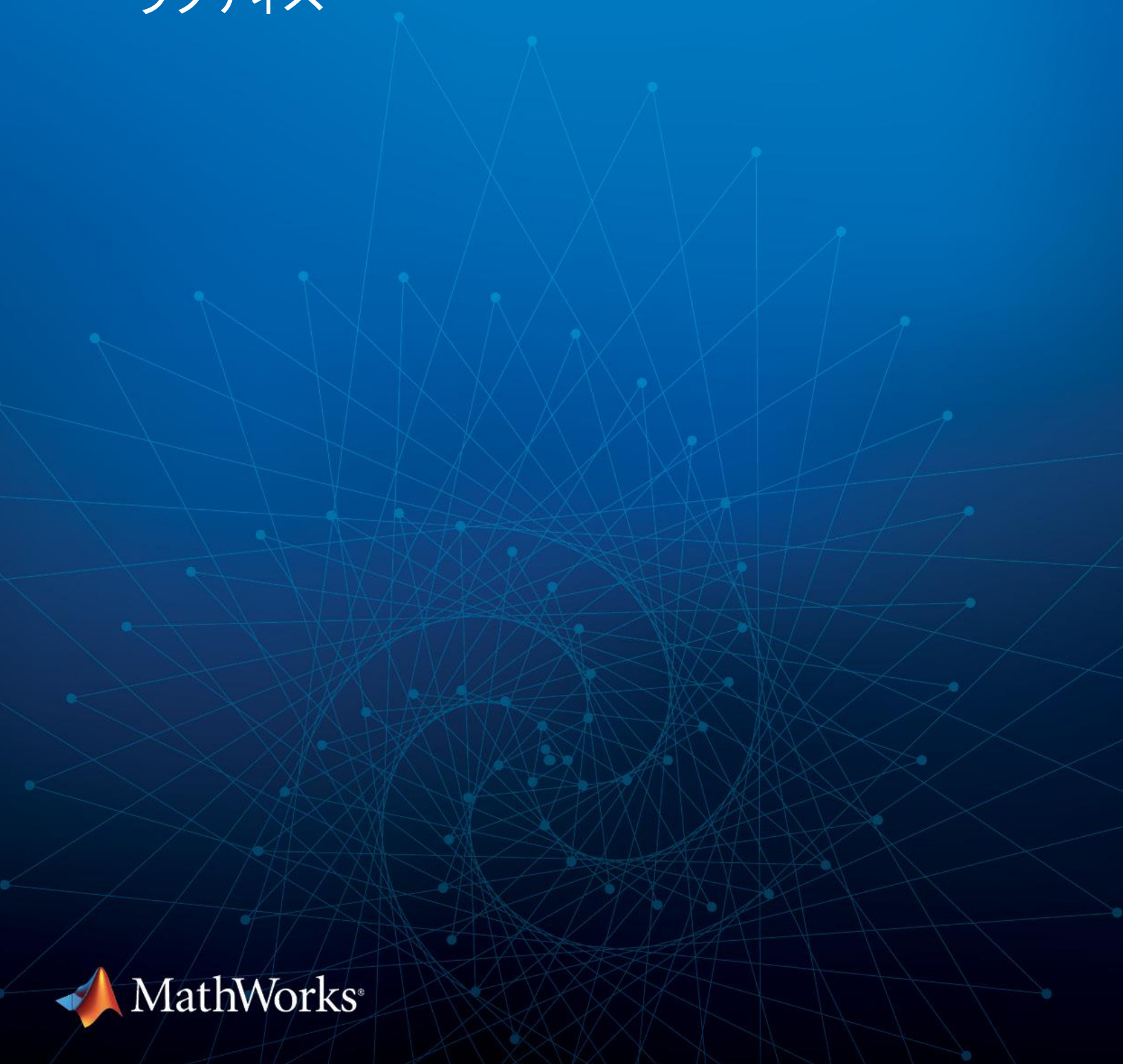
ホワイトペーパー

モデルベースデザインによるIEC 62304 準拠のためのベストプラクティス



MathWorks®

はじめに

医療機器のエンジニアにとって、安全規格 IEC 62304 への準拠は、多くの場合ドキュメントベースの要件、ハンドコーディング、物理デバイスのプロトタイピングを伴う作業となっています。

モデルベースデザイン (MBD、モデルベース開発) を使用すると、医療機器用の高信頼性ソフトウェアを迅速かつ費用対効果の高い方法で作成することができます。その中心となるのは、要件開発、アーキテクチャの解析と仕様化、詳細設計、実装、およびテストにわたるシステムモデルです。シミュレーションとラピッド プロトタイピングを通して設計を調整し、コードを生成して組み込みデバイスに実装します。

このホワイトペーパーでは、IEC 62304 におけるクラス B および C に準拠したプロセスで、MATLAB® および Simulink® を使用したモデルベースデザインを活用する方法について説明します (日本におけるクラス 2 以上)。このホワイトペーパーは、IEC 62304 規格で規定されている主なアクティビティに基づき構成されています。

