題が起きたときにあわてない ・　安全性が保たれていることを証明できる ・　修正の記録が残っていて、あとで調べられる	1) GB（※）: P150-P158 2) 「保守プロセス」に係るセキュリティについては、IEC 81005-1（箇条 6 及び細分箇条 6.1 から 6.3）として、IEC 62304（箇条 6 及び細分箇条 6.1 から 6.3）に追加する要求事項について示されている。 （※）GB : IEC 62304 実践ガイドブック（じほう）

			といった体制を作っておくことが、とても大切である。	
	6.1 ソフトウェア保守計画の確立	（１）序論 細分箇条 6.1「ソフトウェア保守計画の確立」は、医療機器ソフトウェアが市場に出た後に安全かつ有効に使われ続けるための「保守活動」を、事前に組織的・体系的に計画し、明文化しておくことを求めるものである。これは「事が起きてから慌てて対応する」のではなく、「どのような問題が起きても、計画的に対処できる体制を作っておく」ことを意味している。さらに、ユーザからの問題報告を待つだけでなく、使用している外部ソフトの監視等の計画も作成しておく。医療機器は、長期間にわたり現場で使用されるものであり、発売後にも不具合の発見、法規制の変化、使用条件の変化、セキュリティリスクの発生等、多様な問題に直面する。これに適切に対応できなければ、患者の安全が損なわれるだけでなく、製品の信頼性やメーカーの社会的信用も失われる。したがって、IEC 62304 ではソフトウェア開発と同等に「保守活動」も厳格に管理すべき対象と位置づけている。	● はじめに 細分箇条 6.1「ソフトウェア保守計画の確立」は、医療機器ソフトウェアの情報の収集から、問題が起きたときや、新しいバージョンを出すときに、どのように安全に対応するかを、あらかじめ計画として決めておくことを求める規定である。ソフトウェアは作って終わりではなく、使い続ける中でいろいろな問題や変更が出てくる。それに備えて、「どのように情報を収集し、何かがあったとき、どう動くか」を事前に決めておくことが重要である。これが「保守計画の確立」という考え方である。	1） GB（※）：P151-P152 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）保守計画の目的と位置づけ 保守計画とは、リリース後のソフトウェアに対して、どのように情報を収集し、どのように変更を加え、問題を修正し、再リリースを行うかという一連の流れをあらかじめ定めた「運用上の設計図」である。具体的には、以下のような目的を持つ。 ① 保守活動に関する責任と役割を明確にする ② 問題発見から修正までの手順を体系化する ③ 再設計や再検証の条件を明示する ④ リスク評価の方法と関与タイミングを定義する ⑤ 保守活動の記録・承認・変更管理を保証する このような枠組みを事前に整備しておくことで、現場での混乱を防ぎ、速やかで安全な対応が可能となる。	● 「保守計画」とは何か？ 保守計画とは、「情報の収集の仕方、もしソフトウェアに問題が起きたときに、どう調べて、どう直して、どう確認するか」等をあらかじめ整理しておくルールの一覧表のようなものである。次のようなことを計画に含める必要がある。 ・ 誰が対応するのか（担当者やチーム） ・ どうやって情報を集めるのか（報告・記録の方法） ・ どこまで調査するのか（調査範囲と深さ） ・ どう直すのか（修正の方法と責任分担） ・ どう確認するのか（修正後のテスト） ・ 修正したことをどう記録・報告するのか（トレーサビリティ） これをきちんと決めておけば、何か問題が起きても、あわてず冷静に対応できる仕組みを作ることができる。さらに、計画は QMS で定める PDCA サイクルの最初のステップであり、製品の安全性、有効性及び品質を高めるための出発点になる。 ● なぜ「計画」が大切なのか？ 計画というと、何か面倒な書類のように感じるかもしれないが、実はとても大事な意味がある。	

			① 人の命が関わるから 医療機器ソフトウェアは、心臓の動きや呼吸、薬の量の管理等、人の命に関わる場面で使われることが多い。だから、ちょっとしたバグでも大きな事故につながる可能性がある。そんなとき、慌てて対処するとミスが増える。あらかじめ決めてある手順どおりに落ち着いて対応することで、安全性が守られる。 ② 関係者が多いから ソフトウェアの修正には、仕様作成者、設計者、プログラマー、品質管理者、リスクマネジメント担当者等、いろいろな人が関わる。「誰が何をするのか」が決まっていないと、仕事がかぶったり、逆に抜けたりしてしまう。保守計画では、それぞれの役割を明確に決めておくことで、チーム全体がスムーズに動けるようにする。 ③ 記録が必要だから 医療機器に求められるQMSでは、「なぜ直したのか」「どう直したのか」「ちゃんとテストしたのか」を後から説明できなければならない。これは法的な義務でもある。そのため、修正のたびに記録を残す仕組みを、最初から計画に組み込んでおくことが必要である。	
		（３）保守計画に含まれるべき要素 IEC 62304は、ソフトウェア保守計画に含めるべき内容を明記している。以下に主な要素を示す。 ① 保守対象の明確化 ・ どのソフトウェア製品・バージョンが保守対象か ・ 各構成要素（モジュール、サブシステム）の識別と関係図 ・ 関連する外部ライブラリやインターフェースの特定 ② 保守責任体制の確立 ・ 問題受付・分析・対応の責任者（役職・部門）の明示 ・ 承認権限の所在（例えば品質保証部門が修正リリースを承認） ③ 問題受付・分類・優先順位づけの手順 ・ 問題報告の受付窓口（顧客、サービス部門、社内テスト部門等） ・ 関連する外部ライブラリのバグ情報の監視 ・ バグと改善提案、セキュリティ脅威等の分類方法 ・ 臨床リスクに基づく優先順位の定め方（例：重大→即時対応） ④ 修正対応の方針	● IEC 62304における保守計画の内容 IEC 62304では、保守計画には最低限、次のような内容が含まれていなければならない。 | 内容	説明 | |---|---| | 保守プロセスの手順 | 問題を受け付けてから修正・再リリースまでの流れ | | 保守活動の責任者 | 誰がどの部分を担当するか（例：調査担当、実装担当等） | | 必要な入力情報 | 何をもとに調査を始めるか（例：バグ報告、ログファイル等） | | 出力として残す記録 | 修正の内容、テスト結果、バージョン情報等 | | 修正が必要かどうかの判断基準 | 重大性、リスクの大きさ、頻度等によって判断 | | 変更管理との連携 | 修正内容が変更管理プロセスにも記録されること | これらが整っていれば、「もし問題が起きても、きちんと対応できますよ」と証明できる。お好み焼き屋の営業中に、鉄板が急に壊れたとする。そのとき、	

・　修正要否の判定基準（「再現性があるか」「安全性に影響するか」等）
・　設計変更が必要な場合の手順（設計再レビュー、再テスト等）
・　修正が他機能へ与える影響評価の方法

⑤ リスクマネジメントとの連携
・　問題が新たなリスクを生じさせるかの分析
・　修正が既存のリスク制御策に悪影響を与えていないかの確認
・　リスクマネジメントファイルとのトレーサビリティの確保

⑥ 修正後の検証・妥当性確認
・　修正コードのユニットテスト、結合テスト、システムテストの実施条件
・　既存のテストケースの再使用と、新規追加テストの判断基準

⑦ リリース管理と文書化
・　再リリース時のバージョン番号付与ルール
・　リリースノート、改訂履歴、影響範囲の記録
・　配布・更新時のユーザー向け注意事項

・　お客様への状況説明や、注文のキャンセルや返金等の対応はどうするか？
・　どこに連絡するのか？
・　代わりの鉄板はあるか？
・　誰が修理や手配をするか？

が決まっていないと、全員がバタバタして、結局販売が止まってしまう。これを防ぐためには、事前に「トラブルが起きたらこう動く」というマニュアルや計画書が必要である。ソフトウェアでもまったく同じである。IEC 62304 では、ソフトウェアの安全クラス（A～C）によって、保守計画の内容や厳しさが変わり、下記のようなイメージとなる。

クラス	対応の内容
A	最小限の保守計画でも許される
B	調査・修正・記録の流れをしっかり定める必要がある
C	命に関わるため、リスク分析と連動した厳密な計画が必要である

とくにクラス C では、修正によって安全性が下がっていないかどうかを別の人がチェックする仕組み（レビューや承認）も含めなければならない。

（４）保守計画の作成と維持
IEC 62304 では、保守計画を単なる「紙の計画書」で終わらせてはならない。開発中に作成された保守計画は、リリース後も定期的に見直され、運用実態に即した内容に保たれなければならない。特に以下のような変更があった場合は計画の更新が必要である。

・　製品構成の変更（新機能追加、外部インターフェース変更等）
・　運用体制の変更（保守を外部に委託した場合等）
・　法規制や規格要件の改定（例：サイバーセキュリティガイドラインの更新）

また、保守計画書は品質マネジメントシステム（QMS）と統合されている必要があり、ISO 13485 や ISO 14971 と整合した形式で運用されることが重要である。

● 保守計画を活かすために
計画は作っただけでは意味がない。実際に使われてこそ価値がある。そのためには、

・　社内で共有されていること
・　新しい問題が出たときにも計画に沿って動いていること
・　計画そのものも、必要に応じて見直されていること

が大切である。つまり、保守計画は「紙のルール」ではなく、実際に動く「仕組み」でなければならない。

（５）実務上の注意点

		・　属人化を避ける：担当者が変わっても継続して運用できるよう、手順は明文化され、教育も含めて体制化されている必要がある。 ・　トレーサビリティの維持：変更箇所が元の要求・設計・リスクと結びついていない場合、変更による副作用を見逃す恐れがある。 ・　過去の保守履歴の一元管理：以前のバグ情報や修正内容を活用することで、同じ不具合の再発防止や設計改善に活かせる。		
		（６）結論 細分箇条 6.1「ソフトウェア保守計画の確立」は、ソフトウェアの安全性と有効性をリリース後も維持し続けるために不可欠な「体制づくり」を担うものである。開発が終わっても製品の責任は終わらない。むしろ、使用されてはじめて現れる問題やリスクに備えるために、開発時点から計画的に保守体制を整えておくことが、医療機器メーカーとしての信頼性を支える。この工程をおろそかにすると、重大事故やリコール対応に追われる事態になりかねない。反対に、堅実な保守計画があれば、発生する問題にも迅速・確実に対応でき、製品の価値と社会的信頼を持続的に高めていくことができる。	●　まとめ 細分箇条 6.1「ソフトウェア保守計画の確立」は、医療機器ソフトウェアに問題が起きたときに、あわてず、安全に、確実に対応できるようにするための準備をしておく規定である。これをしっかり決めておくことで、ソフトがリリースされた後も長く、安全に使い続けることができる。	
	6.2 問題及び修正の分析	（１）序論 IEC 62304 の細分箇条 6.2「問題及び修正の分析」は、医療機器ソフトウェアの運用中に発見されたバグ、故障、ユーザーからの苦情、セキュリティ脅威、誤使用による不具合等、あらゆる「問題」に対して、それがソフトウェアの修正を要するかどうかを体系的に評価・判断するプロセスを定義したものである。医療機器におけるソフトウェアの問題は、放置すれば患者の安全に直結する重大事故につながりかねない。反面、すべての報告に対して一律に修正対応をすれば、リソースが過剰に消費されるだけでなく、かえって不安定な変更が繰り返される原因となる。したがって、本条項では、「どの問題が、どのような根拠により、どのように処理されるべきか」を定め、計画的かつ合理的な判断に基づいた保守を行うことが求められている。	●　はじめに 細分箇条 6.2「問題および修正の分析」は、医療機器ソフトウェアで何かトラブル（バグや不具合等）が起きたときに、それがどんな問題なのか、どこに原因があるのか、どう直せばよいかをしっかり調べる工程である。例えば、病院で使われている心電図のソフトが誤った心拍数を表示したとしたら、それは大きな問題である。しかし、「ただ直す」のではなく、まず「なぜそんなことが起きたのか」を深く調べることが大切である。それが、この箇条で求められている「分析」である。例えば、お好み焼きを焼いていたときに、「焦げやすい」「なかなか火が通らない」といった問題が起きたとする。このとき、ただ火加減を変えるだけではなく、 ・　鉄板の温度が高すぎる？ ・　生地の水分が足りない？ ・　材料の量にムラがある？ 等をきちんと調べてから、対応を決める必要がある。それと同じで、ソフトウェアでも「ただ直す」のではなく、「なぜ問題が起きたのかをきちんと調べる」ことが、次の対応の正しさを決めるカギとなる。	1）GB（※）：P152-P156 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）問題の定義と範囲 IEC 62304 において「問題」とは、以下のような事象を含む広い概念である。	●　「問題」とは何か？ ここで言う「問題」とは、次のような状態のことである。	

		・　使用中に発生したソフトウェアの不具合（バグ） ・　ハードウェアとの不整合や通信エラー ・　設計仕様との食い違い ・　ユーザーからの苦情・異常報告 ・　臨床現場での予期せぬ挙動 ・　サイバーセキュリティに関する脆弱性 ・　不十分または誤解を招くユーザーインターフェース 　これらの問題はいずれも、最終的に「安全性に影響を及ぼすかどうか」を判断基準として扱われる。	・　ソフトが期待通りに動かない（例：誤作動する、固まる） ・　表示が間違っている（例：温度表示がズレている） ・　操作しにくい、使い方を誤解しやすい ・　過去にうまく動いていたのに、新しい状況でうまくいかない 　こうした問題が現場から報告されたとき、それを放置するのではなく、分析して対応すべきか判断するのがこの工程である。	
		（３）問題の管理フロー（一般的な流れ） 　細分箇条 6.2 で規定される活動の基本的な流れは、以下の 5 つのステップで構成される。 ①　問題の受付 問題は様々な経路から報告される。例えば、 ・　現場の医療従事者からの苦情や問合せ ・　サービス担当者の現場観察 ・　内部品質試験やフィールドテストでの検出 ・　外部審査機関や規制当局からの指摘 　受付時には、報告内容を明確に記録し、日時、発見者、状況、発生頻度等を記載した「問題報告書」等を作成する必要がある。 ②　再現性と影響範囲の確認 　報告された現象が再現可能かどうかを確認する。再現性のない事象に対して修正を行うのはリスクが大きく、まずは実験条件、使用環境、再現手順の把握が優先される。また、同様のコードを使用している他の機能・機種・バージョンに波及している可能性があるかを技術的に調査し、問題の「影響範囲分析」を行う。ここでは以下の情報が活用される。 ・　構成管理データ（どの製品が同じモジュールを使用しているか） ・　過去の修正履歴 ・　トレーサビリティ情報（設計・要求・リスクとの関連） ③　安全性への影響分析（リスク評価）	●　分析とは何をすることか？ 　「分析」とは、簡単に言えば、「何が原因で起こったのかを突き止めること」である。ソフトウェアの問題では、次のようなことを調べる。 ①　再現できるか？ 　同じ操作をしたときに、また問題が起きるかどうか。 ②　どのバージョンで発生したか？ 　過去のバージョンでは問題がなかったのか？ ③　問題が起きた条件は何か？ 　特定のデータ、操作順、時間帯等に依存しているか？ ④　どのソフトの部分に関係しているか？ 　特定のユニットや OS、機能に限られているか？ ⑤　原因は設計ミス？実装ミス？運用の誤り？ 　人間の使い方の問題であれば、教育やマニュアル改善で解決できるかもしれない。 　これらを調査し、問題の本質に迫ることが「修正の前に必要な分析」である。 ●　修正すべきかどうかをどう判断するか？ 　問題が見つかったら、すぐに修正すればよいと思うかもしれないが、医療機器ソフトウェアでは勝手に変更してはいけない。なぜなら、修正したことで別のところに悪影響が出る可能性があるからである。そのため、以下のような基準で「修正が必要かどうか」を判断する。	1）　リスク評価については、細分箇条 7 や ISO 14971 との関係に着目する。

最も重要な工程が「安全性への影響分析」である。ここでは、報告された問題が患者やユーザーにとってどの程度の危険性を有するかを評価する。評価項目は以下の通りである。

・　問題によって生じる可能性のある危害（例：誤投薬、診断ミス）
・　その危害が生じるまでの因果関係（ハザード分析）
・　問題発生の頻度（実績データ、発生条件の一般性）
・　既存のリスク低減策が問題に対応しているかどうか

これらの評価結果に基づいて、修正の必要性の有無と、必要であれば修正の緊急度が判断される。

④ 修正要否の判断

リスク評価の結果、「修正が必要」とされた場合、その修正は以下のように分類される。

・　即時対応（クリティカル）：生命に関わる危害の可能性がある場合。緊急パッチや即時配信が求められる。
・　短期対応（優先度高）：医療行為に支障を来す可能性があるが、回避策が存在する場合。
・　計画的対応（優先度中）：安全性には直接関係しないが、改善すべき動作不良や誤動作。
・　対応不要（優先度低または記録のみ）：安全性に影響しない UI の軽微な不具合、表記ミス、操作ガイドの補足等。

対応不要とされた場合も、その判断理由と根拠を文書化しておくことが重要である（なぜ直さなかったのか、が説明できるようにするため）。

⑤ 次工程（修正実装）への引き渡し、または却下と記録保存

問題に関する以下の情報を保守記録として残す。

・　受付日と報告内容
・　再現性と影響範囲の有無
・　リスク評価結果
・　修正の要否判断と優先順位
・　承認者の氏名と日時

「修正が必要」とされた場合は、次工程である細分箇条 6.3「修正の実装」に正式に引き渡され、設計・テスト・リリース手順が開始される。

判断基準	具体例
安全性に関わるか？	誤表示が命に関わる場合はすぐ修正が必要
法律や規制に違反していないか？	表示方法が法令に違反していれば対応が必要
使用者が誤解する可能性があるか？	操作ミスを誘発するような画面配置であれば見直す
発生頻度は高いか？	毎回起こるなら深刻、年に 1 回なら慎重に判断

このように、問題の重大さと影響を分析してから、修正するかどうかを決める。

● リスクマネジメントとの関係

この箇条では、「リスクマネジメント（危険管理）との連携」が非常に重要である。というのも、問題の中には人の命や健康に直接関わるものもあるため、それが新たなリスクを引き起こしていないかを確認する必要がある。例えば、

・　バグによって、異常な温度でもアラームが出ない
・　通信エラーで患者データが消えてしまう
・　アップデートで以前のセーフティ機能が無効になった

こうしたケースでは、リスクマネジメントの担当者と連携し、必要なら再分析や再設計を行う必要がある。

● 分析の記録を残すことの重要性

問題を調査したら、その内容は必ず記録に残さなければならない。これは、「あとで何があったかを説明できるようにする」ためでもあり、「同じ問題が起きたときにすぐ対処できるようにする」ためでもある。記録に残すべき内容の例として、

