User's Manual

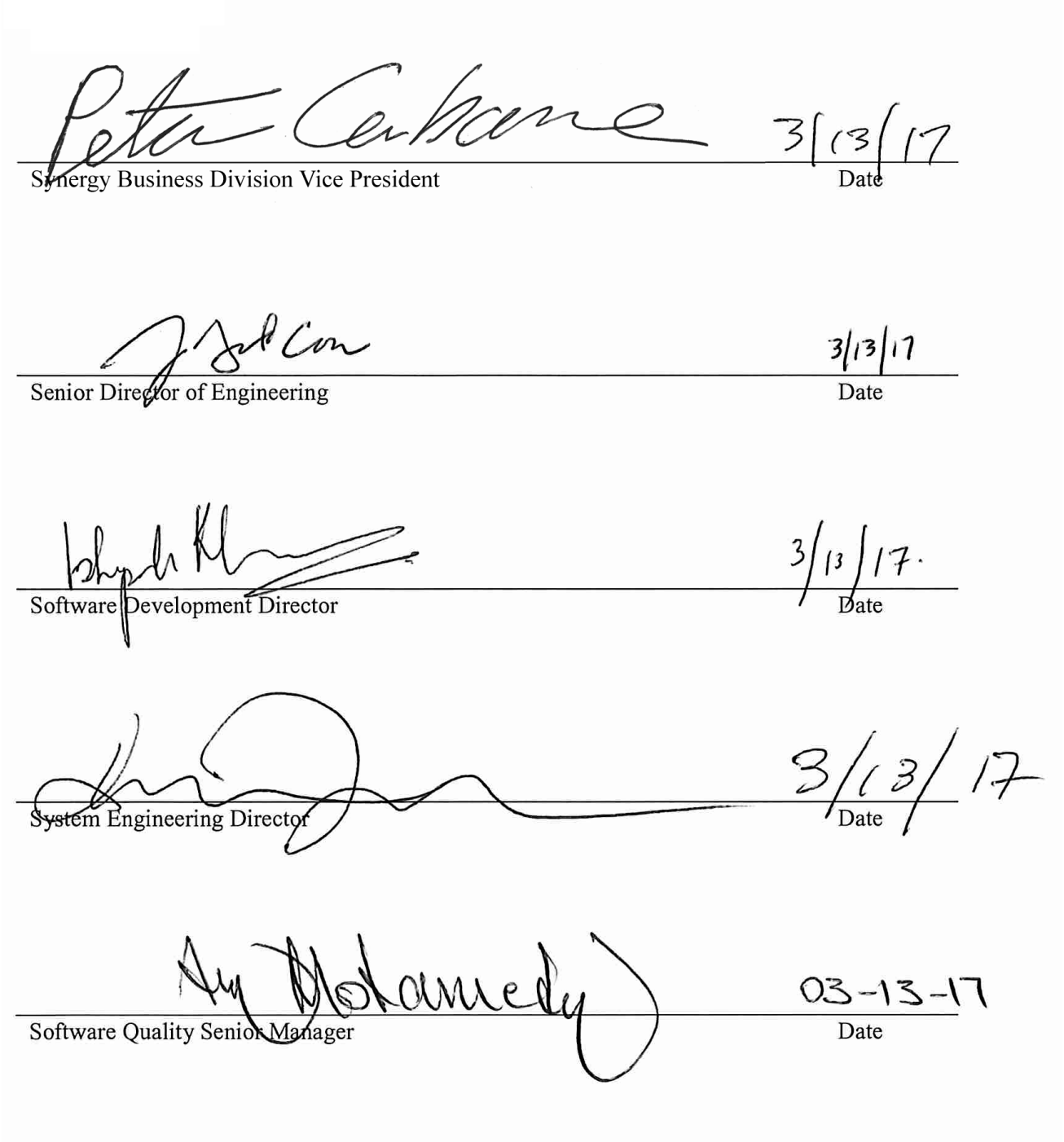
Synergyソフトウェア
品質ハンドブック

Renesas Synergy™ Platform
Synergyソフトウェア
ソフトウェア品質保証

ルネサス エレクトロニクス
www.renesas.com

Rev. 1.18 2017.09

承認


3/13/17
Synergy Business Division Vice President
Date
3/13/17
Senior Director of Engineering
Date
3/13/17.
Software Development Director
Date
3/13/17
System Engineering Director
Date
03-13-17
Software Quality Senior Manager
Date