用語

このホワイトペーパーでは、以下の定義が使用されます。

- 検証: 一連のソフトウェア開発が、要求されるアウトプットを満たしていることを客観的な証拠によって確認すること
- ソフトウェアユニット: 設計、実装、および検証される最小のソフトウェア コンポーネント
- ソフトウェアアイテム: 1 つ以上のソフトウェアユニットで構成されるソフトウェア コンポーネント。通常は中間サイズのものを指すが、ユニットからソフトウェアシステムに至るあらゆるものを指して総称的に使用されることもある
- ソフトウェアシステム: 1 つ以上のソフトウェアアイテムを含むソフトウェアのパッケージ。通常は機能の完全なパッケージを指すが、完全な医療機器のコンポーネントの 1 つのみを指すこともある
- サブシステム: Simulink モデルのグループ化されたコンポーネント。通常、複数の基本ブロックまたは固有ブロックを含む

ソフトウェア開発プロセス

ソフトウェア開発計画

IEC 62304 規格のセクション 5.1 に記載されているソフトウェア開発プロセスは、プロジェクトで主要アクティビティやタスクを達成するための計画から始まります (プロジェクトを分解したものがアクティビティであり、このアクティビティをさらに分解し日々の業務などにおとしこんだものがタスクです)。本規格では、開発プロセスのあらゆる側面に対応できるよう、複数の計画を開発することが推奨されています。モデルベースデザインを導入する組織は、これらの計画ドキュメントにおいて、開発戦略の一環としてツールに言及する必要があります。また、そのプロセスで生成されたアーティファクトをソフトウェアプロセスの成果物リストに含めることも重要です。

生成される可能性のある一般的なアーティファクトとしては、MATLAB® スクリプトや関数、Simulink モデル、データ辞書、生成された量産コード、S-Function などのユーザー ブロック ライブラリ、シミュレーション入力データ (テストベクトル) および結果、設計ドキュメントやテストレポートなどの生成されたドキュメンテーションなどがあります。IEC Certification Kit により、IEC 62304 などの規格への準拠を確認するための一連のモデル設計規格とツールが提供されています。

着手の順番としては、このようなモデル設計規格から始め、個人や組織のニーズに基づいて拡張または調整を行うことが通常です。さらに IEC Certification Kit では、モデル規格に加えて、TÜV SÜD で評価されたリファレンス ワークフローおよびツールの妥当性確認に関するドキュメンテーションも提供されています。MathWorks のモデルベースデザイン ツールは、IEC 62304 に準拠した機能安全の開発において、その妥当性が適切に確認されています。最後に、開発ツール自体もソフトウェア構成の一部として管理し、バージョン管理を行う必要があることに注意してください。

ソフトウェア要件解析

モデルベースデザインの大きなメリットは、設計者がシステムやソフトウェアの機能要件に対する理解を深められることです。初期の設計アイデアをライブ形式の実行可能モデルとして表現することで、従来のテキストによる要件ドキュメントの解析よりもはるかに具体的な方法で要件の一貫性と正確性を確立できます。Simulink と関連する Simscape™ ファミリの物理モデリングツールでは、ソフトウェアアイテムの動作をモデル化できるうえ、ソフトウェア アルゴリズム、機電系コンポーネント、患者の生理学など、ソフトウェアアイテムと相互作用する環境をモデル化できます。シミュレーションを使用すると、ソフトウェア システム モデルと関連する運用環境とを相互作用させることで、機能要件の詳細化と検証を行うことができます。また、モデルでは、基礎、臨床、または解析的に定義されたデータセットに基づいてシミュレーションを行うこともできます。Simulink 内で要件を作成、解析、および管理することができます。