・　問題の内容（いつ、どこで、何が起きたか）
・　分析の結果（原因とその根拠）
・　修正の有無（対応したかどうか、理由）
・　修正の影響（他の部分に影響がないか）

この記録があれば、後から調査が必要になったときも、スムーズに調べることができる。

		（４）結論 細分箇条 6.2「問題及び修正の分析」は、医療機器ソフトウェアが安全に使われ続けるための「品質とリスクの守備線」である。市場に出たソフトウェアに対して、感覚や場当たり的な判断ではなく、体系的な根拠に基づいて問題の重大性と修正の要否を判断するこの工程は、開発段階にも匹敵するほどの重要性を持つ。本プロセスが適切に運用されていれば、現場からの信頼を得られ、リスク対応のスピードと的確性も向上する。逆にここが曖昧であれば、問題が見過ごされる、または過剰に反応して製品が不安定化する等、組織としての健全性に影響を及ぼしかねない。品質維持と市場対応力が密接に関係してくる。	● まとめ 細分箇条 6.2「問題および修正の分析」は、医療機器ソフトウェアに何か問題が起きたときに、それがどんな原因で起こったのかを丁寧に調べ、安全に対応できるかどうかを判断する重要な工程である。ここでの判断を間違えると、余計な修正でソフトが壊れたり、逆に修正が遅れて患者に危害を与えたりすることにもなりかねない。だからこそ、問題への第一歩は「分析」であり、その質がソフトの信頼性と安全性を大きく左右する。	
	6.3 修正の実装	（１）序論 細分箇条 6.3「修正の実装」は、医療機器ソフトウェアにおける保守活動の中で、問題や不具合が修正すべきと判断された場合に、その修正を安全かつ計画的に実施するための工程を定めたものである。本条項は、単に「コードを修正する」ことを指しているのではなく、リスクマネジメント、設計文書の更新、テスト、リリース、構成管理、記録等、あらゆる活動を網羅する「修正ライフサイクルの統制プロセス」である。ソフトウェア修正は、新たな機能追加と同様に、ソフトウェア全体の挙動や安全性に影響を与える可能性がある。そのため、開発時と同等の厳しさで設計・実装・検証を行い、さらに「どのように、なぜ修正したか」を証明可能な形で記録・管理することが求められる。	● はじめに 細分箇条 6.3「修正の実装」は、ソフトウェアに問題が見つかり、それが修正すべきものだと判断されたあとに、実際にその問題を直す作業（実装）を行うときのルールや注意点を定めたものである。つまり、これまでに問題の報告を受けた（6.1）、問題を分析して修正の必要性を判断した（6.2）という流れがあり、この 6.3 ではその結論に基づいてソフトを安全に、確実に直すための作業を行う段階に入る。	1） GB（※）：P157-P158 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）修正の実装におけるステップ IEC 62304 の細分箇条 6.3 では、修正の実装を次のようなステップで体系的に進めることが規定されている。 ① 修正に関する変更の決定と文書化 まず、細分箇条 6.2 で「修正が必要」とされた問題に対して、どのような修正を行うかを正式に決定し、その変更内容を文書化する。これには以下のような情報が含まれる。 ・ 修正の目的（例：エラーメッセージの不具合修正） ・ 修正対象のモジュール・関数・ファイル名 ・ 修正前後で動作がどう変わるかの説明 ・ 修正によって他機能への副作用が予想されるかどうか これにより、関係者間で修正内容の理解を共有するとともに、設計変更の正当性の記録としても機能する。 ② 影響範囲の特定とリスク評価	● 「修正の実装」とは何をすることか？ ソフトウェアの修正とは、次のようなことを行うことである。 ① ソースコードの修正 バグを生んでいるプログラムの部分を書き換える ② 設計文書の更新 どこをどう変えたのか、元の設計書に反映させる ③ 影響範囲の確認 修正したことによって他の部分に影響が出ないか確認する ④ 再テスト 修正後に、ちゃんと直っているかを試験する ⑤ 記録と承認 修正の内容と結果を文書にまとめ、関係者が確認する これらを正しい手順で、かつ記録を残しながら行うことが「修正の実装」であり、これが適切に行われることで、安全性と信頼性とが保たれる。 ● 修正にも「計画」が必要	

		修正が及ぼす影響範囲は技術的な面と安全性の面から評価される。 ・　技術的評価：修正対象が依存しているコードの把握、変更によって影響を受ける可能性のある他の機能を特定。 ・　リスク評価：修正によって新たなリスクが生じないか、既存のリスク制御策が損なわれないかを ISO 14971 に基づいて再評価する。 このステップが不十分であると、修正によって新たな不具合が発生するリスクが高まる。 ③ 設計の変更（必要に応じて） 修正が単純な文法的ミスや画面表示の訂正であれば、設計の変更を伴わない場合もあるが、根本的な動作仕様や安全対策を変更する場合は、設計文書の改訂が必要である。例えば、 ・　状態遷移図の変更（異常時の分岐を追加する等） ・　新しい入力チェック機構の導入 ・　アラームの条件式や閾値の変更 設計文書の変更時には、開発時と同様の設計レビューを実施し、妥当性を確認することが求められる。 ④ 実装（コーディング）の実施 詳細設計に基づいて、実際にソースコードを修正する。修正に際しては、次のような事項が遵守されなければならない。 ・　組織のコーディング規約に従うこと ・　修正箇所を明確にコメントで示すこと（履歴管理） ・　単体テストが可能なように構造化されたコードにすること この工程では、再利用性や保守性も意識した実装が求められ、場当たり的なパッチ的修正は避けるべきである。 ⑤ 検証および必要な妥当性確認 修正後は、以下の検証を実施する。 ・　ユニットテスト：修正箇所の機能が正しく動作することの確認。 ・　結合テスト：関連モジュール間のデータのやりとりが正しく行われるかの確認。	ただコードを変えるだけでは、医療機器ソフトのような安全が求められるシステムでは通用しない。必ず以下のような「計画的な進め方」が必要である。 ・　何をどこまで直すのか（修正範囲の決定） ・　どうやって直すのか（方法の選択） ・　誰が直すのか（担当者の明確化） ・　いつまでに直すのか（スケジュール） このように、「やみくもに直す」のではなく、「設計・実装・確認・記録」のサイクルを守ることが求められる。 ● 設計文書と記録の更新 修正を行った際には、その変更が設計段階にまで影響する場合は、設計文書や要求仕様書も更新する必要がある。例えば、 ・　入力データの扱いを変えた ・　表示のロジックを変えた ・　セーフティ機能を追加した 等の修正を行えば、もともとの「設計と違う動作」をすることになる。そのため、設計文書を「今のソフトに合わせて書き直す」ことで、将来の保守やテストがやりやすくなる。これがトレーサビリティ（追跡可能性）を保つために重要である。 ● 修正のテスト（回帰試験） 修正を加えた場合には、「直したところが正しく動くか」を確認するだけでなく、モジュール間の呼び出し関係を確認し「他のところに悪影響を与えていないか」を確認する必要がある。これを「回帰試験」という。例えば、 ・　アラームの修正をしたが、他の警告表示に影響していないか？ ・　新しい機能を追加したことで、古いデータとの互換性が失われていないか？ こうしたチェックを通じて、「安心して使える修正」であることを確認する。 ● 修正の管理：変更管理プロセスとの関係 IEC 62304 では、「修正の実装」を行うときは、箇条 8 の「変更管理プロセス」と必ず連携することを求めている。これはつまり、	

		・　システムテスト：修正がシステム全体に与える影響がないかを確認。 　また、安全クラスが B または C のソフトウェアであれば、妥当性確認（validation）すなわち「意図された使用に対して正しく機能するかどうか」を改めて確認する必要がある。 ⑥ 記録と構成管理の更新 　修正内容が確定したら、それを以下のような形式で記録する。 ・　修正内容の説明 ・　修正者、実施日 ・　関連する問題報告番号、変更要求番号 ・　対象バージョンと影響モジュールのリスト 　また、構成管理システムにおいては、修正が加えられたソースコード、設計文書、テスト記録のバージョンを明確に管理し、リリースごとの構成として凍結・保管される必要がある。 ⑦ ソフトウェアの再リリースと通知 　修正が完了し、必要な検証と記録が整った後、ソフトウェアは新しいバージョンとして正式に再リリースされる。この際に求められるのは、 ・　リリースノートの作成：どの問題が、どのように修正されたかを利用者に通知。 ・　ユーザー向けの注意事項提示：操作手順や仕様変更点、更新の適用方法等。 ・　修正内容に対する承認プロセスの完了：品質保証部門等による最終承認。 　なお、安全性等に関わる重大な修正である場合は、PMDA 等の審査機関への再申請や規制当局への報告等が必要になることもある。	・　何を、いつ、どう直したのか ・　それによってどんな影響が出たのか ・　誰が承認したのか をすべて記録し、変更として正式に管理することが義務づけられている。変更管理台帳や、バージョン管理システム等を活用して、誰が何を変えたかが後からはっきりわかる状態にしておくことが大切である。 ●　修正の承認 　医療機器ソフトのように命に関わるシステムでは、修正を勝手に製品に反映することはできない。必ず「誰か第三者（上司や品質保証部門等）」が内容を確認し、承認する必要がある。これによって、 ・　「独断での修正ミス」 ・　「無記録の危険な変更」 等を防ぎ、常に複数の目で品質を守る体制を作ることができる。	
		（４）結論 　細分箇条 6.3「修正の実装」は、問題を発見し、修正を要すると判断された後に、その変更を安全に、かつ管理された方法で実施するための中心的な工程である。ソフトウェアの修正は小さな操作であっても、医療機器においては致命的な影響を与える可能性があるため、「正しく直す」こと以上に「安全に直す」ことが重視される。設計の変更、コードの実装、再テスト、記録、リリースといった一連のプロセスを、手順通りに、根拠を持って遂行することが、製	●　まとめ 　細分箇条 6.3「修正の実装」は、医療機器ソフトウェアで問題が見つかったときに、安全かつ確実にそれを修正するための手順とルールを定めた工程である。この工程では、「ただ直す」ではなく、 ・　設計と整合性が取れているか？ ・　他への影響はないか？	

		品の信頼性と組織の信頼性を支える基盤である。	・　記録は残っているか？ ・　第三者の確認が行われているか？ といった多くの視点が求められる。つまり、「安全に責任を持って直す」ための仕組みこそが、この工程の本質である。例えば、お好み焼きを焼くときに「焼き時間が長すぎる」という問題が出たとする。これに対して、「火力を上げてみる」という対応をとったとき、 ・　本当に火力を上げるだけでよいのか？ ・　焦げやすくならないか？ ・　他の班員にも伝わっているか？ ・　この対応を記録しておかないと、次にまた同じ問題が起きるかもしれない というように、その場しのぎではなく、「確認して、記録して、安全に進める」という姿勢が必要である。ソフトウェアの修正も同じで、「直す」だけでなく、「安全に直す」「記録を残す」ことが求められる。	

○　IEC 62304 の箇条７の概説

箇条	細分箇条	概　要　解　説	簡　易　解　説	リファレンス・ポイント
箇条 7「ソフトウェアリスクマネジメントプロセス」		IEC 62304 の箇条 7「ソフトウェアリスクマネジメントプロセス」は、医療機器ソフトウェアに特有のリスクを識別・評価し、そのリスクを適切に管理するためのプロセスを定めたものである。一般的なリスクマネジメントの枠組み（ISO 14971）を土台としつつ、特にソフトウェアが単独で引き起こす危害や、ソフトウェアの振る舞いがシステム全体の安全性に与える影響に特化した補足的な要件がこの箇条に記されている。医療機器におけるソフトウェアは、目に見える物理的な動作ではなく、プログラムされたロジックによって患者の診断や治療、監視を支える極めて重要な構成要素である。したがって、意図しない動作や設計ミス、データの不整合等によって患者に危害を及ぼすリスクが存在する以上、それらを開発段階から体系的に分析・制御・検証し、さらに運用・保守の段階においても継続的に管理しなければならない。IEC 62304 におけるリスクマネジメントの特徴は、「ソフトウェアが単独で危険状態を引き起こす可能性」に焦点を当てている点である。例えば、アラームを鳴らすべきところで鳴らない、誤った計算結果を表示する、データを保存しない等の不具合は、直接的な機械的故障とは異なり、ソフトウェア特有のロジックや処理の問題から発生する。箇条 7 は、以下の 4 つの細分箇条で構成されている。 7.1 危険状態を引き起こすソフトウェアの分析 　どのようなソフトウェア機能が危害の原因となり得るかを特定・分析する。 7.2 リスクコントロール手段 　危害のリスクを低減するための具体的な対策を講じる。 7.3 リスクコントロール手段の検証 　リスク低減策が有効に機能していることをテストや証拠により検証する。 7.4 ソフトウェア更新のリスクマネジメント 　保守や更新によって生じる新たなリスクを再評価し、管理する。 　これらの活動は、ソフトウェア開発・保守の全工程を通じて継続的に実施され、医療機器としての「安全性の担保」を支える重要な基盤となる。	箇条 7「ソフトウェアリスクマネジメントプロセス」は、医療機器に使われるソフトウェアが人の命や健康に悪い影響を与えないように、リスクを事前に見つけて、減らすための考え方や手順を定めたものである。ソフトウェアは目に見えない電気的な仕組みで動いているため、たとえ見た目では問題がなくても、「想定外の動作をする」「間違った情報を出す」といった危険がひそんでいることがある。それが医療機器の場合、患者の治療を間違えたり、重大な事故につながるおそれがある。だからこそ、ソフトを開発するときに「どこが危ないか」を分析し、「どうすれば安全にできるか」を考えておくことが大切である。リスクマネジメントとは、「危険を見つけて、それをできるだけ小さくするための工夫」を意味する。IEC 62304 では、ソフトウェアに特化して以下のような活動が求められている。 ・　ソフトが引き起こす可能性のある危険を見つける（分析） ・　その危険にどう対処するかを考える（コントロール） ・　対処がきちんと効いているかを確認する（検証） ・　ソフトを更新するたびにリスクが増えないかチェックする（更新時対応） 　これらの考え方を実行することによって、「見落としによる事故」を防ぎ、安心して使える医療機器ソフトウェアをつくることができる。IEC 62304 では、ISO 14971 という医療機器全体に関するリスクマネジメントの規格をベースにしながら、ソフトウェア特有の危険（例えば計算ミスや画面のフリーズ等）にも対応できるように補足している。つまり、「ソフトウェアを作る人が行うべきリスク対応のルール」がこの箇条にまとめられている。	1）GB（※）: P160-P170 2）医療機器ソフトウェアセキュリティに係るリスクマネジメントプロセスについては、IEC 81005-1（箇条 7 及び細分箇条 7.1 から 7.5）に示しており、「ソフトウェアリスクマネジメントプロセス」の一部として要求される。 3）各箇条、細分箇条においてリスクマネジメントに係る要求事項は、本箇条を参照する形になる。 4）注意すべき点として、組織によっては ISO 9001 と ISO 13485 を併用している場合や、開発の一部を ISO 9001 のみに準拠した社内他部門あるいは外部委託先に依頼している場合がある。このような場合でも、医療機器としての最終製品に対するリスクマネジメントは ISO 13485 の要求事項に基づいて実施されなければならない。すなわち、ソフトウェア開発プロセスの一部が ISO 9001 に基づいて管理されていたとしても、その成果物は最終的に ISO 13485 の枠組みの中でリスクマネジメントの対象となる。特に臨床的安全性の観点からは、IEC 62304 に基づくソフトウェア特有のリスク管理が必須である。リスクマネジメントの実施において課題となるのが、脅威分析における発生頻度の客観的なデータ収集の困難さである。特にソフトウェアにおいては、故障モードの発生確率を定量的に予測することが難しい場合が多い。このような状況下でリスクコントロールマトリックスの設定根拠を明確にするためには、以下のアプローチが有効であ

				る。 １）類似製品や関連技術における過去の不具合事例の体系的な収集と分析 ２）開発初期段階からのハザード識別とリスク評価の繰り返し実施 ３）複数の専門家による評価（デルファイ法等）の活用 ４）安全側に立った保守的な評価アプローチの採用 ５）不確実性が高い場合の追加的な安全対策の実施 リスクマネジメントを単なる規制要件の充足としてではなく、製品の安全性と有効性を確保するための中核的なプロセスとして位置づけるべきである。 （※）GB：IEC 62304 実践ガイドブック（じほう）
	7.1 危険状態を引き起こすソフトウェアの分析	（１）序論 細分箇条 7.1「危険状態を引き起こすソフトウェアの分析」は、ソフトウェアが単独あるいは他の要因と組み合わさって患者や使用者に危害をもたらす可能性がある「危険状態（hazardous situation)」を分析し、それらを未然に検出・制御するための出発点となるプロセスである。ソフトウェアは、電気的・機械的な部品のように物理的な劣化や摩耗を起こすものではないが、設計の誤りや処理の失敗、操作ミスへの対処不足等によって、機器の挙動に重大な影響を与える。特に医療機器では、ソフトウェアによるアラームの誤判断、測定値の誤表示、治療モードの選択ミス等が、生命に関わるリスクを引き起こす可能性がある。そのため、本条項ではソフトウェアに固有のリスクを体系的に洗い出すための分析作業が必須とされている。	● はじめに 細分箇条 7.1「危険状態を引き起こすソフトウェアの分析」は、医療機器のソフトウェアが患者にとって危ない状態（＝危険状態）を引き起こす可能性がないかを、あらかじめ分析するための作業を指している。医療機器ソフトウェアは、病院で人の命や健康に直接関わる道具として使われるため、ソフトが間違った動きをすると、大きな事故に発展することがある。例えば、誤った薬の量を指示してしまったり、アラームが鳴るべき時に鳴らなかったりすることが命取りになる。そういった「ソフトが原因で危ない状態になる可能性」がどこにあるのかを調べ、見つけ出す作業が、この分析の目的である。 ● なぜこの分析が必要なのか？ 理由はシンプルである。安全でないソフトは、医療機器として使うべきではないからである。ソフトウェアの危険は、目に見えず、発見されにくいため、問題が起きてからでは遅い。だからこそ、 ・ 問題が起こる前に、 ・ ソフトがどんな悪影響を及ぼしそうかを予測し、 ・ そのリスクを減らすための対策を考える という流れが必要である。この 7.1 の分析は、その最初の「何が危ないか？」	1） GB（※）：P162-P165 （※）GB：IEC 62304 実践ガイドブック（じほう）