本資料は英語版を翻訳した参考資料です。内容に相違がある場合には英語版を優先します。資料によっては英語版のバージョンが更新され、内容が変わっている場合があります。日本語版は、参考用としてご使用のうえ、最新および正式な内容については英語版のドキュメントを参照ください。

目次

1. 要旨

本ハンドブックは、Renesas Synergyソフトウェアに適用されるソフトウェア品質保証（SQA）活動のガイドラインです。本ハンドブックは、Synergyプラットフォームを使用して最終製品を開発する開発者、マネージャー、ベンダ、および品質チームに提供されます。

品質の高いソフトウェアを開発するためには関係者全員の協力が欠かせません。ソフトウェアシステムの開発を経験したことのある人なら誰でも、品質の高いソフトウェアを作成する作業が想像するよりはるかに難しいことを理解しているでしょう。機能性の高いソフトウェアの開発には多くの落とし穴があり、リスクがともないます。

ソフトウェアの開発の難しさと課題の多くは、その構想を描くことや物理的特性を記述できないというソフトウェアの非実体性（Intangibility）から生じます。ソフトウェア開発のプロセスは、多くの確立されたエンジニアリング手法を利用しますが、定義されていないプロセスもあります。ソフトウェア開発者は、技術と技能を適切に組み合わせることができますが、個人差もあります。Renesasは、ソフトウェアの品質を形成する要件を共有することと、品質を達成するための統一された手法を用いることが、開発を成功させる上で極めて重要と考えています。このような考えと手法がプロジェクトを成功させる上で重要な役割を果たします。

本ハンドブックの中で説明するSQAの利点はとても現実的です。私たちは、エンジニアリングプロセスにおいて、より良い品質と一貫性のあるソフトウェアを目指して努力し、開発のリスクを低減させます。ソフトウェア開発は決して簡単になることはないかもしれません。それでも、開発プロセスと品質チェック方法を改善すれば、成功度を著しく上げることができます。ソフトウェア品質保証プログラムを実行することで、ソフトウェア製品のエラーの数を大幅に削減でき、システム開発のより早い段階で多くのエラーを確実に検出することができます。

2. 定義

2.1 ソフトウェア (Software)

ソフトウェアとは「コンピュータのプログラム、手続き、規則であり、関連するドキュメントとコンピュータシステムの運用に関係するデータ」を指します。ソフトウェアの定義は、コンピュータ（マイクロコントローラ）のコードだけでなく、付随するドキュメントも含みます。この定義はコード内のインラインコメントのみならず、要求仕様、設計ドキュメント、ユーザマニュアル、および保守ガイドも含みます。つまり、良いソフトウェアとは、プログラムコードだけでなくドキュメントも良いものである必要があります。

2.2 ソフトウェア品質 (Software Quality)

Renesasはソフトウェアが、定義された要求に確実に準拠させることのできるプロセスを確立しました。これはソフトウェアの使用適合性とは異なります。定義された要求は、品質のレベルを決定する役割の基礎となります。障害が発生し、システムが不十分に動作しているときに何をするかを決めることよりも、初めに要求を確立することに重点が置かれています。この品質定義は、2つの行動に影響します。

1. 顧客の要望に製品が正しく合致することを確実にする。
2. 製品の品質を確かなものにするための合理的な手順が実行されたことを検証する。

この品質のゴールは、品質の絶対的なレベルを達成することではなく、指定された要求を満たすことです。このゴールは、100%の信頼性と不具合ゼロを保証することではありません。最終製品に要求されるレベルの品質を達成するために、あらゆる合理的な手順がライフサイクル中に行われたことを検証することです。