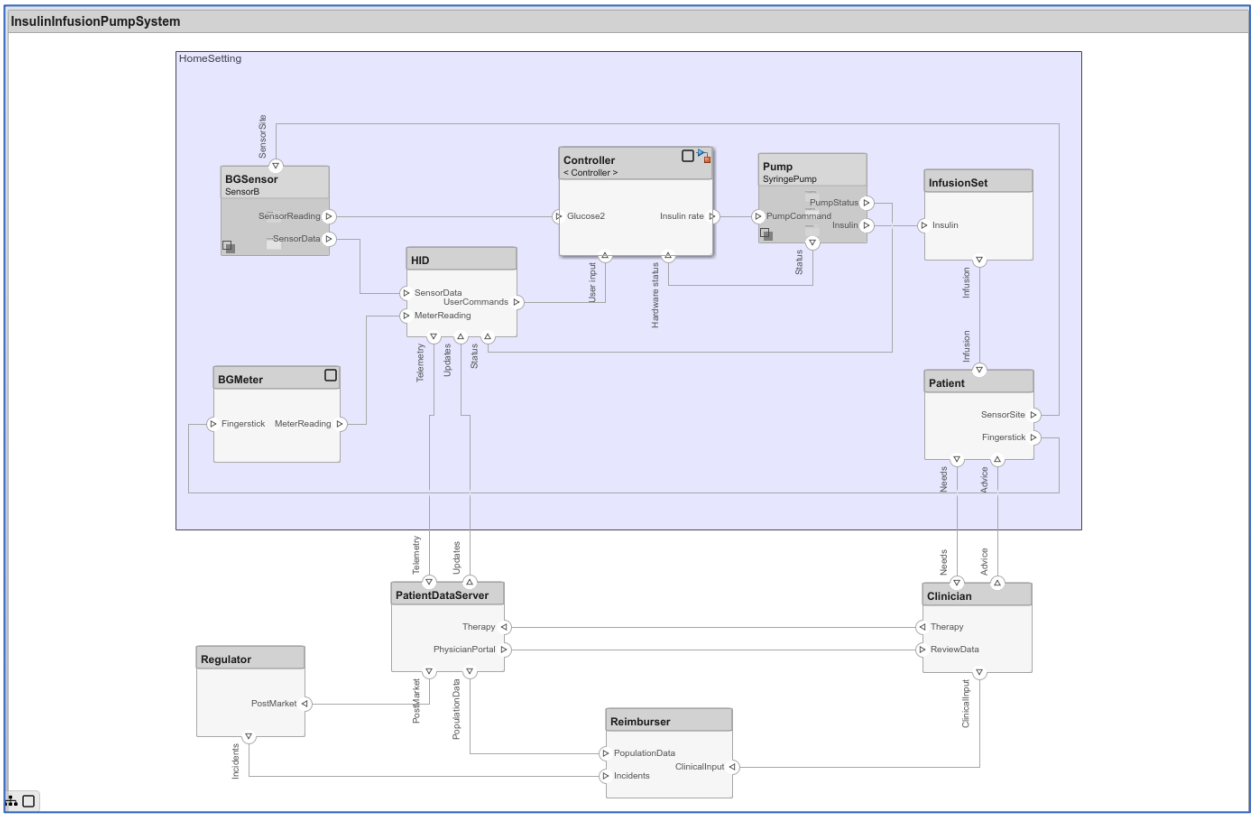
Requirements Toolbox™ には、Simulink ブロックや Stateflow® チャート要素など、要件とモデルのコンポーネントとの間にトレーサビリティを確立するためのリンク機能が用意されています。Simulink から生成されるレポートは、トレーサビリティ解析に対応しています。本規格では、危害から患者を保護するためのこれらのソフトウェアアイテムについて、特に注意することが求められています。このようなリスク制御策には関連する要件があり、それらの要件はあらゆる要件と同様に、Simulink モデル

の要素に関連付けることができます。ベストプラクティスとして、リスク制御要件を一意のキーワードでタグ付けして要件とレポートのフィルター機能を使用すると、リスク制御解析が容易になります。

ソフトウェア アーキテクチャ設計

本規格のセクション 5.3 および付表 B.5.3 によると、ソフトウェア アーキテクチャ設計の目標は、ソフトウェアを主要な構造コンポーネントに分割し、その外部プロパティを示して、コンポーネント間の関係を実証することです。System Composer™ を使用すると、モデルベース システム エンジニアリングとソフトウェア設計のための、アーキテクチャとコンポジションの定義、解析、および仕様化が可能になります。System Composer では、システムおよびソフトウェア コンポーネントの確立、それらのインターフェイスと依存関係の指定、それらのコンポーネントに対するシステム要件と安全要件の割り当てと追跡を行えます。

たとえば、リスク制御策をコンポーネントやサブコンポーネントに分離することで、より直感的にリスク制御策を追跡し、ソフトウェア アーキテクチャの他の部分についてソフトウェア安全クラスを下げるという用途も考えられます。



Simulink でのシステム アーキテクチャの例。コントローラーは、ソフトウェアアイテムを表す Simulink の動作モデルに関連付けられている。

Requirements Toolbox では、機能要件と入出力を System Composer モデルの代表的な構成要素に関連付けることができます。トレーサビリティ レポートを確認することで、ソフトウェア要件に対するソフトウェア アーキテクチャの検証をサポートするのに役立ちます。

一般的なシナリオでは、Simulink モデルから実装へのパス (Simulink Coder™ や Embedded Coder® を介した Simulink モデルからの量産コード生成など) は、ソフトウェアシステムの一部にのみ使用されます。サードパーティの部品メーカーが提供する低水準のドライバーコードが存在する場合や、類似製品や製品群の既存のソースコードを再利用したい場合などに、実用的と考えられます。モデルベースデザインは、ソフトウェアユニットやソフトウェアアイテムのモジュール実装をサポートしています。完全なソフトウェアシステム設計を表す Simulink モデルも完全に実装できます。

ソフトウェア アーキテクチャをグラフィカルに表現することは、表現するだけでなく、検証する上でも有効な手段です。さらに、コンポーネント間で共有されるデータのステレオタイプ化 (基本的に特定の種類のインターフェイスの再利用) により、インターフェイスの整合性をとることができます。アーキテクチャは、System Composer で対話的に確認したり、モデルから生成されたドキュメントのレビューにより確認したりできます。

本規格では、レビューが主要なアーキテクチャ解析手法として推奨されていますが、モデルベースデザインでは、シミュレーションを通してソフトウェア アーキテクチャが機能要件を満たしていることを検証する機会があります。System Composer で指定したアーキテクチャ コンポーネントを Simulink で実行可能な動作モデルに関連付けし、システムレベルの Simulink シミュレーションを作成して、システム アーキテクチャまたはそのサブセクションを実行できます。このような検証は、常にソフトウェア開発の早い段階で行われることが推奨されます。欠陥の修正にかかるコストは、ソフトウェア開発ライフサイクルのステージが遅れるにつれ、大幅に増加するためです。さらに、機能検証時に実行されたモデル部分のカバレッジ解析 (改良条件判定カバレッジなど) により、不完全な要件や不要な設計要素を明らかにすることができます。これらはいずれも開発サイクルの早い段階で解決する方がコストがかかりません。