			を見つけるところにあたる。	
		（２）本条項の目的 IEC 62304 における本条項の目的は次のとおりである。 ① ソフトウェアが引き起こす可能性のある危険状態を特定すること ② それらの危険状態が生じるまでの因果関係を明らかにすること ③ 対象となるソフトウェア構成要素と安全クラスの妥当性を確認すること ④ 今後のリスクコントロール活動（箇条 7.2 以降）に向けた基礎資料とすること この分析によって初めて、適切な設計方針や試験戦略、安全対策が計画可能となる。	● IEC 62304 が求める内容 IEC 62304 では、ソフトウェアに関して以下の 2 つを必ず分析することを求めている。 ① ソフトが直接危険状態を引き起こす場合 例えば、 ・ ソフトがアラームを鳴らすはずの状況で、処理が止まってしまった ・ 数値の表示ミスによって、医師が誤判断してしまう これは「ソフト自体が間違った動作をして、直接危ない状態になる」パターンである。 ② ソフトが間接的に危険状態を引き起こす場合 例えば、 ・ センサーから正しいデータが届かないのに、ソフトが誤って処理してしまう ・ 外部との通信エラーが起きたときに、ソフトが黙ったままで対応しない このように、ソフトが本来の目的以外で、他の部品や環境の問題によって間違いを起こすケースも含めて、分析しなければならない。	
		（３）分析に含まれるべき内容 IEC 62304 において、危険状態の分析には次のような要素を含むことが求められている。 ① 危険状態（Hazardous Situation）の特定 危険状態とは、「ソフトウェアの不適切な動作が、危害を引き起こす前段階の状態」である。具体例としては以下が挙げられる。 ・ アラームが必要な場面で鳴らない ・ モニターの表示が正しく更新されない ・ 記録データが誤って上書きされる ・ 制御値が設定と異なる状態で動作している ・ エラーが発生してもユーザーに通知されない	● 危険状態とは何か？ 「危険状態」とは、患者に対して害（けがや健康被害）が起きる可能性のある状況のことである。例えば、次のような状態が危険状態にあたる。 ・ 心電図のソフトが誤って「異常なし」と判断し、治療が遅れてしまう ・ 注射ポンプが指定以上の薬液を送り、過剰投与となる ・ 体温が異常に上がっているのにアラームが鳴らない これらはすべて、「ソフトの動きが原因で、患者が危ない状態になる」という共通点を持っている。だからこそ、「危険状態につながる原因を、ソフト側からあらかじめ探し出す」ことが重要になるのである。	

		これらの危険状態は、ソフトウェアの仕様、設計、実装、またはユーザーインターフェースの誤りから生じるものである。 ② ソフトウェアによる寄与の特定 その危険状態に対して、「ソフトウェアがどのように寄与しているのか」を分析する必要がある。例えば、 ・ データ取得ロジックの欠陥によって不正確な値を生成した ・ イベント処理の遅延により異常を検出できなかった ・ 条件分岐の誤りにより警告が抑制された ・ 操作ロジックが複雑で誤操作を誘発しやすい このような因果関係を明示的に特定し、修正・制御の対象となる機能を明確にする。 ③ リスクマネジメントファイルとの連携 分析結果は ISO 14971 に基づくリスクマネジメントファイルと連携させる必要がある。すなわち、 ・ 「このソフトウェアのこの機能が、〇〇というハザードに関与する」 ・ 「この構成要素は安全クラス C であり、致命的危害につながる」 といった形で記録し、今後のリスクコントロール手段の設計やテストにも反映させる。		
		（４）分析手法の例 実務においては、以下のような分析手法が用いられる。 ・ FTA（Fault Tree Analysis） トップに「危険状態」を置き、それが起こるまでのソフトウェア的な要因を論理的に下位に展開していく。 ・ FMEA（Failure Mode and Effects Analysis） ソフトウェアの機能ごとに、「どのような失敗が起こり得るか」「その結果どうなるか」を評価し、危険状態につながるモードを抽出する。 ・ 状態遷移図の安全性レビュー 状態の遷移によって「制御不能な状態」「無限ループ」「誤作動モード」等が発生しないかを確認する。	● 分析の進め方 IEC 62304 では、具体的なやり方までは決めていないが、実際には次のような方法がよく使われている。 ① ハザード分析（Hazard Analysis） 「どんな危険がありそうか？」を洗い出す方法 例：「アラームが鳴らないとしたら、どういう結果になる？」 ② フォールトツリー解析（FTA） 「危険状態が起きるには、どんな原因が積み重なったのか？」を逆からたどる方法 例：「薬の投与ミス → 指示ミス → 表示ミス → 計算間違い」 ③ FMEA（故障モード影響解析） ソフトの機能ごとに、「この部分が故障したらどうなるか？」を調べる方法	

			例：「温度表示ユニットが止まったら、どう影響するか？」 こうした手法を使って、ソフトウェアが引き起こす可能性のあるすべての危険状態を洗い出すことが大切である。	
		（５）安全クラスとの関係 この分析結果は、各ソフトウェア構成要素の「安全クラス」の見直しにも使われる。IEC 62304 では、構成要素ごとにクラス A（リスクなし）、クラス B（中程度のリスク）、クラス C（重篤な危害）を分類するが、危険状態分析の結果、クラスが引き上げられる場合もある。例えば、あるサブモジュールが「単なるデータ出力機能」としてクラス B とされていたが、誤出力が誤診を引き起こす可能性があると判明した場合、クラス C に再分類されることになる。		
		（６）分析の記録とトレーサビリティ 危険状態の分析結果は、以下の形式で記録され、他の文書（要求仕様、設計、テスト計画等）とトレーサビリティが保たれる必要がある。 危険状態の記述 | 寄与するソフトウェア機能 | 対応する構成要素 | 安全クラス | リスク ID 異常心拍数でアラームなし | 心拍計測処理＋アラーム出力処理 | AlarmModule | C | RSK-001 このような対応関係を明示することで、「どの危険状態にどのソフトウェアが関与し、それにどう対処したか」が説明可能となる。		
		（７）結論 細分箇条 7.1「危険状態を引き起こすソフトウェアの分析」は、ソフトウェアによる危害の可能性を正しく理解し、設計やリスク制御に反映させるための最初の、かつ最も重要なプロセスである。この分析が不十分であれば、リスクコントロールが的外れになり、製品の安全性が保証できなくなる恐れがある。単に「動くソフト」を作るだけでは、医療機器としての責任を果たしたことにはならない。危険な動作が起こり得る可能性を事前に見抜き、制御し、対策を施すことで初めて、「安全に使えるソフトウェア」を社会に提供することができる。その出発点が、この危険状態の分析である。	● まとめ 細分箇条 7.1「危険状態を引き起こすソフトウェアの分析」は、医療機器ソフトウェアが患者にとって危ない状態を作り出す可能性があるかを、あらかじめ予測・分析し、安全なソフト設計につなげるための第一歩である。この分析がしっかりできていれば、ソフトが原因となる事故はぐっと減らせる。だからこそ、「作ってから考える」のではなく、「作る前から考える」ことが、この箇条の一番大事な考え方である。例えば、たこ焼きを焼くときに、もし鉄板のコードが足に引っかかりやすい場所にあったとしたら、それは「危険状態」である。 ・ 火傷のおそれがある ・ 鉄板がひっくり返るかもしれない ・ 電源が切れて焼けなくなる	

			こうした危険があるなら、あらかじめそれに気づいて、対策（コードの位置を変える等）をするはずである。ソフトウェアも同じで、「何が危ないか」に気づければ、事故の多くは防げる。	
	7.2 リスクコントロール手段	（１）序論 細分箇条 7.2「リスクコントロール手段」は、細分箇条 7.1 で特定・分析されたソフトウェア由来の危険状態に対して、危害の発生を防止する、あるいはその発生確率や重篤度を低減するための「対策（制御手段）」を講じる工程である。ソフトウェアによって引き起こされるリスクは、物理的な機械故障とは異なり、コードの不具合や設計思想、ユーザーインターフェースの設計ミス、例外条件の見落とし等から生じる。これらに対しては、ソフトウェアならではの「論理的」な制御策を組み込むことで、リスクを効果的に制御することが可能である。IEC 62304 のこの条項では、そうした「ソフトウェアに適したリスク低減策」を開発プロセスの中で適切に設計・実装し、それが実際に危険を減らすものであることを明確にすることが求められている。	● はじめに 細分箇条 7.2「リスクコントロール手段」は、ソフトウェアに潜んでいる危険を減らすために、どのような対策を行うかを決めて、実際にその手段を組み込む作業のことである。例えば、あるソフトウェアが「高熱なのにアラームを出さない」という危険を持っていたとする。このままでは患者に重大な健康被害が出てしまうかもしれない。そこで、「高熱を検出したら必ずアラームを鳴らす機能を入れる」といった対策を講じる。これが「リスクコントロール手段」である。この工程は、前の箇条 7.1 で見つけた危険状態に対して、どうやって安全なソフトにするかを具体的に形にするステップである。	1） GB（※）：P165-P166 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）リスクコントロール手段とは何か リスクコントロール手段（Risk Control Measures）とは、リスクの大きさ（＝危害の重大性 × 発生確率）を受容可能なレベルにまで低減するために講じる具体的な対応策である。ソフトウェアにおけるリスクコントロール手段には、主に次の 3 つのタイプが存在する。 ① 設計によるリスク低減 ・ 安全側の設計（fail-safe 等） ・ 冗長性の確保（複数経路による異常検知） ・ ソフトウェアによる入力制限・自動補正 ・ 状態遷移の制限（不正遷移を防止） ② ユーザーインターフェースによる支援 ・ 誤操作を防ぐ入力チェック（例：確認ダイアログ） ・ 警告表示、アラート音等の通知 ・ 誤解を防ぐ明確な表示設計 ・ 操作フローの強制（例：手順のスキップ不可） ③ 外部的手段との組み合わせ ・ ハードウェア側でのバックアップ制御 ・ 手順書、使用説明書、教育資料による補完 ・ 外部のリスクコントロール（例：トレーニング、二重確認）	● 「リスク」と「リスクコントロール」とは？ まず言葉の意味をはっきりさせておこう。 ・ リスク：危険が起きる可能性と、その結果の重大さを合わせたもの。 ・ リスクコントロール：そのリスクを小さくするための方法。 つまり、リスクコントロールとは、「どうすれば危ないことを減らせるか？」を考え、実際にソフトに取り込む作業を意味する。 ● どんな手段があるのか？ リスクコントロールには、いくつかの方法がある。IEC 62304 では特定のやり方に限らず、ソフトウェアで可能な手段を柔軟に選べるようになっている。以下に代表的な例を示す。 ① 検出機能（モニタリング） ・ 異常な値（例えば心拍数、温度等）を自動で見つける ・ センサーからの入力が止まっていたら、すぐにエラー表示をする ② アラーム・警告機能 ・ 危険を検出したときに音や画面で警告を出す ・ 操作者に注意を促して、事故を防ぐ ③ フェイルセーフ機能	

<table>
<tr>
<td></td>
<td></td>
<td>本条項では、特にソフトウェア自体が実装するコントロール手段について重点的に設計・文書化することが求められている。</td>
<td>・　システムが誤動作した場合に、安全な状態に自動で移行する
例：処理を止めて、何も出力しないようにする

④ 入力制限・チェック
・　操作ミスが起こらないように入力項目に上限・下限を設ける
例：薬の投与量に最大値を設定しておく

⑤ ソフトウェアの二重チェック（冗長性）
・　同じ処理を 2 つの方法で実施して結果を比較し、差があれば警告する
例：血圧測定のアルゴリズムを 2 種類使う

このように、リスクの内容に応じて適切な手段を選び、それを設計の中にきちんと組み込むことが求められる。</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>（３）実装すべきリスクコントロールの条件
IEC 62304 において、リスクコントロール手段を設計・実装すべきかどうかは、基本的に以下の判断基準によって決定される。

・　危害の重大性が高い
・　発生確率が高い、または未知である
・　既存の制御策では不十分である
・　ソフトウェアによって制御可能である

これらに該当する場合は、必ずソフトウェアレベルでのリスクコントロール手段を講じ、それを設計文書に明示しなければならない。</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>（４）リスクコントロール手段の設計プロセス

① 対象リスクの明確化
例：「心拍数が一定値を超えたとき、アラームが鳴らない場合に危険がある」

② 適用可能な制御策の検討
・　センサー値の範囲チェック
・　アラーム起動条件の見直し
・　ユーザーへのアラート表示
・　二重監視による冗長化

③ 安全性とユーザビリティのバランス確認
・　誤報を防ぐためにしきい値の調整が必要か</td>
<td>● 手段を決めるときの考え方
リスクコントロール手段を選ぶときは、次のようなポイントを考慮する必要がある。

<table><tr><th>観点</th><th>内容</th></tr><tr><td>効果</td><td>その対策は、本当にリスクを減らせるか？</td></tr><tr><td>実現可能性</td><td>今のソフト設計で実装できるか？</td></tr><tr><td>操作性への影響</td><td>ユーザーが混乱しないか？操作が複雑にならないか？</td></tr><tr><td>他の部分への影響</td><td>その変更が、別の機能に悪影響を与えないか？</td></tr><tr><td>再発防止</td><td>同じタイプの問題が起きないような設計になっているか？</td></tr></table></td>
<td></td>
</tr>
</table>

<table>
<tr>
<td></td>
<td></td>
<td>・　アラームの頻度が高すぎて無視されるリスクがないか

④ 設計文書への反映とレビュー
・　詳細設計書に明記
・　安全性機能としてタグ付け
・　レビュー記録を残す

（設計例）
① アラーム機能の強化
・　通常のアラームに加えて、アラーム不作動時に作動する監視タイマを追加（ウォッチドッグ）
・　アラーム設定値を操作不能にし、認証レベルの高いユーザーのみ変更可能とする

② 投薬量制御機能の安全対策
・　投与量が一定値を超えた場合、処理を即座に停止し、画面に警告を表示
・　前回の設定値と比較し、急激な変化があれば確認ダイアログを表示

③ データ保存機能の冗長性
・　保存先を 2 箇所に分ける
・　書き込み後に CRC チェックを実行し、整合性を確認</td>
<td>つまり、ただ「警告を出す」だけではなく、本当に安全性が上がるかどうかを慎重に考えて選ぶ必要がある。例えば、たこ焼きを焼くときに、鉄板が熱くてやけどの危険があるとする。その場合、

・　「やけど注意」と張り紙を出す（警告）
・　手袋を用意する（対策）
・　誰でも触れないように柵をつける（安全設計）
・　自動的に電源を切る装置をつける（フェイルセーフ）

といった工夫が考えられる。これらすべてが「リスクコントロール手段」にあたる。ソフトウェアの世界でも同じであり、危ないとわかったら、どう対処するかを考えるのがこの工程である。</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>（５）リスクコントロールの文書化
IEC 62304 では、設計されたリスクコントロール手段が以下のように文書化されていることを求めている。

<table><tr><th>リスク ID</th><th>関連機能</th><th>制御手段</th><th>実装箇所</th><th>テスト項目とのリンク</th></tr><tr><td>RSK-001</td><td>心拍監視</td><td>アラーム監視＋2 重検出機構</td><td>AlarmModule</td><td>TC-Alarm-01</td></tr><tr><td>RSK-002</td><td>投薬設定</td><td>範囲制限＋確認ダイアログ</td><td>DoseManager</td><td>TC-Dose-04</td></tr></table>
このように、リスクとその制御策、対応モジュール、テストケースの間のトレーサビリティを保つことが重要である。</td>
<td>● コントロール手段の文書化
どんな対策を講じたかは、すべて記録に残さなければならない。IEC 62304 では、以下のような情報を文書にすることが求められている。

・　どの危険に対して、どんな対策を取ったか
・　その対策はどの機能やモジュールに関係しているか
・　どの設計・試験文書に反映されているか

これを「リスクコントロールのトレーサビリティ」と呼び、どこでどんな安全対策がされているかがあとからでも追えるようにすることが重要である。</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>（６）結論
細分箇条 7.2「リスクコントロール手段」は、ソフトウェアに潜むリスクを</td>
<td>● まとめ
細分箇条 7.2「リスクコントロール手段」は、医療機器ソフトウェアの中で</td>
<td></td>
</tr>
</table>

		具体的な設計によって制御可能にするための最も実践的かつ重要な工程である。リスクの存在を知るだけでは不十分であり、それに対して「どう対応したか」「それが有効か」「どのように管理されているか」を明確にできなければ、医療機器としての安全性は担保できない。ソフトウェアは設計された通りに動作する。ゆえに、リスクもまた設計された通りに制御しなければならない。その責任を果たすために、設計者は安全性機能を明確に設計し、それを証明できる形で記録・管理することが求められる。	見つかった危険を減らすために、どんな対策を設計・実装するかを決め、きちんと反映するためのステップである。この工程を丁寧に行うことで、ソフトは単に動くだけでなく、「安全に動く」ようになる。つまり、リスクコントロール手段こそが、ソフトウェアに「安全性」という命を吹き込む工程である。	
	7.3 リスクコントロール手段の検証	（１）序論 細分箇条 7.3「リスクコントロール手段の検証」は、ソフトウェアに実装されたリスク低減策が、実際に有効に機能しているかどうかを確認する工程である。設計者がどれほど精密にリスクコントロール手段を組み込んだとしても、それが「本当に正しく動作し、想定通りに危険を防いでいるか」は、試験や解析によって確かめなければならない。この工程の意義は、医療機器の「安全性を証明する」ことである。ユーザーや患者、さらには規制当局に対して、「この機器は安全である」と自信を持って言えるためには、その裏付けとなる明確な検証結果が不可欠である。本条項は、そうした信頼性の根拠を構築するために極めて重要な役割を果たす。	● はじめに 細分箇条 7.3「リスクコントロール手段の検証」は、ソフトウェアに組み込んだリスクを減らすための対策（リスクコントロール手段）が、本当に効果を発揮しているかどうかを確かめる作業である。前の段階（7.2）では、例えば「高温になったらアラームを鳴らす」といった対策を設計し、ソフトに組み込んだ。この 7.3 では、「本当に高温になったときにアラームが鳴るか？」「誤作動しないか？」といったことを実際の試験や確認を通じて検証する。つまり、リスクに対する「お守り」を作ったあとで、「そのお守りが本当に効くか」を試す工程だと考えればよい。	1） GB（※）: P167-P168 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）本条項の目的 IEC 62304 におけるリスクコントロール手段の検証の目的は以下の通りである。 ① 設計されたリスク制御機能が、実際に意図した通りに動作するかを確認する ② 危険状態に対して、制御機能が正しく起動し、被害を防げるかを評価する ③ 制御機能に欠陥や抜け漏れがないことを証明する ④ 検証結果を記録として残し、リリースや承認において活用できる状態にする これらは、単なる品質保証活動にとどまらず、「患者の命を守る」ための最終的な安全保障手段ともいえる。	● IEC 62304 が求める検証の内容 IEC 62304 では、以下のような確認を行うことを求めている。 ① コントロール手段がソフトに正しく組み込まれているか？ ・ 設計書どおりのコードが実装されているか ・ 入力条件と処理内容が合っているか ② コントロール手段が意図した効果を持っているか？ ・ 本当に危険を減らすことができているか ・ 誤作動（アラームの鳴りっぱなし等）がないか ③ テスト結果が証拠として残っているか？ ・ 実施したテストの内容、手順、結果を文書にまとめる ・ 「これで大丈夫」といえる理由が説明できるようにする このように、「本当に効いているか？」「ちゃんと作られているか？」「記録に残っているか？」を確認するのが、検証のポイントである。	
		（３）検証とは何か？	● なぜ検証が必要なのか？	

IEC 62304 でいう「検証（verification）」とは、一般的な意味での「確認」や「テスト」よりも、やや厳格な意味を持つ。それは、「設計に基づいて実装された機能が、仕様通りに動作していることを、客観的証拠をもって証明する行為」である。検証手段には、以下のような方法が含まれる。

・　試験（テスト）：実際にコードを動かして挙動を確認する
・　レビュー：コードや設計書を人間の目で読み、正しさを確認する
・　解析：ツールによる静的解析、データフロー解析等
・　シミュレーション：モデルや仮想環境で動作を再現し確認する

この中でも、特にリスクコントロールに対する検証では「試験」が中心となる。

いくら「リスクコントロール手段」を設計しても、それが実際に効いていなければ意味がない。例えば：

・　アラームを鳴らすはずが、条件を間違えて鳴らない
・　データチェックのプログラムがミスを見落としている
・　表示が一瞬すぎて、使う人が気づかない

といった事態が起こる可能性がある。だからこそ、「意図した通りにソフトが安全対策を実行しているか？」を第三者の目線で客観的に確認することが不可欠となる。

（４）検証の手順

IEC 62304 においては、リスクコントロール手段の検証を以下の手順で体系的に実施することが推奨されている。

① 検証対象の明確化

・　どのリスクに対して
・　どの制御策が実装されており
・　どのソフトウェア構成要素に含まれるか

を一覧化する。例えば、

リスク ID	制御策	実装場所	テスト項目 ID
RSK-001	アラーム 2 重監視	AlarmManager	TC-ALM-001
RSK-002	投薬値チェック	DoseManager	TC-DSE-003

② 検証手法の選定

リスクの重大性、実装手段に応じて、適切な検証手段（試験・レビュー・解析）を選定する。例えば、

・　致命的なリスクに関してはシステムレベルでのテストを実施
・　単純な入力制限であれば単体テストで確認
・　計算ロジックであれば数値シミュレーションによる解析を活用

③ テストケースの設計

● 検証の方法

検証には、次のような方法がある。ソフトの性質や対策の種類に応じて選ぶことができる。

① 試験（テスト）
・　実際に条件を与えて動かし、結果を確認する
・　例：体温を 42℃に設定 → 警報が鳴るか？

② レビュー（設計書やコードの読み合わせ）
・　設計文書と実装コードを比較して、ずれていないかチェックする
・　第三者がチェックすることでミスを見つけやすくなる

③ シミュレーション
・　実機ではなく仮想環境で動かして確認する
・　実験が難しいケース（例えば心停止時）に有効

④ 自動テスト
・　決められた条件で自動的にチェックする仕組みを使う
・　一貫性とスピードが強み

こうした方法を使って、「手段がちゃんと効いている」ことを科学的に証明する。例えば、たこ焼き屋を営むときに火を扱ううえで、その前に「消火器があるか？」「避難経路に問題はないか？」をチェックすることがある。このとき、

・　実際に消火器が使えるかを試す（テスト）

1）ISO 13485 の 7.3 項の設計開発に関する要求事項は、IEC 62304 だけでは完全に満たせない部分がある。以下に各要求事項と SaMD における具体的な対応例を示す。

○　7.3.6 設計開発の検証

IEC 62304 はソフトウェア検証活動を規定しているが、ISO 13485 の検証要求を満たすためには以下も必要である。

・　事前に定義した受入基準に対する検証
・　検証記録の維持

【SaMD の場合の具体例】

・　ソフトウェア要求事項に対する機能テスト
・　セキュリティ要件の検証
・　性能要件（処理速度、メモリ使用量等）の検証

○　7.3.7 設計開発バリデーション

IEC 62304 はバリデーションの詳細を規定していないため、ISO 13485 のバリデーション要求を満たすには以下が必要である。

【SaMD の場合の設計開発バリデーション方法例】

		検証は「根拠ある証明」が求められるため、曖昧なテストでは不十分である。以下のような要素を明確に含むことが必要である。 ・　入力条件：正常系・異常系・境界値等を網羅 ・　期待される出力や挙動：リスク制御機能が作動する条件と結果 ・　前提条件：使用状態、他モジュールの状態等 ・　合否基準：判断の客観性と再現性を確保 ④ テストの実施と記録 ・　実施日時、担当者 ・　使用ソフトウェアのバージョン、試験環境 ・　実際の出力（ログ、画面、アラーム等） ・　合否判定と理由 等、第三者が見ても結果を再現可能なレベルで記録する。 ⑤ レビューと承認 試験結果は、設計担当者や品質保証部門等によりレビューされ、合否と妥当性が承認される必要がある。特に安全クラス C の場合は、形式的な証明書が必要になることもある。 （実装例） ① 入力チェック機能の検証 ・　想定される最大値、最小値、空欄等の境界値入力を行い、正しく拒否されるか確認 ・　異常値を送信しても内部ロジックに影響を与えないか（例：例外処理が機能しているか） ② アラーム機能の検証 ・　閾値を超えた入力が与えられたとき、警報が起動されるか ・　複数の異常状態が同時に発生した場合でも、適切な順序・表示で通知されるか ・　警報が停止する条件が正しく動作するか	・　避難経路の図と現地を比べる（レビュー） ・　火災の想定をしてシミュレーションする（模擬訓練） こうした確認ができていれば、「ちゃんと安全対策できている」と証明できる。ソフトでも同じである。「安全な仕組みを作った」だけでなく、「その仕組みが効いている」ことを証明する検証が必要である。	・模擬臨床データを用いた評価 ・　ユーザビリティ評価（実際のユーザー環境での操作性確認） ・　臨床的判断の正確性評価 ・　インテグレーションテスト（他システムとの連携確認） ・　上市前の限定的フィールドテスト 〇　7.3.8 設計移管業務 SaMD における設計移管は有体物と異なるアプローチが必要である。 【SaMD の設計移管の具体例】 ・　開発環境から検証環境、そして本番環境へのコード移行手順 ・　配布用パッケージの作成プロセス ・　クラウド環境へのデプロイ手順と検証 ・　配布プラットフォーム（App Store 等）への登録プロセス ・　配布後のインストール確認プロセス
		（５）トレーサビリティと文書化 IEC 62304 では、リスクとその制御手段、さらに検証結果がトレーサブル（相互に追跡可能）であることを求めている。例えば、 ・　要求 → 設計 → 実装 → テスト → テスト結果	●　文書化（記録を残すこと）の重要性 IEC 62304 では、検証の内容をしっかり記録として残すことが義務となっている。これは、 ・　あとから誰かが確認できるようにするため	

		の流れがすべて明文化されており、それぞれが一意の ID で結び付けられていることが望ましい。これにより、問題発生時の原因調査や、規制当局への説明が迅速かつ確実に行える。	・　規制当局（厚生労働省等）の審査に使うため ・　万が一問題が起きたときに、原因をさかのぼるため である。記録には、以下のような情報が含まれるべきである。 ・　どのリスクに対する検証か？ ・　どのような方法で確認したか？ ・　結果はどうだったか？ ・　確認した人は誰か？ ・　日付やソフトのバージョン 　このように、単なる「確認しました」ではなく、「確認したことを証明できる」ようにすることが求められる。	
		（６）結論 　細分箇条 7.3「リスクコントロール手段の検証」は、設計された安全機能が確実に機能していることを、明確な根拠とともに証明する工程であり、IEC 62304 全体の中でも、製品の信頼性を客観的に保証する要となるプロセスである。ソフトウェアは設計通りにしか動作しない。したがって、リスク制御機能も意図した通りに働いているかどうかを、必ず検証し、記録し、説明できるようにしておくことが、製品の安全性を社会的に担保する条件である。	●　まとめ 　細分箇条 7.3「リスクコントロール手段の検証」は、ソフトウェアに組み込んだ安全対策が本当に効果的に働いているかを、具体的な方法で確かめる重要な工程である。この検証によって、「このソフトは人の命に関わる医療機器に使っても大丈夫」と言える根拠が得られる。つまり、安全性を証明する最後の関門がこの検証であり、それは技術と責任の両方が問われるステップである。	
	7.4 ソフトウェア更新のリスクマネジメント	（１）序論 　細分箇条 7.4「ソフトウェア更新のリスクマネジメント」は、すでに市場に出荷された医療機器ソフトウェアを更新する際に、新たなリスクが生じないように管理するための工程を定めたものである。ソフトウェア更新には、バグ修正や機能改善、新しい法規制への対応、セキュリティ強化等さまざまな目的があるが、それに伴って「新しい問題」や「既存機能への悪影響」が発生する可能性も否定できない。この条項では、ソフトウェア更新が製品の安全性や有効性に影響を与えないよう、更新に際してリスクを評価・管理し、それが安全であることを確認したうえで実施することを求めている。	●　はじめに 　細分箇条 7.4「ソフトウェア更新のリスクマネジメント」は、医療機器ソフトウェアに何らかの変更や更新（アップデート）が行われるときに、その変更が新しいリスクを生まないか、あるいは過去のリスク対策に影響を与えないかを確認するための工程である。例えば、ソフトの新しい機能を追加したり、バグを修正したりすると、一見よくなっているように見えるが、それによって別の不具合が起きることがある。こうしたことを防ぐために、変更や更新があるたびに「リスクの見直し」を行うのがこの工程の目的である。	1） GB（※）：P169-P170 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）本条項の目的 　7.4 の主な目的は、以下の 4 点に集約される。 ・　ソフトウェア更新が新たな危険状態やリスクを引き起こすことを防止すること ・　更新によって既存のリスクコントロール手段が損なわれないことを確認すること ・　更新が安全であることを文書で裏付け、第三者にも説明できるようにす	●　IEC 62304 が求めること 　IEC 62304 では、ソフトウェアを更新するときに次のようなことを行うことを求めている。 ①　変更内容の把握と記録 ・　どこを、なぜ、どう変更したのかを明確にする ・　それを記録として文書化する（例：変更管理票）	

		ること ・　適切な検証・承認プロセスを通じて、安全なリリースを保証すること 更新によって安全性が悪化するような状況は、製造者の信頼を損ねるだけでなく、法的責任にもつながる可能性がある。そのため、更新の実施には「開発と同等の慎重さ」が必要とされる。	② 変更による影響の評価（リスク分析） ・　その変更によって、新しい危険状態が生まれないか？ ・　過去に設定したリスク対策が無効になっていないか？ ③ 必要に応じたリスクコントロールの見直し ・　変更によってリスクが高まるなら、新しい対策を追加する ・　以前の対策が使えなくなるなら、他の方法に切り替える ④ テストと検証の実施 ・　変更した部分だけでなく、関係する周辺部分のテストも行う（回帰試験） ・　安全性が維持されていることを証明する このように、「変更したら終わり」ではなく、「変更によって安全性が保たれているかどうかを確認する」ことが、この細分箇条の中心である。	
		（３）ソフトウェア更新とは何か 本条項でいう「ソフトウェア更新」には、以下のような変更が含まれる。 ・　不具合修正（例：誤表示やバグの除去） ・　新機能の追加（例：データ出力形式の追加、対応機器の拡張） ・　使用環境への対応（例：OS アップグレード、通信仕様変更） ・　セキュリティ強化（例：脆弱性対応、暗号化方式の更新） ・　表示やユーザーインターフェースの調整 これらすべてが、新たなリスクの発生源となり得るため、変更の大小に関わらず、変更管理とリスク評価を省略してはならない。	● なぜソフトの更新にリスクがあるのか？ ソフトウェアは、見た目は変わらなくても、内部の動きがとても複雑である。一部を変えただけで、思いがけず他の場所に影響を与えることがよくある。例えば、 ・　表示画面のデザインを変更 → 操作者が使い方を間違える ・　計算方法を修正 → 別の機能の結果に誤差が出る ・　通信方法を変えた → データが正しく届かないことがある このように、ソフトの「部分的な変更」が、全体に新たなリスクをもたらすことがあるため、更新時には慎重な対応が必要である。例えば、たこ焼きを売っているときに、「ソースを変えたい」と考えたとする。たったそれだけでも、 ・　味が変わることでお客さんの反応がどうなるか？ ・　アレルギー成分が含まれていないか？ ・　新しいソースで材料の賞味期限が変わらないか？ といったさまざまな影響を考える必要がある。「ちょっとした変更」でも、まわりにたくさんの影響が出るかもしれないという視点が、ソフトウェアにも必要である。	
		（４）リスクマネジメントにおける基本ステップ	● 実際の進め方（例）	

IEC 62304 では、ソフトウェア更新時のリスクマネジメントを以下の流れで行うことを求めている。

① 変更の影響範囲を特定する
- どのファイル、どのモジュール、どの機能が変更されたか
- 他の構成要素との依存関係があるか
- ハードウェアや外部システムへの影響があるか

② 新たな危険状態の可能性を分析する
- 追加された機能によって、新たな誤動作や誤解が生じる恐れはないか
- 処理順序の変更によってタイミング異常が発生しないか
- セキュリティの変更が既存の認証制御と衝突しないか

③ 既存のリスクコントロール手段の影響評価
- 更新によって、これまで有効だった安全機能が無効化されていないか
- 制御ロジックが変更された場合、そのテストが正しく再実施されているか
- エラーハンドリングの対象が拡張された場合、それに対応した検証が行われているか

④ リスク受容判定
- 新たに検出されたリスクについて、ISO 14971 に基づき「受容可能かどうか」を評価
- 残留リスクがある場合は、ユーザーへの注意喚起や取扱説明書での明示が必要

⑤ 更新後ソフトウェアの検証と妥当性確認
- 影響範囲に応じてユニットテスト・結合テスト・システムテストを再実施
- 変更箇所だけでなく、その周辺や依存先への影響も考慮した回帰試験（リグレッションテスト）を行う

更新のリスクマネジメントは、以下のような流れで行う。

① 更新内容の確認
例：温度表示の単位を摂氏から華氏に変更

② 影響範囲の特定
例：表示部分だけでなく、アラーム機能も影響を受ける

③ 新しいリスクの分析
例：医師が単位を見間違えて判断ミスをする可能性

④ 対策の検討と実施
例：単位を強調表示し、切り替え時に警告を出す

⑤ テストと記録
例：すべての動作を試験して、問題がないことを確認。結果を記録に残す

このように、1 つの変更がどこまで影響するかを事前に予測し、安全が守られているかを確認するプロセスが求められる。

（５）更新の文書化と承認

ソフトウェア更新には、以下のような記録を整備しておくことが重要である。

文書種別	内容例
変更要求書	修正の理由、内容、背景、緊急度等

		<table><tr><td>変更設計書</td><td>実装箇所、ロジックの違い、影響分析結果</td></tr><tr><td>リスク評価書</td><td>新規リスクとその評価、制御策、残留リスク説明</td></tr><tr><td>テスト仕様書</td><td>新規テスト項目、再実施対象の既存項目、合否判定基準</td></tr><tr><td>リリースノート</td><td>更新内容の要約、ユーザー向けの注意点</td></tr></table> また、更新を正式に市場に反映させるためには、品質保証部門、設計責任者、場合によっては第三者（規制当局、Notified Body 等）の承認が必要となる。		
		（６）バージョン管理と配布管理 更新によって新たなソフトウェアバージョンが生成された場合は、その構成情報（ソースコード、ドキュメント、ライブラリ、ツールチェーン等）を正確に管理する必要がある。例えば、 ・　バージョン番号や識別コード（例：v2.1.3） ・　リリース日と適用製品モデル ・　前バージョンとの差分 ・　試験完了記録 さらに、更新されたソフトウェアをどの医療機器に、どのタイミングで、どのようにインストール・配布したかについても記録が求められる。これは、将来的な不具合対応や回収、監査において極めて重要な情報となる。	●　変更管理との連携 IEC 62304 では、更新のリスクマネジメントを行う際には、箇条 8（構成管理）や箇条 9（問題解決プロセス）との連携も重要であるとされている。具体的には、 ・　変更が行われたら構成台帳に記録 ・　問題として報告された変更なら、再発防止も同時に考える ・　修正されたソフトのバージョン番号や日付、対応履歴を明確にする このように、ソフトの更新の「歴史」をちゃんと記録に残しておくことも安全性の一部である。	
		（７）結論 細分箇条 7.4「ソフトウェア更新のリスクマネジメント」は、医療機器のライフサイクル全体において、最も見落とされがちである一方、最もリスクを伴う領域を管理するための極めて重要な工程である。更新は進化であるが、それは同時にリスクである。だからこそ、更新を実施する際には、開発時と同様、あるいはそれ以上の慎重さと、体系だったリスク評価・管理・文書化が求められる。医療機器として社会に安全を保証し続けるためには、「直す」だけでなく、「安全に直す」体制と思想が必要である。	●　まとめ 細分箇条 7.4「ソフトウェア更新のリスクマネジメント」は、ソフトを更新・変更するときにその変更によって新たなリスクが生まれていないか、安全性が損なわれていないかをしっかり評価・管理するためのプロセスである。「更新は安全の敵にも味方にもなり得る」。だからこそ、更新するたびに、初心に返ってソフトを見直す姿勢が、信頼できる医療ソフトを育てる鍵である。	