ソフトウェア詳細設計

ソフトウェアの詳細設計は、Simulink で動作モデルとサブシステムの階層を作成し、ソフトウェア アーキテクチャを詳細化することで実現できます。アーキテクチャが System Composer で表現され、コンポネントとインターフェイス使用を取り込んでいるのであれば、Simulink と System Composer は統合/協調し、アーキテクチャと動作モデルの紐付けをします。この紐付けにより、ソフトウェア設計とアーキテクチャの整合性を証明するソフトウェアシステム設計の検証が可能です。

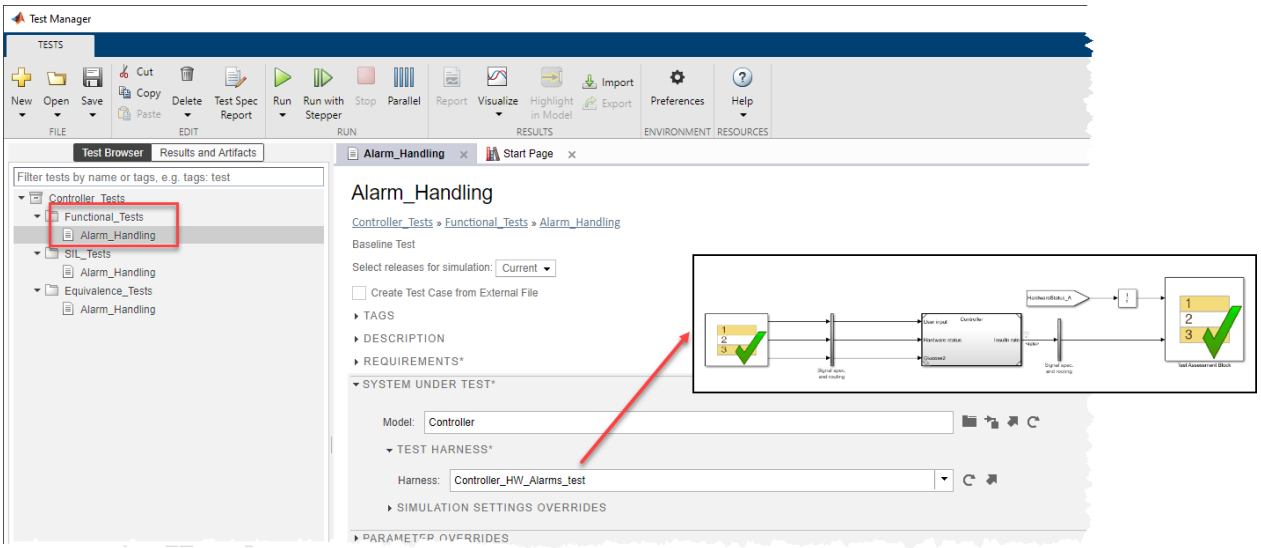
ソフトウェアアイテムがさらに細分化され、ユニットレベルに到達すると、ユニット間の構造、特性、および依存関係が再度明らかになります。ソフトウェア安全クラス C では、各ソフトウェアユニットに対して文書化および検証された設計が必要です。この設計では、ソフトウェアの実装をサポートするために必要なユニットのすべての側面が指定される必要があります。ソフトウェアユニットを表す Simulink モデルまたはサブシステムは、この詳細設計の重要な部分になる可能性があります。Simulink モデルは、ソフトウェアユニットのインターフェイスや他のソフトウェアユニットおよびソフト

ウェアアイテムとの関係を明確に示します。これらは、ブロック線図によって明らかに示されるだけでなく、シミュレーション機能により実証されます。

上述のソフトウェア アーキテクチャの検証原理は、安全クラス C のソフトウェアユニットの詳細な設計情報を含む、さらに詳細な Simulink モデルにも適用できます。ソフトウェアユニットの設計検証では、ユニットがその要件を満たすことを実証する必要があります。関連付けはモデルから要件ドキュメントに向けて、またはその逆方向でも行うことが可能で、変更の発生時に同期させることができます。レポートや強調表示により、Simulink モデルにおける要件リンクのカバレッジの程度を示すことで、検証をサポートできます。