○　IEC 62304 の箇条８の概説

箇条	細分箇条	概　要　解　説	簡　易　解　説	リファレンス・ポイント
箇条 8 「ソフトウェア構成管理プロセス」		IEC 62304 の箇条 8「ソフトウェア構成管理プロセス」は、医療機器ソフトウェアの安全性と品質とを維持するために、ソフトウェアの構成（バージョン、構成要素、文書等）を一貫して識別・追跡・管理するためのルールと手順を定めたプロセスである。構成管理は、ソフトウェアがどのように構成され、どのような変更が加えられ、どのように記録されているかを正確に把握するために必要不可欠な活動である。ソフトウェアの開発および保守では、多数のソースコードファイル、設計書、テスト仕様書、リリース資料が使用される。それぞれにバージョンが存在し、しばしば並行して変更が発生する。このような複雑な環境の中で、どの時点で、どの構成要素が、どのような内容であったかを明確にしなければ、ソフトウェアの再現性や安全性の説明責任を果たすことはできない。構成管理プロセスは、主に以下の 3 つの細分箇条から構成されている。 8.1 構成識別 管理すべきソフトウェア構成要素（ソースコード、文書、ツール等）を明確に識別し、追跡可能な状態に置く。 8.2 変更管理 ソフトウェア構成に対して変更を加える際に、影響評価、承認、文書更新等を含む正式な手続きを実施する。 8.3 構成状態の記録 いつ、誰が、どのような変更を加えたのか、現在の構成はどうなっているのかを記録として保持する。 これらの構成管理は、一般的なソフトウェア開発のプロセスと同様である。このプロセスを通じて、開発者、保守担当者、品質保証者、規制当局のいずれからも「このソフトウェアは安全に構成されている」と説明可能な状態を保つことができる。医療機器の場合は、特に説明可能にしておくことが重要であり、それにより PDCA の QMS も実施可能になる。構成管理は、ソフトウェアライフサイクル全体における維持活動である。 記録は紙であっても良いし、電子的な記録でも良い。ただし、説明責任には、それが虚偽でないことや、意図してうっかりかに限らず改竄されていないかも説明できないといけない。	箇条 8「ソフトウェア構成管理プロセス」は、医療機器ソフトウェアがいつ、どこで、どのように変更されてきたかをきちんと把握・管理するための仕組みである。医療機器のソフトは、複雑で多くの部品（プログラムのファイルや設計図）が関係しており、これらがどのバージョンで、どの人が作ったものかがわからなくなると、大きな事故や混乱を生む可能性がある。市場に出した後に安全性に関する事象が発生した場合、その原因を徹底的に調査することが要求される。例えば、バグの原因を調べようとしても、どのファイルが本物か分からなければ対応できない。そこで、この「構成管理」という仕組みが必要となる。構成管理とは、以下のようなことを行う活動である。 ・　ソフトの部品を「これは何か？」と識別する（構成識別） ・　修正や変更を記録しておく（変更管理） ・　今の状態をいつでも確認できるようにする（構成状態の記録） つまり、ソフトウェアの「部品管理」「履歴管理」「状態管理」をまとめて行う仕組みである。構成管理は、例えば次のようなときに役立つ。 ・　バグが起きたときに、過去のバージョンを調べられる ・　誤って古いファイルで医療機器を動かすことを防げる ・　管理する組織によって承認されていない変更が行われていないかを確認できる 医療機器の安全性を保つためには、医療機器ソフトウェアについて「いつ・どこで・何を・誰が」変えたのかを明確にしておくことが不可欠であり、その役割を果たすのが構成管理プロセスである。	1）GB（※）：P172-P179 2）セキュリティについては、IEC 81005-1（箇条 8）において、「脆弱性」に着目した構成管理の必要性を要求している。 （※）GB：IEC 62304 実践ガイドブック（じほう）
	8.1 構成	（１）序論	●　はじめに	1）GB（※）：P173-P175

	識別	IEC 62304 の細分箇条 8.1「構成識別」は、医療機器ソフトウェアの開発および保守において、対象となるソフトウェア構成要素を明確に定義・識別し、それらがどのようなバージョン、関係性、依存性を持っているかを把握できるようにすることを目的としたプロセスである。ソフトウェアは、ソースコードだけで構成されているわけではない。設計書、要求仕様書、テスト手順書、コンパイラ、ツール、外部ライブラリ等、多数の要素が組み合わさって１つの製品として成立しているため、これらはすべて構成要素となりうる（構成要素をどの単位で区切るかの明確な基準はないが、ソフトウェアを管理する上で必要な単位で構成要素を区切る必要はある）。構成要素をそれぞれ一意に識別し、管理下に置くことで、開発・保守・更新の過程での混乱を防ぎ、品質の一貫性と安全性の保証を実現する。	細分箇条 8.1「構成識別」は、医療機器のソフトウェアに使われるさまざまなファイルや情報（＝構成要素）をきちんと見分けられるようにしておくためのルールである。簡単に言えば、「このファイルは何のためのもので、どのバージョンで、誰が使っているか」をはっきりとわかるようにしておくことが目的である。ソフトウェアはプログラムだけでなく、設計書、試験記録、マニュアル等、いろいろな「部品」でできている。そして、これらが増えたり、修正されたりすると、どれが最新版なのか？間違ったものを使っていないか？といった混乱が生じることがある。構成識別は、こうした問題を防ぎ、「正しい部品を、正しく管理する」ための土台となる仕組みである。	（※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）構成識別の目的 構成識別が求められる理由は、次のような具体的な課題に対応するためである。 ① どのバージョンの何が使われているかを明確にするため 例えば、v2.1 の心拍モニタはどのコードと仕様書に基づいているのかを即座に把握できるようにする。 ② 不具合の原因追跡を可能にするため 問題が発生した際、当該バージョンの構成情報から原因特定を迅速に行う。 ③ 安全性の証明根拠を確保するため 規制当局や品質監査に対し、「このバージョンはこの設計・試験に基づいてリリースされた」と説明可能にする。 ④ 変更管理との連携を取るため 修正対象がどのバージョンのどのファイルなのか、管理上の混乱を避ける。		
		（３）構成識別の対象（構成要素） 構成識別の対象となる項目としては、以下のようなものが挙げられる。 ・ ソースコードファイル（例：.c, .cpp, .py 等） ・ ヘッダファイル・ライブラリファイル（例：.h, .dll） ・ ソフトウェア要求仕様書（SRS） ・ ソフトウェア設計文書（SDS） ・ テスト仕様書、テスト結果レポート ・ リスクマネジメントファイル ・ コンパイラやビルドスクリプト、使用ツールのバージョン	● 「構成」とは何か？ 構成とは、ソフトウェアを構成している「部品」のことを意味する。例えば以下のようなものがある。 ・ ソースコード（プログラムの元になるファイル） ・ 実行ファイル（コンピュータが動かせる形のファイル） ・ 設計書、仕様書 ・ テスト仕様書とその結果 ・ 使用説明書やマニュアル ・ バージョン情報	

		・　外部ベンダー製のソフトウェアやオープンソースコンポーネント これらを「構成管理対象」として、バージョン番号や識別子を付与して管理下に置くことが求められる。	これらはすべて、ソフトウェア製品の一部であり、１つでも間違ったものが含まれていれば、誤作動や事故につながるおそれがある。だからこそ、これらを１つずつ正しく識別しておく必要がある。 ●　「識別」とはどういうことか？ 「識別」とは、「どの部品が何であるかを、他と区別できるようにすること」である。例えば以下のようなことが求められる。 ・　それぞれのファイルに名前とバージョン番号をつける ・　いつ作成されたか、誰が作ったかを記録に残す ・　そのファイルがどの製品のどのバージョンに使われているかをひと目でわかるようにする このようにすることで、間違ったバージョンを使ってしまうことを防ぎ、誰が何を扱っているのかをはっきりさせることができる。	
		（４）識別方法 構成要素を一意に識別するためには、以下のような方法が用いられる。 ① バージョン番号 各構成要素にバージョン番号（例：v1.0.0）を付けて管理する。変更履歴や機能追加があるたびに更新される。 ② 一意の識別コード 構成要素に対して一意な ID を付与する（例：SRS-001、MOD-ALM-002 等）。 ③ リリースタグやスナップショット すべての構成要素をある時点で凍結し、リリースパッケージとしてタグを付けて保存する（例：Release_2025-05-01）。 ④ ファイル名＋日付＋チェックサム ファイルの内容確認や改ざん検出のために、ファイル名と作成日、MD5 や SHA-256 といったハッシュ値を記録する。		
		（５）構成識別の例 構成識別の成果物の例は以下の通り。		

構成 ID	構成名	種別	バージョン	所属モジュール	最終更新日
MOD-ALM-001	アラーム制御モジュール	ソースコード	v1.2.3	AlarmModule	2025-04-12
SRS-ALM-002	アラーム要求仕様書	文書	v1.1	AlarmModule	2025-04-10
TST-ALM-003	アラームテスト仕様書	テスト文書	v1.1	AlarmModule	2025-04-15

　このようなリストをもとに、各リリースや変更の際に構成要素を参照し、一貫性を保証する。

（６）トレーサビリティとの関係

　構成識別は、他のソフトウェアライフサイクル工程（要求、設計、実装、テスト）との間にトレーサビリティを確保するためにも不可欠である。例えば、

　　要求 SRS-ALM-002 に基づき、MOD-ALM-001 が実装され、
　　それを TST-ALM-003 が検証する。

　という関係性を識別子ベースで明確に結び付けることで、品質の確保だけでなく、問題発生時の調査や影響分析を迅速に行えるようになる。

●　なぜそれほど重要なのか？

　ソフトウェアは目に見えず、触れないものである。紙の製品ならラベルを見ればどれがどれか分かるが、ソフトウェアはそうはいかない。しかも、少しの違いで動作が大きく変わる。そのため、「どれが正しいか」「どれが最新か」が明確でなければ、

・　過去のバグが再発する
・　承認されていないプログラムが使われる
・　誤って古いファイルでテストしてしまう

　等の深刻な問題を引き起こす。構成識別は、それらを防ぐための「見えない製品管理ラベル」のような役割を果たす。例えば、演劇をやるとき、照明係・音響係・脚本係がそれぞれ資料や道具を用意していたとしよう。何種類もある資料や道具の中で、

・　「これは最新のシナリオか？」
・　「この音響は 1 日目用？2 日目用？」
・　「このスイッチは予備？本番用？」

　等が分からなくなると、間違った準備や演出につながり、全体がぐちゃぐちゃになってしまう。これを防ぐには、名前やタグをつけて管理すること（いわゆる「構成識別」）が大切である。ソフトウェアの世界も、まったく同じである。

（７）結論

　細分箇条 8.1「構成識別」は、ソフトウェア製品の品質と安全性を一貫して

●　まとめ

　細分箇条 8.1「構成識別」は、医療機器ソフトウェアに含まれるすべての部

		保証するための基盤となる活動である。構成要素を明確に識別し、それを誰が見ても一意に追跡できるようにしておくことは、ソフトウェアが「設計通りに作られ、設計通りに検証された」ことを示す客観的な証拠にもなる。構成管理がなされていなければ、不具合の原因究明は困難となり、再発防止も不可能となる。逆に言えば、構成が明確であればこそ、問題が起きても迅速かつ正確に対応ができる。その意味で、構成識別は、開発・保守の全工程を支える情報基盤となる。	品（ファイルや文書）について、どれが何かを見分けられるようにし、安全で確実な管理を実現するための基本的なルールである。この工程がしっかりしていれば、どんなに複雑なソフトウェアでも、「今どんな状態で、どこが使われているか」がひと目で分かる。まさに「安全なソフトを支える名札管理」とも言える工程である。	
	8.2 変更管理	（１）序論 細分箇条 8.2「変更管理」は、医療機器ソフトウェアの構成要素に対して何らかの変更を加える際に、その変更が安全性や有効性に悪影響を与えないよう、計画的かつ記録的に実施することを目的としたプロセスである。ソフトウェアはその特性上、たった１行のコード変更でもシステム全体に影響を及ぼすことがある。そのため、変更は「実装」よりも「評価」と「管理」が重要視される。変更管理とは、単なる作業手順ではなく、リスクの再評価、関係文書の更新、影響範囲の分析、承認フローの確立、記録の保管といった一連の活動から構成されており、製品の安全性と一貫性を守るための防波堤ともいえる存在である。	● はじめに 細分箇条 8.2「変更管理」は、医療機器ソフトウェアに対して何かを変更するとき、その変更を安全かつ確実に管理するためのルールである。ソフトウェアの世界では、「ちょっと修正するだけ」という軽い気持ちでコードや設定を変更することがある。しかし、医療機器のように人の命に関わるソフトでは、それが大事故につながることもある。例えば、アラーム機能に関するコードを変えたら、患者の異常に気づけなくなった、等ということが実際に起こりうる。だからこそ、「何かを変える」前には、その理由・影響・方法をしっかり考え、記録し、確認してから行うことが必要になる。これが「変更管理」の考え方である。	1） GB：P176-P178 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）変更管理の目的 IEC 62304 における変更管理は、以下のような目的を果たすものである。 ・ 変更によって新たなリスクが発生しないことを確認する ・ 既存のリスクコントロール策が維持されていることを保証する ・ 変更の内容、理由、影響範囲が明確であるようにする ・ 関係者全員の承認を経て、計画的に変更を適用する ・ 将来的に変更履歴を追跡・証明可能にしておく このように、変更管理は「何を変えたか」よりも「どうやって変えたか」に重点を置いた活動である。	● 変更管理の目的とは？ 変更管理の目的は、ソフトに手を加えるなら、「なぜ変えるのか」「どう変えるのか」「変えた結果どうなったのか」をすべて把握しておかなければならないということになる。 ● どんなときに変更管理が必要か？ 変更管理が必要な場面はたくさんある。例えば、 ・ ソフトのバグ（不具合）を修正するとき ・ 医師や看護師からの要望で機能を追加・変更するとき ・ 関連法規の変更に合わせて表示を修正するとき ・ セキュリティを強化するため、通信の仕組みを変えるとき これらはすべて「変更」にあたり、そのたびに変更管理の手順に従って進めなければならない。	
		（３）変更の対象となる項目 ソフトウェアのコードそのものだけでなく、以下のような幅広い構成要素が対象となりうる。 ・ ソースコードの修正、追加、削除		

		・　ソフトウェア要求仕様書（SRS）の改訂 ・　設計仕様書やアーキテクチャ文書の変更 ・　テスト仕様・テスト手順の更新 ・　使用される開発ツールやライブラリのバージョンアップ ・　記録文書、リリースノートの変更 ・　構成識別子、ファイル名、格納先の変更等		
		（４）変更管理のプロセス 　IEC 62304 では、変更を以下のような手順に従って管理することを求めている。 ①　変更要求の提出 ・　誰が、何の目的で変更を提案したのかを記録する ・　変更要求には、変更理由、対象構成、目的、想定される効果を記載する ②　影響範囲の分析 ・　変更によって影響を受ける構成要素、仕様、機能を技術的に洗い出す ・　テストケース、リスクマネジメントファイルへの影響も検討する ・　他のモジュールや製品、バージョンへの波及効果も分析 ③　リスク評価 ・　変更が新たな危険状態を引き起こす可能性がないかを評価する ・　既存のリスクコントロール策が引き続き有効であるかを確認する ・　必要であればリスクマネジメントファイルの更新を行う ④　承認 ・　技術責任者、品質保証部門、場合によっては経営陣の承認を得る ・　承認プロセスは記録され、変更の実施条件とともに文書化される ⑤　変更の実施 ・　コーディング、文書改訂、構成更新等を手順に従って実施 ・　同時に、実施記録（誰がいつ何を変更したか）を保存 ⑥　検証・試験 ・　ユニットテスト、結合テスト、システムテスト等で変更の正当性を確認 ・　既存の機能に悪影響を与えていないことを回帰試験で確認 ⑦　完了・記録保存 ・　変更内容、試験結果、承認情報を一元管理システムに登録		

<table>
<tr><td></td><td></td><td>・　変更完了後、対応するリリースへの反映と通知を行う</td><td></td><td></td></tr>
<tr><td></td><td></td><td>（５）変更管理文書の例

<table><tr><th>項目</th><th>内容例</th></tr><tr><td>変更 ID</td><td>CHG-ALM-20250401</td></tr><tr><td>提出日</td><td>2025 年 4 月 1 日</td></tr><tr><td>提案者</td><td>医療システム開発部 第 2 課 大谷翔平</td></tr><tr><td>変更内容</td><td>アラーム作動閾値の調整</td></tr><tr><td>影響範囲</td><td>AlarmModule, SRS-ALM-002, TC-ALM-001</td></tr><tr><td>リスク評価結果</td><td>新たなリスクなし／残留リスクに変更なし</td></tr><tr><td>承認者・日付</td><td>品質保証部長／2025 年 4 月 2 日</td></tr><tr><td>試験結果</td><td>TC-ALM-001, TC-ALM-004 合格</td></tr><tr><td>リリース ID</td><td>Release_2025_04_03</td></tr></table></td><td></td><td></td></tr>
<tr><td></td><td></td><td>（６）結論
細分箇条 8.2「変更管理」は、医療機器ソフトウェアの品質と安全性を維持するために不可欠なプロセスである。変更そのものよりも、変更を「どう扱ったか」「どう証明できるか」が重要であり、透明性と追跡可能性のある変更こそが、安全性を裏付ける強固な基盤となる。変更管理は、信頼の履歴である。信頼される製品とは、確実な記録と慎重な手続きの積み重ねによって成立する。</td><td>●　まとめ
細分箇条 8.2「変更管理」は、医療機器ソフトウェアに対して何かを変更するとき、その変更を正しく評価し、記録し、安全に反映するための管理手順である。変更とは、便利になることもあれば、危険を増やすこともある。だからこそ、「変更はリスク」と考え、それに備える仕組みが必要である。例えば、演劇をやっているときに、「セリフの一部を変更しよう」となったとする。そのとき、

・　他のセリフとの流れは合っているか？
・　演出に影響しないか？
・　他の役の人に連絡したか？
・　台本を最新版に更新したか？

等をしっかり確認してから変えるべきである。勝手に変えると、他の人が混乱したり、事故につながる。ソフトウェアでもまったく同じであり、小さな変更にこそ、慎重な手続きが必要である。</td><td></td></tr>
<tr><td></td><td>8.3 構成状態の記録</td><td>（１）序論
細分箇条 8.3「構成状態の記録」は、医療機器ソフトウェアの安全性と一貫性を維持するために、ソフトウェア構成の状態（どのバージョンが、どの内容</td><td>●　はじめに
細分箇条 8.3「構成状態の記録」は、医療機器ソフトウェアの「今の状態がどうなっているか」を、いつでも正確に把握できるように記録しておくことを</td><td>1） GB：P179

（※）GB：IEC 62304 実践ガイドブック</td></tr>
</table>

		で、誰によって、いつ作成・変更・承認されたか）を正確に記録・保存し、いつでも追跡可能な状態にしておくことを目的としたプロセスである。ソフトウェアは変更や更新が頻繁に発生する対象であり、特に医療機器では一つひとつの変更が患者の生命や治療効果に影響を与える可能性がある。したがって、過去・現在・将来にわたって、「どの構成要素がどのような状態で存在していたか」を明確に記録し、必要に応じて証明できることが不可欠である。このプロセスは、「いつ、何を、誰が、なぜ、どう変えたか」を証明可能にするためのものであり、品質保証と規制対応の中核となりうる活動である。	求める規定である。例えば、あるソフトのバージョンが「Ver.1.2.3」であるとき、その中にはどんなプログラムが含まれていて、どの設計書が使われていて、どんな試験が行われたか、それらすべてをはっきりと記録に残しておくことが「構成状態の記録」である。この記録がなければ、あとで問題が起きたときに「何を使っていたのか」がわからず、原因の特定もできず、再現もできない。そのため、ソフトウェアの「今の姿」を記録しておくことが、非常に大切になる。	（じほう）
		（２）構成状態とは何か IEC 62304 における「構成状態」とは、構成要素（ソースコード、設計書、テスト仕様書等）それぞれの以下の情報を含む状態を指す。 ・ バージョン番号（例：v1.0.3） ・ ファイル名や識別子（例：MOD-ALM-003） ・ 状態（例：作成中、承認済み、リリース済み、廃止済み） ・ 作成者、承認者、修正者 ・ 修正日、承認日、リリース日 ・ 適用された変更要求番号（例：CHG-20250401） ・ 関連リスクやテストとのリンク情報 これらの情報を「構成状態記録」として保存・更新していくことが求められる。	● 「構成状態」の一例 構成状態とは、ある時点におけるソフトウェアの中身（構成）がどうなっているかを示す情報のことである。例えば、「2025 年 5 月 10 日リリースの心電図ソフト Ver.1.4」には： ・ ソースコード：ecg_main_v1.4.c ・ 設計書：ECG_Design_v1.3.pdf ・ テスト結果：ECG_TestReport_20250509.xlsx 等が含まれていたとする。これらをひとまとめにして「これがこのバージョンの構成です」と記録することが、構成状態の記録である。	
		（３）構成状態の記録の目的 構成状態の記録は、次のような目的を果たす。 ① 構成の一貫性と整合性を維持するため すべての構成要素が正しいバージョンで一致しているか確認可能にする。 ② トレーサビリティ（追跡可能性）を確保するため 不具合発生時に、どの構成が関与していたかをすぐに特定できるようにする。 ③ 品質監査や規制当局対応を可能にするため 医療機器の承認申請や品質監査において「設計通りに作られている」ことを証明する。 ④ 変更履歴と構成の復元を可能にするため	● 「構成状態の記録」を求めるタイミング ① すべての構成品目が把握できること ・ ソフトウェアアイテム（モジュールや機能） ・ ソフトウェアユニット（小さな部品） ・ 文書類（設計書、試験手順書、マニュアル等） これらが「どのバージョンのソフト」に含まれていたのかを明確にしておく。 ② 構成状態を一意に識別できること ・ 「どの製品に、どの構成が含まれているか」 ・ 「その構成は、どのタイミングで、誰が承認したか」 を識別番号や日付等で管理する。	

		ある時点の構成にさかのぼって製品を再現（リプロダクション）できるようにする。	③ 状態が変わったら記録を更新すること ・ 例えば、Ver.1.4 から Ver.1.5 になったとき ・ 修正・追加・削除があったとき ・ 試験結果が更新されたとき 等、構成が変わったタイミングで、記録も必ず更新することが必要である。 ● なぜ必要なのか？ 構成状態の記録がないと、以下のような問題が起きうる。 ・ バグの原因を調べても、どのファイルで発生したか分からない ・ 修正しようにも、どの設計に基づいて作られたソフトか分からない ・ 誰が、いつ、どのファイルを使っていたかが不明で、責任があいまいになる 医療機器は命に関わる機器であるため、こうした「うやむや」は絶対に許されない。正確な記録があることで、再現、修正、責任追跡が可能になる。	
		（４）構成状態記録に含まれる情報 構成状態記録の例として、以下のような情報が含まれる。 **項目** / **説明例** 構成 ID / MOD-ALM-003 種別 / ソースコード／要求文書／テスト文書等 バージョン / v1.2.1 状態 / 作成中／レビュー中／承認済み／廃止／リリース済み 作成者／作成日 / tanaka／2025 年 4 月 10 日 修正者／修正日 / yamamoto／2025 年 4 月 15 日 承認者／承認日 / qasuzuki／2025 年 4 月 16 日 変更要求番号 / CHG-ALM-20250401 関連リスク ID / RSK-ALM-001 関連テストケース ID / TC-ALM-005		

		（５）他の工程との関係 構成状態の記録は、他の工程と深く関わっている。 ・　変更管理（8.2）：変更があれば、構成状態も更新される ・　リスクマネジメント（7 章）：安全性に関わる部品の変更は特に注意して記録する ・　保守プロセス（6 章）：不具合があった場合、どの状態で発生したかを調査するために構成記録が役立つ		
		（６）結論 細分箇条 8.3「構成状態の記録」は、医療機器ソフトウェアの品質と安全性の「記録的保証」を支える要である。変更があるたびに正確な状態を記録し、将来にわたって再現性と説明責任を果たすためには、常に「今この瞬間の構成はどうなっているか」が即答できる管理体制が必要である。状態の記録は単なる事務作業ではなく、「この製品は安全に作られ、安全に保たれている」ことを社会と法規制に対して証明するための、最も実用的な証拠となりうる。	●　まとめ 細分箇条 8.3「構成状態の記録」は、医療機器ソフトウェアに含まれるすべての構成要素が、どのバージョンで、どのように使われていたかを明確に記録し、いつでも確認できるようにする工程である。これは、「見えないソフトウェアの中身を『見える化』するための方法」であり、ミスや事故を未然に防ぐ強力な仕組みとなる。つまり、構成状態の記録は、安全で信頼できるソフトづくりを支える背骨のような存在になる。例えば、パンフレットを作るとき、 表紙デザイン：version 2 タイトルロゴ：version 1 原稿：version 3 等のように、それぞれのバージョンが混ざっていたら、最終版がどれなのか分からず、間違った組み合わせで印刷してしまうかもしれない。これを防ぐには、「このバージョンには、どのファイルが含まれているか」を一覧で記録しておくことが必要である。ソフトウェアも同じで、完成形を正しく記録しておくことが、信頼と安全につながる。	

○　IEC 62304 の箇条９の概説

箇条	細分箇条	概　要　解　説	簡　易　解　説	リファレンス・ポイント
箇条 9 「ソフトウェア問題解決プロセス」		IEC 62304 の箇条 9「ソフトウェア問題解決プロセス」は、医療機器ソフトウェアに関して発生した問題に対して、組織的・一貫的・追跡可能な方法で対応するための枠組みを定めたものである。ここでいう「問題（problem)」とは、ソフトウェアの不具合、設計上の誤り、仕様との不一致、使用者からの苦情、あるいは製品安全性にかかわる異常挙動等、幅広い現象を含む。本プロセスの主目的は、問題を単に「修正する」ことにとどまらず、その発見→分析→対応→再発防止→記録保持→有効性検証までを一連の流れとして体系化することで、製品の安全性・信頼性を持続的に保証することである。とりわけ医療機器においては、些細なソフトウェア不具合であっても、診断ミスや治療エラーに直結する可能性があるため、問題対応プロセス自体が安全性の根幹を支える仕組みとなる。IEC 62304 の箇条 9 は、以下の 8 つの細分箇条に分かれており、それぞれが問題解決の段階を順にカバーしている。 9.1 問題報告の作成 問題が発見された際に、記録・分類・初期情報を整理して報告書を作成する。 9.2 問題の調査 報告された問題の原因、影響範囲、再現性を分析し、対応の要否を評価する。 9.3 関係者への通知 重要度に応じて、関係部門や第三者機関へ情報を速やかに共有する。 9.4 変更管理プロセスの使用 必要な修正は、正規の変更管理プロセスに則って実施される。 9.5 記録の保持 すべての問題対応について、発見から完了までの記録を保持・管理する。 9.6 問題の傾向分析 発生した問題を蓄積し、再発防止や設計改善に役立てる。 9.7 ソフトウェア問題解決の検証 対応策が実際に有効であったことを試験や評価で確認する。	箇条 9「ソフトウェア問題解決プロセス」は、医療機器に使われるソフトウェアで問題（バグ、不具合、誤作動等）が見つかったときに、それをどう扱い、安全に解決していくかを決めたルールである。ソフトウェアはとても複雑な仕組みでできており、開発中や使用中に予想しない問題が起きることがある。例えば、表示が消えてしまったり、計算結果がおかしかったり、操作しても反応しないことがある。医療機器の場合、こうした問題が患者の安全に大きく関わることがあるため、あわてず確実に対応できる仕組みが必要である。この「問題解決プロセス」では、問題が起きたときに次のようなことを順番に行う。 ・　問題の報告を記録に残す ・　その内容を詳しく調べる ・　関係者に知らせる ・　必要な修正を決めて変更管理を使って対応する ・　結果を記録に残す ・　同じような問題が繰り返されていないかを分析する ・　修正したことを確認する ・　試験記録をまとめておく このように、「発見 → 調査 → 修正 → 確認 →記録」までの一連の流れを決めたのが、この箇条の内容である。	1) GB（※）: P182-P186 2) セキュリティについては、IEC 81005-1（箇条 9）において、「脆弱性」に着目した問題解決プロセスを要求している。 （※）GB : IEC 62304 実践ガイドブック（じほう）

		9.8 試験文書の内容 　試験内容、合否判定、トレーサビリティを含む文書化要件を定める。 　このプロセスは、開発者のみならず、保守担当者、品質保証部門、規制当局等、ソフトウェアのライフサイクルに関与するすべての関係者にとって、製品の安全性を担保する監視と対応の柱になりうる。		
	9.1 問題報告の作成	（１）序論 　細分箇条 9.1「問題報告の作成」は、医療機器ソフトウェアに問題が発見された際、その現象を正確かつ網羅的に記録し、適切な対処へとつなげるための出発点である。問題解決プロセスの第一歩として、誰が、いつ、どこで、何を、どのように発見したかを明確に記録することにより、その後の調査・分析・修正・再発防止が円滑に進む。「問題報告」とは、ソフトウェアの不具合や異常な挙動、ユーザーからの苦情、設計仕様との不整合、テストでの失敗事例等を文書化したものである。単なる「バグ報告」ではなく、安全性・品質・信頼性にかかわる情報を統合的に整理したものとして扱われる。	●　はじめに 　細分箇条 9.1「問題報告の作成」は、医療機器のソフトウェアで問題（バグ、不具合、動作ミス等）が見つかったときに、それを正しく記録に残すための作業を指す。ここでいう「問題報告」とは、「いつ、どこで、誰が、どんな問題を見つけたのか」を整理して書き出す記録であり、この第一歩が、後の調査や修正の出発点となる。例えば、心電図表示ソフトが特定の条件で画面がフリーズするという問題が起きたとき、それをただ「止まった」と伝えるだけでは不十分である。どういう状況で止まったのか？何をしていたときか？ソフトのバージョンは？といった情報を正しく残すことで、あとで問題を再現・分析できるようになる。それが「問題報告の作成」という作業である。	1） GB（※）：P183 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）問題報告の目的 　問題報告書の作成は、以下のような目的を果たす。 ①　事実に基づく記録を残すこと 　時間が経過しても、誰もが同じ情報に基づいて対応できるようにする。 ②　再現性と再調査の基盤を作ること 　「何が問題だったのか」を正確に記録することで、他の開発者や品質保証者が検証可能になる。 ③　影響範囲の評価につなげること 　同様のモジュール、バージョン、製品にも同じ問題がないかを検討する際の起点となる。 ④　透明性と説明責任を確保すること 　将来の監査や規制当局からの問合せに対し、「そのとき、どのように対処したか」を示すための証拠とする。	●　なぜ問題報告が大切なのか？ 　医療機器のソフトは、人の命や健康に関わるものである。だからこそ、「どんな問題が起きたのか」をすぐに把握し、対策をとる必要がある。だが、そのためにはまず、正しい情報が必要である。情報が不十分だったり、あいまいだったりすれば、原因の特定が難しくなり、修正も遅れてしまう。例えば、 ・　「画面が変になった」とだけ書かれていても、どの画面かが分からない ・　「患者データが消えた」とあっても、どの操作で消えたのかが不明 ・　「エラーが出た」とあるが、エラーメッセージの内容が記録されていない 　こうした不十分な報告では、せっかく問題を見つけても、その後の対応が進まなくなってしまう。だからこそ、「問題を報告すること」は単なる作業ではなく、ソフトの品質を守るための大切な責任ある行動である。例えば、演劇のリハーサルをしているとき、照明がうまく動かないトラブルがあったとしよう。このとき、「照明が変だった」とだけ言うのでは、原因が分からない。 ・　どのシーンで？ ・　誰がスイッチを押した？ ・　何度目の通しだった？	

			・　他の電気は動いていた？ 　等、詳しい情報をメモしておけば、照明係がすぐに問題を調べて対処できる。ソフトウェアの世界も同じであり、「詳しい問題の報告」が、その後のすべての対応の出発点になる。	
		（３）「問題」とは何か 　以下のような現象が「問題」としてあげられる。 ・　システムクラッシュ、ハング、異常終了等の致命的な障害 ・　画面表示の誤り、データ保存失敗、計算ミス等の機能的異常 ・　ユーザー操作に対する不適切な反応、UI 上の誤解を招く表記 ・　ソフトウェア要求仕様や設計仕様との不一致 ・　過去に存在しなかったバグの新規出現（リグレッション） ・　サイバーセキュリティ上の脆弱性の発見 ・　市場からの苦情や医療現場でのトラブル報告 これらは重大・軽微を問わず、記録される必要がある。		
		（４）問題報告書に含まれるべき情報 　問題報告書には、以下のような項目を含める。 **項目** / **内容** 問題報告 ID / 一意に識別される番号（例：PR-20250510-001） 報告日 / 最初に問題が記録された日付 報告者 / 氏名、所属、連絡先 製品名／ソフトウェア名 / 問題が発生した対象 ソフトウェアバージョン / 該当するバージョン番号（例：v2.3.1） 問題の分類 / 機能的問題／表示問題／性能低下／UI 不具合／セキュリティ等 問題の概要 / 起きたこと、期待される動作、実際の動作、再現手順等 添付資料 / スクリーンショット、ログファイル、試験結果等 緊急度／影響度 / 使用中止が必要か、臨床リスクがあるか、応急処置が必要か等		

		<table><tr><td>関連するリスク ID</td><td>ISO 14971 に基づくリスクとのリンク（可能であれば）</td></tr></table>		
		（５）報告と管理体制 報告の受付・分類・一次確認を行う責任者や部門が定められており、未分類の報告が放置されることのないよう、ワークフローが整備されている必要がある。医療機器の場合、外部からの苦情（例：医療機関やユーザーからの電話やメール）も問題報告の起点となる。そのため、苦情管理プロセスと連携した問題受付体制が構築しておく必要も考慮する。		
		（６）結論 細分箇条 9.1「問題報告の作成」は、ソフトウェアに関連する安全性上の異常や不具合を「見える化」するための最初のステップであり、製品の信頼性と将来の安全性対策を築く基盤である。たとえ小さな異常でも、その情報が正しく記録されなければ、重大な危険の兆候を見逃すことにつながる。問題の報告とは単なる「バグ提出」ではなく、「安全性の警鐘」であり、それを正確に鳴らすためには、標準化された形式と責任ある記録が欠かせない。製品の進化と安全性は、こうした１件１件の報告から始まる。	● まとめ 細分箇条 9.1「問題報告の作成」は、ソフトウェアに問題が起きたときに「何が、いつ、どう起きたか」を正確に記録する工程である。この記録があることで、問題の調査や修正、再発防止策が確実に進められる。つまり、「問題報告」は単なる報告書ではなく、ソフトの安全性と品質を守る最初の防波堤である。誰もが正しく報告できる仕組みを整えることが、信頼できる医療ソフトを育てる第一歩となりうる。	
	9.2 問題の調査	（１）序論 細分箇条 9.2「問題の調査」は、ソフトウェアの不具合や異常事象が報告された際に、その原因を突き止め、影響の範囲を明らかにし、今後の対処方針を決定するための重要な工程である。これは、問題を「正確に理解する」ことによって、誤った対策や過剰・過少な修正を防ぐために不可欠な活動である。医療機器ソフトウェアは、使用環境・入力条件・他機器との連携等多様な要因が複雑に絡み合って動作しているため、問題が発生した場合でも「なぜ起きたのか」を即座に判断することは難しい。そのため、本条項では、体系的で客観的な分析手法に基づき、原因を段階的に追跡・特定するプロセスが求められる。	● はじめに 細分箇条 9.2「問題の調査」は、医療機器ソフトウェアで問題（バグや不具合等）が見つかったときに、その原因をしっかりと調べるための工程である。前の段階（9.1）で「問題があった」と報告された内容をもとに、「なぜ起きたのか？どこが悪いのか？どう対応すべきか？」を明らかにしていくことが、この調査の目的である。例えば、あるソフトで心拍数の表示が止まってしまったという問題が報告されたとする。その場合、ただ「直す」のではなく、「なぜその表示が止まったのか？」という原因を調べない限り、また同じような不具合が起きてしまう。だからこそ、調査はとても重要であり、ソフトの品質と安全性を守るための第一歩である。	1) GB（※）：P183 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）問題調査の目的 問題調査の主要な目的は、以下の 4 点である。 ① 問題の再現性を確認する 報告された事象が事実であるか、どの条件で発生するかを明らかにする。 ② 原因を特定する プログラム上の不具合なのか、設計仕様の欠落なのか、外部要因によるものなのかを分類・分析する。	● 問題調査でやることとは？ 問題の調査として次のようなことを行う。 ① 問題の再現を試みる ・ 同じ条件を再現して、本当にその問題が起きるかどうかを確かめる。 ・ 「再現できない問題」は、調査が難しいため、報告の内容が正確であることが前提となる。 ② 原因を調べる ・ ソフトのどの部分で問題が発生したのか？	