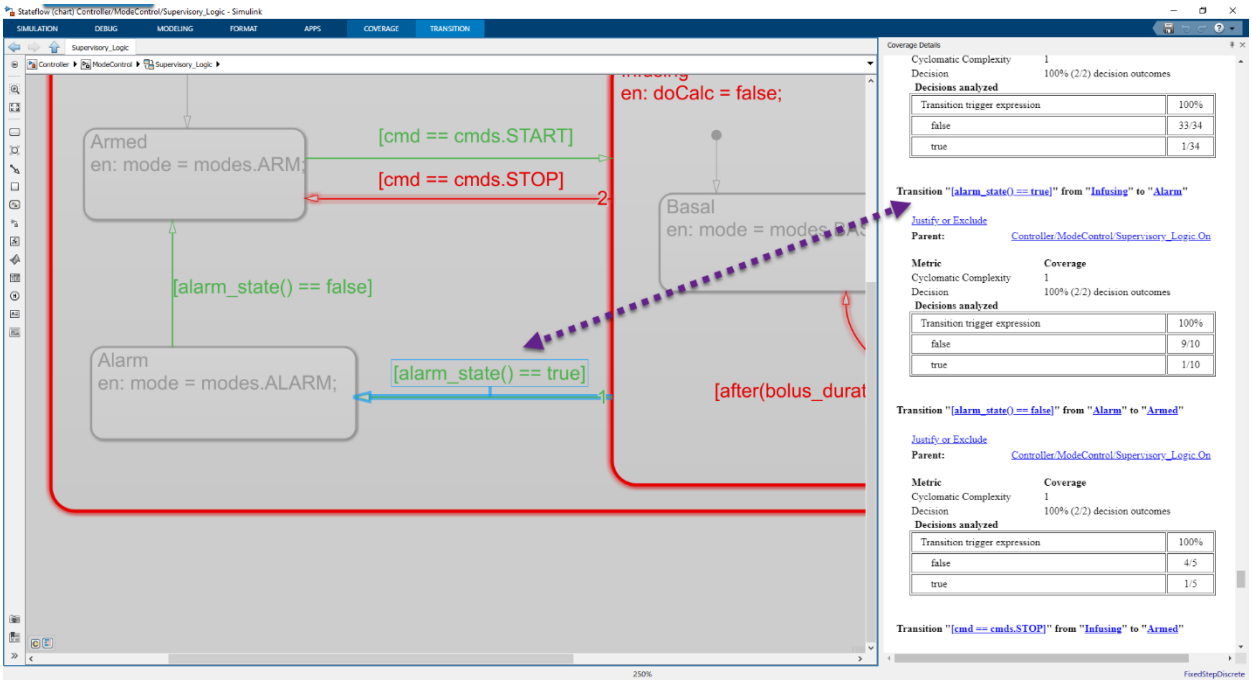
モデリング標準は、ユニット設計の解析に役立ちます。Simulink Check™ に用意されているオプションの 1 つは、IEC 62304 や MISRA-C などの業界標準とモデルの比較です。お客様ご自身で、この自動チェックのリストをカスタマイズおよび拡張することができます。

ソフトウェアユニットの設計検証には、モデル シミュレーションを通した機能検証を含めることができます。これは通常、ソフトウェア ユニット モデルを参照するテストハーネスモデルを作成するか、ソフトウェアユニットのサブシステムへのライブラリリンクを含めることで実現されます。テストハーネスモデルは、さまざまな可能性を考慮した複数のユニット テスト シナリオを実行するように構成することができます。Simulink Test™ は、テストシナリオを管理および実行し、想定される結果を提供するためのツールです。これらのテストケースは、関連するシステムおよびソフトウェア要件に関連付けることができます。



Simulink Test を使用し、テストハーネスモデルを呼び出して設計モデルを実行することにより、アラーム処理テストケースを実装。ハーネス内。Test Sequence ブロックでテストベクトルを入力し、Test Assessment ブロックで出力を確認。

Simulink Coverage™ は、モデルをインストルメント化し、シミュレーション中に実行、データ範囲、さまざまな種類の論理判断とブランチのカバレッジを収集して、ユニット テスト シナリオの完成度を評価するために使用できます。



カバレッジ結果を強調表示したモデル。左側の動作 (Stateflow チャート) はアラームが検知されたことを示し、右側の HTML レポートはアーティファクトの概要を示す。組み込みのハイパーリンクにより、テスト対象モデルへのトレーサビリティを提供。

原則として、要件ベースの機能テストケースは、ユニットテスト計画の一環として実行される場合に高いモデルカバレッジを提供する必要がありますが、実際には必ずしもそうとは限りません。たとえば、*入力範囲の確認など、設計の防御面は、機能要件ベースのテストでは実行されない可能性があります。*Simulink Design Verifier™ は、モデル構造に基づいて追加のテストケースを導出するのに役立ちます。その機能の 1 つは、形式的なモデル解析手法を用いて、モデルのカバレッジ オブジェクティブ (100% の改良条件判定カバレッジ (MC/DC) など) を満たすテストケースを作成したり、設計上の制限によりそのようなカバレッジ目標を達成できないことを示したりすることです。このような追加のテストケースは、欠落しているソフトウェアの派生要件の理解および文書化や、設計の修正によるテスト容易性の向上のために使用できます。

ソフトウェアユニットの実装と検証

Simulink Coder および Embedded Coder を使用すると、Simulink モデルを C または C++ プログラミング言語でのソフトウェア実装に変換できます。ソフトウェア実装の詳細は、詳細化されたモデルで表現された詳細設計から推測され、ソフトウェア アーキテクチャは、コードのインターフェイスや、機能およびファイルベースのコードモジュールへのパッケージ化などについて、設計に忠実なものとなっています。コード生成ツールには、モデル要素からコードへの両方向のトレーサビリティが用意されており、モデル要素に関連付けられた要件の説明をコメントとして含めることもできます。これは、そのような機能的特徴やリスク制御策の実装を管理および追跡するための基礎となります。

ソフトウェアへの変換により、ユニットのランタイム性能の制約やレガシコードのインターフェイス、または組織でのコーディング標準への準拠など、追加の派生要件が明らかになる場合があります。これらのアイテムは、詳細設計モデルを修正し、コードを繰り返し生成することで対応できます。この処理は、ユニット検証が成功すると終了します。

ユニット検証の重要な領域として、機能検証があります。ソフトウェアユニットの Simulink モデルで、機能の詳細な設計検証するためのテストシナリオやテストハーネスを、実装機能ユニットの検証を実行するためのベースラインとして使用できます。生成されたコードは、ホスト (開発者がソフトウェアの設計や作成に使用しているコンピューター) でコンパイルされ、テストハーネスモデルの S-Function として呼び出されるため、同じテストベクトルを使用してモデルとコードを直接比較できます (ソフトウェアインザループ (SIL) テストと呼ばれています)。また、Embedded Coder のターゲット サポート パッケージを使用して、ターゲット向けにコンパイルされたコードに対してこのようなテストを実行することもできます。このプロセッサインザループ (PIL) 手法を使用することで、最終的なデバイスの一部としてソフトウェアを実行中であっても、実行可能なソフトウェアが Simulink モデルと同じ動作をすることを検証できます。コードカバレッジは、意図しない機能が導入されていないことを実証するためのテストプロセスの一環として捉える必要があります。Simulink Coverage は、SIL シミュレーションで生成されたコードのインストルメント化をサポートしています。

安全クラス C のソフトウェアユニットの場合、ソフトウェアユニットの採択に追加の基準があります。イベントシーケンス、データおよび制御フロー、変数の初期化など、一部は Simulink の詳細設計モデルの一部として自然に詳細化されます。その例としては、データ可用性に基づくブロックの自動並べ替えと関数呼び出しによってトリガーされる実行、明示的な信号フローとデータ ストア メモリ アクセス、モデル内の信号と状態の明示的または既定の初期化オプションなどがあります。モデルカバレッジをテストの適切な実施対策として使用することで、ソフトウェアのこれらの機能を実行するための機能テストシナリオが作成されます。先に述べたように、実装のランタイム パフォーマンスやコーディングスタイルの側面は、モデルであまり詳細に表現されないことがあり、基準を満たすためにモデルの更新とコードの生成を繰り返し行うことが必要になる場合があります。Simulink Check には、MISRA C:2012 などのモデリング標準への準拠を適用するモデル アドバイザー チェックが用意されています。また、Polyspace® 製品を使用して、手書きまたは生成された C/C++ コードの堅牢性と重大な実行時エラーを解析できます。主な機能として、業界標準 (MISRA や JSF++ など) に対するコードの静的解析と、形式的なソフトウェア解析手法による潜在的ランタイムエラーの検出の 2 つがあります。

機能ユニット検証アクティビティの結果は、Simulink Report Generator™ で取得し、ユニット検証文書パッケージ全体、つまり規制遵守のための技術ファイルに追加できます。

ソフトウェアの統合とテスト

モデルベースデザインにおけるソフトウェア統合プロセスには、2 つの異なるフェーズがある場合があります。(ユニットからアイテム、つまりボトムアップの観点から) 1 番目は、統合タスクが Simulink 内で行われる場合です。モデルの階層構造は、検証済みユニットやその他のアイテムから詳細化されたソフトウェアアイテムを構成するために使用され、コードは階層のどのレベルでも生成できます。Simulink セマンティクスは、インターフェイス、実行率と実行順序、および呼び出しツリーを指定します。より大きなアイテムやアーキテクチャ モデル全体からコードを生成する場合、これらのセマンティクスは変換されます。ソフトウェアユニットのテストについて上述した検証、テスト、およびドキュメンテーションの機能はすべて、モデルの階層構造の高次のレベルに適用できます。また、増分統合時の回帰テスト用に、テストハーネスモデルや Simulink Test の手続を作成および管理できます。このような反復テストを自動化し、再現性を高めることが推奨されます。

1 つ以上のソフトウェアアイテムのモデルから生成したコードを統合し、他のソフトウェアとさらに統合して、ソフトウェアシステムを完成させることがよくあります。このプロセスでは、モデルベースデザイン環境のメリットは得られませんが、この統合段階において、コードの生成元に起因する特別な考慮事項はありません。IEC 62304 に準拠した統合計画であれば、どのようなものでも構いません。継続的インテグレーション (CI) などのツールや手法では、外部で開発されたソフトウェアをモデルベースデザインを使用して生成されたソフトウェアとともに構築するメカニズムを形成する場合があります。

Simulink のモデル履歴機能は、モデルベースデザインで開発されたアイテムのソフトウェア問題解決プロセスの一部にすることができます。モデルを保存するたびに、ユーザーは変更の根拠を示すことができます。これらのエントリは、ソフトウェアの問題解決システムで追跡されたアイテムを参照することができます。