		③ 影響範囲を把握する 類似バージョンや他製品への波及があるかどうかを評価する。 ④ リスクレベルを評価する の問題が医療行為や患者の安全にどのような影響を及ぼし得るかを明確にする。 これらを通じて、適切な対応方針（修正、設計変更、マニュアル更新、再教育等）を選定する基盤が形成される。	・ プログラムのミスなのか、設計の見落としなのか？ ・ ユーザーの使い方による誤操作なのか？ 等を一つひとつ可能性を検討し、根本的な原因（Root Cause）を探る。 ③ リスクへの影響を判断する ・ この問題は、患者の安全にどれくらい影響するのか？ ・ 医療現場で使われた場合に、どのような危険があるか？ というように、「どれくらい深刻か」を評価し、対応の優先順位を決める。 ④ 修正が必要かどうかを判断する ・ 軽微な問題であれば、様子を見るという判断もありうる。 ・ 安全に関わるものであれば、すぐに修正の計画を立てる必要がある。 ● なぜ「調査」が重要なのか？ 問題が起きたときに、すぐに修正してしまうと「その場しのぎ」で終わってしまい、本当の原因がわからないままになる危険がある。例えば、 ・ 表示がおかしかったのでデザインだけを直した 実は裏の計算式が間違っていた ・ 動きが遅かったので処理時間を伸ばした 本当はメモリ不足が原因だった このような対応では、また同じような問題が出てきてしまう。だからこそ、問題の「本当の根っこ」を突き止める調査が必要である。例えば、演劇をやるときに、音響が急に鳴らなくなったとする。このとき、 ・ 「スピーカーの電源が切れていた？」 ・ 「ケーブルが抜けていた？」 ・ 「音源のスマホがミュートになっていた？」 等、原因を一つずつ調べていくはずである。これが「問題の調査」であり、ただボリュームを上げるだけでは解決にはならない。ソフトウェアでもまったく同じで、「なぜ」を突き止めることが、次のステップへの正しい道しるべになる。	
		（３）調査における主なステップ		

問題調査は、以下のような手順で進められる。

① 事実確認と再現試験
報告内容に基づいて、問題が実際に発生する条件を明確化する。

・ 発生した製品のモデル、ソフトウェアバージョン、設定状態の特定
・ 入力条件、操作手順、発生頻度の確認
・ 再現試験の実施とログ取得

このステップは、特に再現性の有無を確認する上で重要である。再現できなければ、原因分析や検証が進められない。

② 根本原因の特定
再現された事象に対して、原因を技術的に突き止める工程である。よく用いられる手法には以下がある。

・ コードレビュー
問題の発生源となる関数・モジュールを特定

・ バグトラッキングとの照合
過去に類似の問題が存在していなかったかを調査

・ デバッグツールの使用
処理フローをトレースして異常点を把握

・ ログ解析
通信記録やエラーメッセージから不整合の発生箇所を突き止める

・ 設計仕様との照合
仕様の誤解や曖昧な要件によってバグが発生した可能性も評価

③ 影響範囲の分析
原因が判明した後は、同様のロジック・コード・仕様が使われている他の場所（モジュール、製品、バージョン）に同様の問題が存在しないかを確認する。例えば、

・ 同一関数が別モジュールでも使用されている
・ 同じ外部ライブラリを他の製品でも使用している
・ 派生製品が同じバージョンをベースにしている

<table>
<tr>
<td></td>
<td></td>
<td>この分析により、問題が局所的なものか、広範囲に波及するものかが判断される。

④ リスク評価と緊急度判定
問題が患者の健康に与える可能性のある影響について、ISO 14971 に基づいてリスク評価を行う。

・ 危害の重篤度
・ 発生確率
・ 現行のリスクコントロール策の有効性

これらを総合して、「即時対応が必要か」「次回リリースで修正すればよいか」「記録のみで対応不要か」等の対処方針が決定される。</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>（４）調査結果の文書化
IEC 62304 では、調査結果を文書に記録し、次工程（関係者通知や変更管理）と連携できる状態にしておくことを求めている。記録には以下のような情報が含まれる。

<table><tr><th>項目</th><th>内容例</th></tr><tr><td>問題報告 ID</td><td>PR-20250510-001</td></tr><tr><td>再現条件</td><td>特定のモードでデータ量が閾値を超える場合に発生</td></tr><tr><td>原因分析</td><td>BufferOverflow 関数の範囲チェック漏れ</td></tr><tr><td>影響範囲</td><td>AlarmModule、DoseControlModule</td></tr><tr><td>リスク評価</td><td>クラス C、緊急修正が必要</td></tr><tr><td>結論</td><td>修正要／変更管理プロセスを起動</td></tr></table></td>
<td>● 調査の記録も大切
調査の内容は、必ず文書に残すことが求められている。なぜなら、

・ 後で「何を調べたか」「どう判断したか」を見返すことができる
・ 規制当局（例えば厚生労働省等）に説明が必要になることがある
・ 別の問題が出たとき、過去の調査結果が役立つことがある

例えば、「2025 年 5 月 12 日、ログ解析の結果、心拍数表示停止の原因は通信エラーだった」といった形で、原因・経緯・影響・結論をしっかり記録することが、信頼できるソフトづくりにつながる。</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>（５）結論
細分箇条 9.2「問題の調査」は、ソフトウェア不具合の真の原因を特定し、正しい対処へ導くための最も重要な工程である。問題への対応は、単に「直す」ことではなく、「なぜそうなったか」「どれだけ影響するか」を明らかにすることが安全性の担保につながる。調査とは、事実を探るだけでなく、「再び同じ問題を起こさない」ための知見を得る行為である。医療機器の信頼性は、このような丁寧で継続的な分析作業によって支えられている。</td>
<td>● まとめ
細分箇条 9.2「問題の調査」は、ソフトウェアで起きた問題の本当の原因を明らかにし、安全性や品質への影響を評価するための工程である。ここでの調査が正しく行われなければ、表面的な対応に終わってしまい、再発を防げなくなる。だからこそ、「調査」は単なる技術的な作業ではなく、ソフトウェアの安全を守る知的な探偵活動のようなものであり、次の判断を正しく導くための重要なプロセスになりうる。</td>
<td></td>
</tr>
</table>

	9.3 関係者への通知	（１）序論 細分箇条 9.3「関係者への通知」は、ソフトウェアにおける問題が確認された後、その問題の性質・影響・緊急性に応じて、適切な関係者に情報を速やかに共有することを目的とする工程である。通知は単なる連絡ではなく、問題の拡大防止や、迅速な対応を促進するための「リスクコミュニケーション」としての機能を持つ。医療機器のソフトウェアにおいては、たとえ軽微に見える不具合であっても、診断・治療・モニタリング等患者の生命に関わる結果につながる可能性がある。そのため、関係部門や外部関係者が状況を把握し、適切な判断と対応を行えるよう、正確かつ迅速な通知が求められる。	●　はじめに 細分箇条 9.3「関係者への通知」は、医療機器のソフトウェアで問題が見つかり、その内容や影響が明らかになったときに、その情報を関係する人たちに正しく伝えるための決まりである。問題を見つけたあとは、その調査や対策だけでなく、「この問題は誰に知らせるべきか？」を考え、必要な人にきちんと伝えることが非常に重要である。これは、患者や使用者の安全を守るだけでなく、会社の信頼や責任を果たすうえでも欠かせない行動である。	1）GB（※）：P184 2）GVP に要求される対応（不具合報告や添付文書、情報提供文書等、安全対策上の措置）と関連するため、規制当局に求められる必要な情報を入手し、それらに基づいた必要かつ迅速な対応を、通知する相手にあった形で通知することが重要になる。 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）通知の目的 関係者への通知には以下の目的がある。 ① 問題の存在と重要性を共有すること 初動の遅れによる被害の拡大を防ぐ。 ② 役割に応じた対応を促すこと 設計部門には修正を、品質部門にはリスク評価を、営業部門には顧客対応を依頼する。 ③ 組織内外の一貫した情報対応を確保すること 複数部門が別々の認識で動かないようにする。 ④ 記録とトレーサビリティの起点とすること 誰に、いつ、どのような内容を伝えたかを文書化しておくことで、将来の監査や訴訟にも備える。	●　なぜ通知が大切なのか？ ソフトウェアの問題は、黙っていても自然には解決しない。例えば次のような危険がある。 ・　医師や看護師が、問題を知らずにそのまま使ってしまう ・　別のチームが古いバージョンのソフトを誤って使い続ける ・　修正されたと思っていたが、実は誰にも反映されていなかった これらはすべて、「情報が届いていなかった」ことによるトラブルである。だからこそ、問題が明らかになったら、できるだけ早く、正確に、必要な人に伝えることが大事になる。	
		（３）通知すべき「関係者」とは誰か 通知対象となる関係者は、問題の種類や影響範囲に応じて以下のように分類される。 ① 社内関係者 ・　設計部門：原因の調査、修正、影響分析を担当 ・　品質保証部門：リスク評価、修正の妥当性確認を担当 ・　製造部門：製品の出荷停止や再作業の判断材料とする ・　営業・カスタマーサポート部門：顧客からの問合せ対応、通知文書の作成を行う ・　上層部／経営層：重大問題の場合の意思決定を担う		

② 社外関係者

- 顧客（医療機関・販売代理店等）：使用中止の判断や運用上の注意喚起のため
- 規制当局（PMDA、FDA、NB 等）：重大な安全性問題の場合は法令上の報告義務がある、また、QMS や設計審査において問題対応をチェックする立場にある
- サービスプロバイダ：クラウドや通信等の外部機能に影響がある場合

（４）通知内容に含めるべき情報

IEC 62304 では通知内容の標準様式は規定していないが、一般的には以下のような項目が含まれる。

項目	内容例
問題 ID	PR-20250510-001
発見日／報告日	2025 年 5 月 10 日
製品名・バージョン	Model A / v2.1.3
発生条件	特定の設定下で画面がフリーズする
影響範囲	AlarmModule、DoseModule を含む機種群
暫定対応策	一時的な設定変更により回避可能
修正予定	修正バージョン v2.1.4 を 5 月末に提供予定
要求対応	使用中止／注意喚起／更新の実施 等

このような通知文書は、内部向けと外部向けで表現・用語が異なることがあるため、誤解を招かないよう分けて作成する必要がある。

● 通知内容に含めるべき情報

通知には、次のような情報を明確に含めることが望ましい。

- 問題の内容（どんな問題が起きたか）
- 原因（分かっていれば）
- 影響範囲（どの製品・バージョンか）
- 対策の方針（修正予定や注意喚起）
- 緊急性（使用停止の必要があるかどうか）
- 連絡先（問い合わせ先や追加情報の提供方法）

これらを丁寧に伝えることで、誤解やトラブルを防ぎ、安全な対応が可能になる。

● 通知の手段とタイミング

通知の手段は状況によってさまざまである。例えば、

- 社内向け：メール、会議、社内ポータル等
- 顧客向け：通知文書、FAX、電話、Web ページ等
- 規制当局：正式な報告書（フォーマット指定あり）

重要なのは「誰に、いつ、どうやって伝えたか」を記録として残すことである。これにより、「通知した・していない」というトラブルを避けることができる。

（５）記録と追跡

誰に、いつ、どの手段で通知したかについては、すべて文書として記録され、将来的に追跡できるようにしておく必要がある。

- 社内：連絡メール、報告書のコピー、社内掲示物の記録
- 社外：顧客への通知書（郵送または電子）、規制当局への報告記録（報告

		書・提出証跡） この記録は、監査・トラブル調査・製品責任訴訟等において重要なエビデンスとなる。		
		（６）結論 細分箇条 9.3「関係者への通知」は、ソフトウェアにおける問題対応を「組織全体の力で解決する」ための初動活動であり、安全性維持のための重要な連携手段である。問題の発見と原因調査が適切に行われても、その情報が伝わらなければ、適切な対策も実現されない。通知とは単なるお知らせではなく、リスクに基づいた判断と行動を共有するための戦略的な活動であり、医療機器の安全文化において欠かすことのできない柱である。	● まとめ 細分箇条 9.3「関係者への通知」は、ソフトウェアで問題が見つかり、調査された後に、その内容や影響をすべての関係者に正しく伝えることで、安全な対応を実現するための大切なステップである。「伝える」という行為は、単に知らせるだけでなく、「危険を未然に防ぐ」「信頼を守る」「次の行動につなげる」ための土台である。情報共有の力が、安全と品質を守る大きな武器になる。	
	9.4 変更管理プロセスの使用	（１）序論 細分箇条 9.4「変更管理プロセスの使用」は、ソフトウェアにおける問題を修正する際に、それが設計、コード、仕様、テスト文書等の構成要素に変更を加える行為であるならば、正規の変更管理プロセスに従って実施することを義務づける規定である。医療機器ソフトウェアの開発においては、たとえ小さな修正であっても、安全性・有効性・規制対応に影響を及ぼす可能性がある。よって、すべての修正は場当たり的に行うのではなく、変更の要否を検討し、影響を分析し、正式な手続きを経て実施する必要がある。この条項は、問題解決という緊急対応の場面であっても、「プロセスの統制」を優先させるという、医療機器開発特有の考え方を反映している。	● はじめに 細分箇条 9.4「変更管理プロセスの使用」は、医療機器ソフトウェアに問題が発生し、それを修正しようとする場合に、「変更管理」というルールにしたがって変更を進めなさいという内容である。ソフトウェアに問題があったとき、いきなりコードを書き直して修正してしまうと、他の部分に悪い影響を与えてしまうことがある。とくに医療機器では、ソフトの小さな変更が患者の命に関わる結果になることもある。そのため、どんな変更も「変更管理プロセス」によって、安全に、計画的に、記録を残しながら行う必要がある。この箇条では、「問題対応＝修正」のときも、その変更はきちんと管理された方法でやりなさい、という原則が示されている。	1） GB（※）：P184-185 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）変更管理プロセスの必要性 問題に対する対応が変更を伴う場合、その変更は以下のようなリスクを内包している。 ・　既存機能への副作用（リグレッション） ・　安全機能の動作不全 ・　他の設計仕様や文書との整合性崩壊 ・　規制要件（例：ISO 14971、QMS）からの逸脱 そのため、すべての修正行為は、あらかじめ定義された変更管理手順を適用することで、上記のような修正による新たな問題を防止することが可能となる。		
		（３）変更管理プロセスの基本的流れ IEC 62304 では変更管理の詳細な手順までは定義していないが、一般的なプロセスは以下のように構成される。		

		① 変更要求の提出 問題報告 ID と関連付け 修正の目的、背景、対象構成要素を明示 ② 影響範囲の分析 直接・間接の影響を設計、コード、テストの観点から洗い出す 他の製品バージョンへの波及も検討 ③ リスク評価の実施 修正により新たなリスクが発生しないか 既存のリスクコントロール策が維持されるか ④ 承認と実施 品質保証部門、設計責任者による承認 実装とテストの実施（再テストを含む） ⑤ 変更記録の保存 変更 ID、内容、影響、承認記録、試験結果を文書として保持		
		（４）問題解決と変更管理の統合 IEC 62304 の問題解決プロセスと変更管理プロセスは独立した機能ではなく、連携するべき活動である。特に以下のような連携が求められる。 ・ 問題報告と変更要求を一対一、あるいは一対多で対応付ける ・ 問題の影響範囲分析が、そのまま変更影響評価に活用される ・ 問題のリスク評価結果に基づき、変更の優先度や承認ルートを決定する ・ 問題対応の試験結果が、変更検証結果として記録される このように、問題対応と変更統制をセットで管理することが、安全性と文書整合性の維持に寄与する。	● 細分箇条 9.4 で求められること この細分箇条 9.4 では、次のようなポイントが特に重要とされている。 ① 問題の修正に「変更管理プロセス」を必ず使うこと 修正内容がどんなに小さくても、変更とみなされるものはすべて管理対象にするというのが基本ルールである。 ② 安全性に影響がある変更かどうかを評価すること 修正によって、安全性に悪影響が出ないかをリスクの観点から評価する必要がある。場合によっては、リスクマネジメントの再実施も必要となる。 ③ 修正後の確認を行うこと 修正が完了したら、それで終わりではなく、きちんと動作するか、元の問題が本当に解決しているかをテストすることが求められる。 例えば、演劇で照明がうまく点灯しないトラブルが発生したとする。そのとき、誰かが勝手に配線を変えてしまったら、今度は音響が使えなくなってしまった、ということが起きるかもしれない。これを防ぐには、	

			・　何を変えるかを記録する（例：スイッチの場所を変更） ・　関係者に伝える（照明班、音響班、電源係等） ・　試してみる（点灯確認テスト） ・　元に戻せるようにしておく（記録を取る） 　このような対応が、「変更管理プロセス」の考え方である。ただ直すのではなく、「安全に直す」「問題が広がらないようにする」という考え方がとても大切である。	
		（５）結論 　細分箇条 9.4「変更管理プロセスの使用」は、ソフトウェアの問題に対する修正が「安全に」「確実に」「文書化された形で」行われるための仕組みであり、医療機器の品質保証体制の根幹をなす活動である。不具合をただ直すのではなく、「正しい手順で、安全を守りながら直す」ことが重要であり、そのためには変更管理という枠組みを確実に適用することが求められる。安全はプロセスの継続的な適用によって保証される。	●　まとめ 　細分箇条 9.4「変更管理プロセスの使用」は、ソフトウェアに問題が見つかって修正が必要なときに、その修正を計画的かつ安全に行うために、変更管理というルールを正しく使うことを求める工程である。このプロセスを通して、「直したことで新たな問題が出ないようにする」「修正の履歴を明確に残す」「安全性を常に確保する」といった目的が達成される。つまり、変更管理は「よかれと思ってやったことが、裏目に出ないようにする」ための仕組みである。	
	9.5 記録の保持	（１）序論 　細分箇条 9.5「記録の保持」は、ソフトウェアにおける問題対応に関して、発見から調査、対処、検証、最終的なクローズに至るまでの一連の活動を正確かつ網羅的に記録し、それらを適切に保管することを求める規定である。医療機器ソフトウェアの品質と安全性は、単に問題が「解決された」という事実のみならず、それが「どのように解決されたか」「関係者が何を判断し、どのような手順で処理したか」という記録によって証明される。ゆえに、記録の保持は単なる事務作業ではなく、品質保証と法的責任の両面において重要な意味を持つ。	●　はじめに 　細分箇条 9.5「記録の保持」は、医療機器に使われるソフトウェアで問題が起きたとき、その対応の記録をきちんと残しておくことを義務づけたルールである。ソフトウェアで問題が発生し、調査して原因を特定し、修正まで終わったとしても、それを記録として残していなければ、「本当にちゃんと対応したのか？」「同じことが前にもなかったか？」という大事なことが確認できない。だからこそ、問題対応の流れを文書としてしっかり残しておくことが必要である。	1）GB（※）：P185 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）記録の保持の目的 　このプロセスの主な目的は以下の通りである。 ①　問題対応の経緯と妥当性を証明すること 　将来的な監査・訴訟・承認審査等において、問題への対応が適切であったことを文書で示す。 ②　組織内での知見の蓄積・共有を可能にすること 　類似問題の再発時や新製品開発時に、過去の記録を参考にできる。 ③　問題傾向の分析に活用できるようにすること	●　なぜ記録が重要なのか？ 　医療機器において「記録がない対応は、していないのと同じ」とされる。それくらい、「記録があること」が信頼性の証明になるからである。例えば、 ・　同じような不具合が別の病院でも起きたときに、「以前も発生していた」と分かれば、早く対応できる。 ・　法律上のトラブルや、製品の安全性についての問い合わせが来たときに、記録があれば「ちゃんと対策していた」と証明できる。 ・　社内の別のチームが同じミスをしそうになったとき、過去の記録を見れば再発防止になる。	