ソフトウェア システム テスト

開発プロセスのこの段階に至るまで、モデルベースデザイン ツールは、要件の完全性と正確性、これらの要件に対する設計とコードの実行に必要なテストケース、およびソフトウェアシステムを構成するソフトウェアアイテムのユニットレベルおよび統合レベルの検証を実証するのに役立つドキュメンテーションに関する重要な洞察を提供してきました。システムテストは、モデルベースデザイン環境外で実行される場合が多いものの、標準規格のセクション 5.7 に準拠した方法で、一般的な組織的慣行に沿って行われます。

システムテスト中は、システム環境の Simulink モデルを使用して、ソフトウェアにスティミュラスを提供することができます。この手法は、専用の決定論的ハードウェアおよびソフトウェアシステムのサポートを受けて行われるのが一般的で、多くの場合にハードウェアインザループ (HIL) テストと呼ばれます。関係する数学の複雑さにもよりますが、HIL システムでは多くの場合、閉ループ制御の性能評価を目的としたシステム環境のリアルタイム エミュレーションを実現できます。HIL テストは Simulink Real-Time™ を使用して実装できます。

ソフトウェア保守プロセス

モデルベースデザインは、変更の影響を受けるソフトウェアアイテムのシュレーションにより、提案されたソフトウェアの変更がもたらす影響の評価を支援します。さらに、モデルは提案された変更をさまざまなソフトウェアアイテムと組み合わせる際の評価に役立つ重要な設計アーティファクトです。変更の実装が決定されると、ソフトウェア保守計画の一環として組織が選択した適切なアクティビティが、上述のソフトウェア開発プロセスに関する推奨事項に沿って実行されます。

ソフトウェアリスク管理

モデルベースデザインは、主に上述の要件とモデル要素を関連付ける機能によって、ソフトウェアのリスク管理をサポートしています。この関連付けと Requirements Toolbox のレポート機能を使用することで、エンジニアは文書化された危害要因からモデル (設計) における制御策、生成されたソフトウェアアイテムにおけるこれらの対策の実装、および最終的にその効果を評価するためのシミュレーション テストまでを追跡できます。トレーサビリティは、コメントを介して生成されたソフトウェアのコードまで拡張され、コード生成の HTML レポートと Simulink および Embedded Coder の [コードに移動] 機能で移動できます。Embedded Coder では、追加の証拠として、ブロックからコードへの詳細なトレーサビリティ レポートを作成することもできます。

MathWorks は、MathWorks のツールにおける既知の主要な問題のリストを公開しています。プロジェクト中にこれらの外部バグレポートを確認し、関心のある項目を異常リストの一部として管理できます。

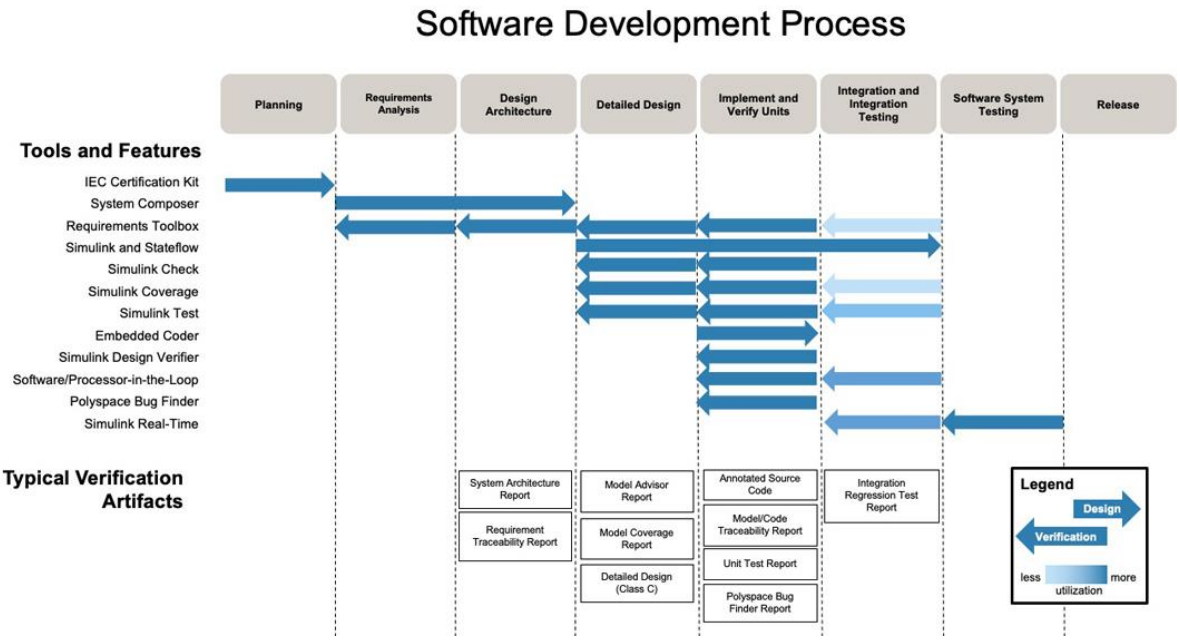
ソフトウェア構成管理

ソフトウェア エンジニアリング プロセスの一環として作成されたモデルやデータは、結果として得られるソフトウェアアイテムと共に識別、管理する必要があります。これらのモデルやデータは、ソフトウェアの設計履歴の一部になり、ソフトウェアの再生成や保守を行うために必要です。開発において生成される一般的なアーティファクトとしては、Simulink モデル、MATLAB® スクリプトや関数、データ辞書、生成された量産コード、S-Function などのユーザー ブロック ライブラリ、シミュレーション入力データ (テストベクトル) および結果、設計ドキュメントやテスト結果などの生成されたドキュメンテーションなどがあります。バージョン番号、保存日、作成者などの構成管理システムの情報は、モデル情報ブロックに表示される情報テキストとして Simulink モデルに組み入れることができます。

まとめ

MathWorks のモデルベースデザイン ツールを使用すると、IEC 62304 のソフトウェア ライフサイクル プロセス要件に準拠することができます。Simulink とその関連ツールは、ソフトウェア開発および保守のプロセスにおいて特に高い価値があります。要件解析、ソフトウェア設計、ユニット実装、およびテストという主要なアクティビティを完了するための機能が、紙ベースの方法論よりもはるかに優れているためです。また、プロセスのアーティファクト (要件ドキュメントなど) とモデルの間の関連付けがサポートされているため、医療機器メーカーは、リスク解析および問題解決のプロセスでシステムおよびソフトウェアモデルとの密接な同期を維持することができます。モデルベースデザイン ツール

には、ソフトウェアとその開発プロセスのドキュメンテーションにも貢献する機能があります。下図は、ツールの一般的な使用シナリオと、ツールで作成する一般的なドキュメンテーション アーティファクトを示しています。



IEC 62304 準拠のソフトウェア開発におけるモデルベースデザイン ツールの適用性。

医療機器向け MATLAB および Simulink の使用の詳細については、
以下を参照してください。

mathworks.com/solutions/medical-devices.html

参照

1. “Caterpillar Automatic Code Generation,” Jeffrey M. Thate (Caterpillar), Larry E. Kendrick (Caterpillar), and Siva Nadarajah (MathWorks), Proceedings of the Society of Automotive Engineers World Congress 2004 (2004-01-0894)
2. “Rapid Deployment of Aerospace Flight Controls,” Edward L. Burnett (Lockheed-Martin Aeronautics), 2006, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html
3. “GM Powertrain Automatic Code Generation Process,” 2005, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html
4. “Medrad Ensures Safety of MRI Vascular Injection Pump Using MathWorks Tools,” 2004, http://www.mathworks.com/company/user_stories/userstory6313.html
5. “Alstom Generates Production Code for Safety-Critical Power Converter Control Systems,” 2005, http://www.mathworks.com/company/user_stories/userstory10591.html
6. “Achieving Six Sigma Software Quality Through the Use of Automatic Code Generation,” Bill Potter (Honeywell International), 2005, http://www.mathworks.com/programs/techkits/pcg_tech_kits.html
7. IEC 62304, “Medical Device Software – Software Life Cycle Processes,” International Electrotechnical Commission, Edition 1.0b, 2006
8. “Control Algorithm Modeling Guidelines Using MATLAB, Simulink, and Stateflow,” MathWorks Automotive Advisory Board - Version 2.1, 2007
9. “Weinmann Develops Life-Saving Transport Ventilator Using Model-Based Design,” https://www.mathworks.com/company/user_stories/weinmann-develops-life-saving-transport-ventilator-using-model-based-design.html, 2013
10. Solis-Lemus J A, Costar E, Doorly D, Kerrigan E C, Kennedy C H, Tait F, Niederer S, Vincent P, and Williams S, “A Simulated Single Ventilator / Dual Patient Ventilation Strategy for Acute Respiratory Distress Syndrome During the COVID-19 Pandemic,” Royal Society Open Science 7:200585, August 2020
11. *Software Requirements*, Karl Weigers, 2nd Edition, 2003, Microsoft Press
12. “Understanding and Controlling Software Costs,” Barry W. Boehm and Philip N. Papaccio, 1988, IEEE Transactions on Software Engineering 14(10), pages 1462-1476
13. MISRA C:2012.The Motor Industry Software Reliability Association:Guidelines for the use of the C language in critical systems, 2012
14. Embedded Coder で生成された C コードの MISRA C 準拠については、http://www.mathworks.com/support/solutions/en/data/1-1IFP0W/?solution=1-1IFP0W で確認できます。
15. “Certify embedded systems developed using Simulink and Polyspace products to IEC 61508 and ISO 26262,” http://www.mathworks.com/products/iec-61508/
16. “Sound Verification Techniques for Developing High-Integrity Medical Device Software,” Jay Abraham (MathWorks), Paul Jones (FDA/CDRH), and Raoul Jetley (FDA/CDRH), Proceedings of the Embedded Systems Conference, San Jose, CA, 2009
17. MathWorks の外部バグレポート データベースは、http://www.mathworks.com/support/bugreports からアクセスできます。
18. “Configuration Management of the Model-Based Design Process,” Gavin Walker (MathWorks), Jonathan Friedman (MathWorks), and Rob Aberg (MathWorks), Proceedings of the Society of Automotive Engineers World Congress 2007 (2007-01-1775)