<table>
<tr><td></td><td></td><td>特定のモジュールや工程で繰り返し問題が発生していないかを統計的に把握できる。

④ トレーサビリティ（追跡可能性）を維持すること
各問題がどの変更に対応し、どのテストで確認されたかを時系列で追えるようにする。</td><td>このように、記録は「安全の証拠」となる。</td><td></td></tr>
<tr><td></td><td></td><td>（３）記録する内容
記録として保持する情報は、以下のようなものがある。

<table><tr><th>記録項目</th><th>内容例</th></tr><tr><td>問題報告 ID</td><td>PR-20250510-001</td></tr><tr><td>発見日時・報告者</td><td>2025 年 5 月 10 日／山田技師</td></tr><tr><td>ソフトウェア名・バージョン</td><td>AlarmModule v2.1.3</td></tr><tr><td>問題の概要</td><td>高負荷時に表示がフリーズ</td></tr><tr><td>原因分析の結果</td><td>UI 更新処理の非同期競合</td></tr><tr><td>影響範囲の分析結果</td><td>DisplayModule に波及の可能性</td></tr><tr><td>修正方針と実施内容</td><td>非同期処理の同期化を実装</td></tr><tr><td>リスク評価</td><td>クラス C、即時対応必要と判断</td></tr><tr><td>試験の実施内容</td><td>回帰試験 10 項目、すべて合格</td></tr><tr><td>承認者と承認日</td><td>品質保証部長／2025 年 5 月 15 日</td></tr><tr><td>最終クローズ日</td><td>2025 年 5 月 17 日</td></tr></table></td><td></td><td></td></tr>
<tr><td></td><td></td><td>（４）結論
細分箇条 9.5「記録の保持」は、問題解決プロセスにおける「行動の証跡」を残すことで、安全性・信頼性・説明責任を担保する中核的な活動である。記録は、ただの履歴ではなく、「製品が安全であること」「問題に適切に対処されたこと」の証拠であり、将来にわたって製品と企業を守る「防衛線」でもある。安全な医療機器開発は、正しい記録とそれを活かす仕組みの上に成り立つ。</td><td>● まとめ
細分箇条 9.5「記録の保持」は、ソフトウェアの問題に対応した一連の作業内容を、証拠としてしっかり記録に残し、将来にわたって参照できるようにするための工程である。これは、「安全に対処する」だけでなく、「その安全性を証明する」ための土台となる。つまり、記録とは、過去を振り返り、未来の事故を防ぐ安全のバトンのようなものであり、医療機器ソフトにとって欠かせない信頼の支えとなる。</td><td></td></tr>
<tr><td></td><td>9.6 問題の傾向分析</td><td>（１）序論
細分箇条 9.6「問題の傾向分析」は、過去に発生したソフトウェアの問題記録を体系的に分析し、潜在的な設計上の弱点や品質上の偏りを見つけ出し、将来の不具合を未然に防ぐことを目的とした活動である。単発の不具合に対処す</td><td>● はじめに
細分箇条 9.6「問題の傾向分析」とは、ソフトウェアで発生したさまざまな問題（バグ、不具合、動作ミス等）を集めて調べ、それらの共通点や傾向を見つける作業である。問題が 1 件起きたとき、それを調査して修正することはと</td><td>1） GB（※）：P185

（※）GB：IEC 62304 実践ガイドブック（じほう）</td></tr>
</table>

		るだけでなく、「なぜ同じような問題が繰り返されるのか」「どの部分に集中して発生しているのか」といった根本的な改善課題を抽出することは、医療機器ソフトウェアの長期的な品質・安全性向上において不可欠である。	ても大事である。しかし、同じような問題が何回も起きていたらどうだろうか？それは、たまたまのトラブルではなく、どこかに根本的な原因があるかもしれないというサインである。傾向分析とは、こうした「見えない危険のパターン」を見つけ出し、再発防止やソフトの品質向上につなげるための取り組みである。	
		（２）傾向分析の意義 傾向分析は、以下のような重要な意義を持つ。 ① 再発防止策の立案に役立つ 同種の問題が繰り返されている場合、設計そのものや運用体制に構造的な課題があると判断できる。 ② 品質改善の優先順位決定に貢献する 問題件数や重大度が集中しているモジュールを特定し、開発リソースを重点配分できる。 ③ 外部監査や規制当局への説明に有効である 「組織として問題を管理・改善している」という姿勢を記録と分析結果によって証明できる。 ④ 継続的改善を実現する基盤となる ISO 13485 や QMS 体制における PDCA サイクルの「Check」に相当する要素として機能する。	● 傾向分析とは何をすることか？ 傾向分析では、以下のような観点から複数の問題を比較・分析する。 ・ どんな種類の問題が多いか？ ・ 特定の機能やモジュールで繰り返し発生していないか？ ・ どのソフトのバージョンや時期に多く発生しているか？ ・ 開発工程や作業者に偏りはないか？ ・ 発生頻度が増えていないか？ これらを総合的に調べることで、「このままだと将来また大きな問題につながるかもしれない」というリスクに気づくことができる。 ● なぜ傾向分析が必要なのか？ １つの問題を個別に対応するだけでは、「同じような問題を何度も繰り返す」ことになる。これは時間とコストのムダであるだけでなく、医療機器としての信頼性を損なう危険がある。例えば、 ・ 年に 3 回も「アラーム音が鳴らない」バグが発生している ・ 毎回修正しているのに、次のバージョンでまた発生 ・ 調べたら、アラーム設定モジュールの設計ミスが原因だった このように、傾向を分析して根本の原因にたどり着ければ、より効果的な再発防止ができる。	
		（３）傾向分析の進め方 IEC 62304 では具体的な分析手法までは定義していないが、一般的な手順は以下の通りである。 ① データ収集 ・ 問題報告書、変更履歴、修正記録、試験成績書、顧客苦情記録等から、一定期間内の事象を抽出 ・ 分析対象は可能な限り網羅的にし、部門・工程・ソフトウェアバージョンの偏りを排除する		

		② 分類と整理 ・ 発生モジュール、問題種別、重大度、安全クラス、原因分類等の軸で分類 ・ 表計算ソフトや専用ツールを用いてクロス集計・フィルタリング ③ パターンの特定 ・ 「特定のモジュールに不具合が集中している」 ・ 「リグレッションが多く発生している」 ・ 「入力チェック漏れが原因となる問題が頻出している」 といった傾向を可視化する。グラフやヒートマップ等の視覚化が有効である。 ④ 根本原因の探索 ・ 表面的な分類にとどまらず、なぜその問題が繰り返されるのかを設計・運用・文化的視点から深堀りする ・ フィッシュボーン図（特性要因図）や 5Why 分析等の手法が有効 ⑤ 是正措置・予防措置（CAPA）の立案 ・ 頻出モジュールのコードレビュー強化 ・ テスト項目の再設計 ・ 開発者教育の見直し ・ 設計標準・テンプレートの改善 等、分析結果に基づいた具体的な改善行動を設定する。 ⑥ フィードバックと記録 ・ 分析結果は品質会議や設計レビューで共有 ・ 改善計画とともに QMS 文書に反映し、次回以降の分析の出発点とする		
		（４）結論 細分箇条 9.6「問題の傾向分析」は、単発の問題対応を「将来の品質向上」へとつなげるための「知的改善活動」である。記録を蓄積し、そこから傾向を読み取り、再発を防ぎ、製品の信頼性を着実に高めていくことが、医療機器ソフトウェアの開発者に求められる責務である。製品の進化は、失敗を記録し、学び、繰り返さないという姿勢から始まる。そのためにこそ、傾向分析は欠かせないプロセスである。	● まとめ 細分箇条 9.6「問題の傾向分析」は、ソフトウェアに起きた問題のデータを使って何が繰り返されているか、どこに注意が必要か、という兆候を見つけ出し、より安全でミスの少ないソフトウェアを目指すための大切な工程である。単発の問題対応では気づかない全体の動きに目を向けることで、より深い理解と、より本質的な改善が実現できる。それが、この傾向分析の持つ力である。例えば、毎週提出するレポートに、例えば「まとめの書き方が弱い」「表紙の書式ミス」「提出期限オーバー」等の同じようなミスが何度も出ているとす	

			る。そのときに、「誰が何回間違えたか」「どのページの問題が多いか」等を先生が分析して、「ここをもっと分かりやすく説明しよう」と対策を立てる。これが「傾向分析」である。同じミスを繰り返さない工夫をするための作業という点では、ソフトウェアでもまったく同じである。	
	9.7 ソフトウェア問題解決の検証	（１）序論 細分箇条 9.7「ソフトウェア問題解決の検証」は、報告されたソフトウェアの問題に対して行った対処（修正、変更、制御策等）が、実際に意図通りに機能しており、問題を解決していることを客観的に確認する工程である。この条項は、単に修正を加えることに満足するのではなく、「それが本当に効果を発揮しているか」「副作用を起こしていないか」を実証するという、医療機器における安全性確保の本質を反映したものである。	● はじめに 細分箇条 9.7「ソフトウェア問題解決の検証」は、医療機器のソフトウェアに発生した問題に対して修正を行ったあと、本当にその修正が正しく行われていて、安全性や機能に悪い影響を与えていないかを確認する作業である。言い換えれば、「修正したつもり」で終わらせず、「ちゃんと直ったかどうか」「その直し方が他のところに悪影響を与えていないか」を確かめることが、この検証の目的である。	1) GB（※）：P186 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）「検証」の必要性 医療機器ソフトウェアにおいて、バグや設計ミスに対する修正は、製品の信頼性を回復させるうえで不可欠であるが、修正が新たな問題を生むリスクも常に存在する。例えば、以下のような事態が起こり得る。 ・ 修正が別の機能を間接的に破壊する（リグレッション） ・ 修正が誤った箇所に適用されている ・ 修正後にリスクが増加する ・ 修正の効果が限定的で、再発可能性が残っている そのため、修正の有効性と副作用の有無を客観的な手段で確認することが不可欠であり、それを担保する工程がこの「検証」である。	● なぜ検証が必要なのか？ ソフトウェアの修正は、問題のある部分だけに手を加えたつもりでも、実は他の部分に影響を与えてしまうことがある。それはまるで、時計の歯車の１つを少し変えたら、全体の動きがズレてしまうようなものである。例えば、 ・ アラーム機能の不具合を直したら、今度は設定画面の表示がずれてしまった ・ データ保存のバグを修正したら、ファイルが壊れる別の不具合が発生した このようなことが起きないように、修正した結果をしっかり検証して、「直したことで他が壊れていないか」を確認することが絶対に必要である。	
		（３）検証の目的と要件 IEC 62304 における検証とは、次の 3 点を明らかにすることである。 ・ 問題が再現されないことを確認する 修正後も同じ条件で問題が発生しないことを試験で確認する。 ・ 修正が意図した通りに機能していること 修正内容に基づいた動作が正確に行われているかを確認する。 ・ 他の機能への悪影響がないこと（回帰の確認） 影響範囲内に含まれる周辺機能に異常が出ていないことを確かめる。 これらは、文書化され、第三者が再確認できる形で実施されなければならない。		

（４）検証の手段と内容
検証に使用される代表的な手段は以下の通りである。

① テスト（試験）
　最も一般的であり、修正後に同一のテストケースを再実行して合否を確認する。

・　再現テスト：修正前に問題が発生した条件を再現し、問題が消失しているか確認
・　機能テスト：修正された機能が意図通り動作しているかを確認
・　回帰テスト：修正が他機能へ悪影響を及ぼしていないかを確認

② レビュー
　変更された設計書、ソースコード、テスト仕様書等を第三者またはチーム内でレビューし、妥当性と一貫性を確認する。

③ 静的解析／ツール検証
　コードの構造や安全性に対して静的解析ツールを用いて、自動的にバグやセキュリティ欠陥が生じていないかを確認する。

④ シミュレーション／実機テスト
　場合によっては、開発環境だけでなく、実際の機器や実運用に近いシミュレーション環境で動作を確認することも行われる。

（５）検証結果の文書化
　検証結果は以下のような形で記録されることが望ましい。

項目	内容例
問題 ID	PR-20250510-001
検証責任者	品質保証課／鈴木
実施日	2025 年 5 月 18 日
使用テストケース	TC-ALM-004、TC-ALM-005
結果	全項目合格（ログ添付）
回帰影響有無	なし（TC-ALM-001～003 も合格）
結論	修正は有効、問題は解決された

		<table><tr><td>添付資料</td><td>試験ログ、画面キャプチャ、バージョン情報</td></tr></table>		
		（６）結論 細分箇条 9.7「ソフトウェア問題解決の検証」は、問題に対する修正が「確実に有効である」ことを証明するプロセスであり、医療機器ソフトウェアの安全性確保における最後の砦である。不具合を「修正した」だけでは不十分であり、「その修正が意図通りに機能し、安全性を回復した」ことを文書と試験で証明することが、社会的責任を伴う医療機器開発においては当然の要件になる。	● まとめ 細分箇条 9.7「ソフトウェア問題解決の検証」は、ソフトウェアの問題に対して修正を行ったあと、その修正がきちんと効果を発揮しているか、そして他に悪影響を与えていないかを確認する重要な工程である。これは、ただ「直した」と言うだけでは終わらず、本当に安全に直ったと証明できるかどうかが問われるステップである。信頼できる医療機器ソフトウェアは、このような一つひとつの丁寧な検証の積み重ねによって成り立っている。	
	9.8 試験文書の内容	（１）序論 細分箇条 9.8「試験文書の内容」は、ソフトウェアの問題解決に関連する試験を行った際に、その内容、結果、方法、根拠等を記録として適切に文書化することを求める規定である。この条項は、単に試験を「行う」だけではなく、その試験が誰にでも理解・再現可能であり、かつ後から証拠として利用できる状態にあることを重視している。すなわち、文書化は「信頼性の証明」と「トレーサビリティの確保」を同時に担う極めて重要な活動である。	● はじめに 細分箇条 9.8「試験文書の内容」は、医療機器のソフトウェアに問題が発生し、それを修正した後に行われたテスト（試験）の記録や報告書に、どんな情報を含めなければならないかを定めたルールである。ソフトウェアに限らず、どんな製品でも「修理しました」「試験しました」と口で言うだけでは信頼されない。本当に試験したのか？その結果はどうだったのか？どんな条件で試したのか？といった情報が、正しく文書に残っていなければならない。それがこの箇条で求められていることである。	1） GB（※）：P186 （※）GB：IEC 62304 実践ガイドブック（じほう）
		（２）試験文書の必要性 医療機器ソフトウェアにおいて、不具合の修正や問題解決後の状態が安全であることを確認する試験は、「安全性回復のための検証行為」である。その結果を記録せずに済ますことは、以下のような重大なリスクを伴う。 ・ 記録がないため、後日問題が再発した際の原因追跡ができない ・ 修正が不十分だったことに対して誰も気付けない ・ 規制当局の審査において「証明責任」を果たせない ・ 品質監査において「不適合」と判定される可能性がある したがって、試験文書は単なる技術記録ではなく、安全と信頼の裏付けとなる根拠資料となる。	● なぜ「試験文書」が必要なのか？ 医療機器のソフトウェアは、人の命や健康に直接関わるものである。例えば、バグを直したつもりでも試験していなければ、 ・ 直っていないことに誰も気づかない ・ 他の部分に悪影響が出ていることを見逃す ・ 証拠がないため、責任の所在が不明になる といった問題が起きる。だからこそ、「試験した」という証拠となる文書＝試験文書を正しく作成し、保管することが重要である。	
		（３）試験文書に含めるべき内容 IEC 62304 では、試験文書には以下のような情報を含めることが求められている。 ① 試験の目的と対象 ・ どの問題に対する試験なのか（例：PR-20250510-001 に対応）		

・　試験対象のソフトウェア構成要素（例：AlarmModule v2.1.4）

② 試験の条件
・　実施環境（OS、ツールバージョン、実機 or シミュレータ）
・　試験前提（初期状態、外部条件等）

③ 試験の手順
・　実行されたステップ（例：ボタン押下 → データ入力 → 応答確認）
・　使用したテストケースの ID（例：TC-ALM-004）

④ 試験結果
・　実際の出力／ログ
・　期待される出力との一致状況
・　合否判定（Pass／Fail）とその根拠

⑤ 実施責任者・日付
・　試験を誰が、いつ行ったかを明記
・　承認者と承認日も記録

⑥ 関連ドキュメントとのリンク
・　どの修正設計、変更管理、リスク文書と対応しているかを示す

（試験文書の記録例）

試験番号	項目	条件	期待結果	実結果	判定	試験日	担当者
TC-101	アラーム動作確認	体温 41℃設定	5 秒以内に音が鳴る	4 秒後に音	合格	2025/05/12	Tanaka

（４）結論

細分箇条 9.8「試験文書の内容」は、ソフトウェアの問題対応が「正しく、確実に、証拠を伴って」実行されたことを示す最終的な証明手段である。単なる試験結果の羅列ではなく、何をどう確認し、安全性がどう裏付けられたかを明確に記録した文書が、製品の信頼性を支える品質の盾となる。

● まとめ

細分箇条 9.8「試験文書の内容」は、ソフトウェアの問題を修正したあとの試験について、どのように記録し、どんな内容を含めておくべきかを定めた工程である。この記録は、単なるメモではなく、医療機器ソフトの「安全性と信頼性を証明するための武器」であり、責任と技術の結晶とも言える。つまり、試験文書とは「安全な医療を守る設計図の一部」となりうる。