2.3 ソフトウェア品質保証 (SQA：Software Quality Assurance)

SQAとは「ソフトウェアが、確立された技術的要求に適合することを確信させるための、体系的な一連の行動」です。この定義には以下が含まれます。

1. ソフトウェア開発ライフサイクル全体における、ソフトウェアの品質に寄与しうるすべての必要な活動
2. ソフトウェア品質の目標を達成するための計画の体系的な実行
3. 実際面でプロセスの効果を実証するための技術開発

Synergyソフトウェアに対するRenesasの品質計画には、初期段階における市場要件の確認から、ソフトウェアの検証および保守に至るまでのソフトウェア開発プロセス全体が含まれます。最も重要な点は、ソフトウェアの変更が、ソフトウェアの一貫性と後方互換性 (Backward Compatibility) を確保しつつ、市場要件を満たし、ソフトウェアアーキテクチャを保持することです。ソフトウェアの変更は、その正確性と完全性がモニターされ、検証済みの各リリース製品を対象にトレースされます。これには不良解析と不良検証が含まれます。そのための仕様テスト、手続き、分類、テストの種類、およびテスト方法を決める必要があります。このようなテスト志向のプログラムで注意を払うべき重要な点は：

ソフトウェア製品を**テストすることで品質をもたらすことはできない**、ソフトウェア製品に**品質があらかじめ組み込まれている**ということです。

2.4 品質要因 (Quality Factors)

品質の要因は、性能課題、設計課題、または適用課題によって、グループ分けされます。これらは一般的な分類であり、ソフトウェアプロジェクト品質の性質を本質的に特徴付けます。

性能品質要因 (Performance quality factors) は、ソフトウェアがどの程度上手く機能するかの特性を示します。性能要因は、効率、整合性、信頼性、生存性、および使い勝手です。

設計品質要因 (Design quality factors) は、ソフトウェア設計の特性を示します。設計要因は、正確さ、保全性、および検証性です。

適応品質要因（Adoption quality facftors）は、ソフトウェアの適応特性を示します。適応要因は、拡張性、柔軟性、相互作用性、移植性、および再利用性です。

1. **効率（Efficiency）**- ソフトウェアがある機能を実行するために必要な演算用資源とコードの量
2. **整合性（Intergrity）**- 許可されない人物によるソフトウェアまたはデータへのアクセス制御の程度
3. **信頼性（Reliability）** - 与えられた時間内で、ソフトウェアが故障せずにその意図された機能を実行する範囲
4. **生存性能（Survivability）**- システムの一部が動作不能な場合に、ソフトウェアが評価の基準となる重要な機能を実行およびサポートする範囲
5. **使い勝手（Usability）** - ソフトウェアを学習し、運用し、インプットを用意し、およびアウトプットを解釈するために必要な取り組みの程度
6. **正確さ（Correctness）** - ソフトウェアが仕様を満たし、ユーザの目的を実現する程度
7. **保全性（Maintainability）** - 運用中のソフトウェアでエラーを発見し、修正するために必要な取り組みの程度
8. **検証性(Verifiability)**- ソフトウェアが、その意図された機能を実行することをテストし、検証するために必要な取り組みの程度
9. **拡張性（Expandability）** - 現在の機能を強化する、または新たな機能もしくはデータを追加することで、ソフトウェアの機能または性能を向上するために必要な取り組みの程度
10. **柔軟性（Flexibility）** - 運用中のソフトウェアを変更するために必要な取り組みの程度
11. **移植性（Portability）** - ソフトウェアを別の環境で使用するためにソフトウェアを移動する際に必要な取り組みの程度
12. **再利用性（Reusability）** - ソフトウェアを他のアプリケーションで使用できる程度
13. **相互運用性(Interoperability)**- あるシステムのソフトウェアを別のシステムのソフトウェアと組み合わせるために必要な取り組みの程度

SQAは上記の品質要因を使用して、プロジェクトがどの程度、品質目標を満足しているかを決定します。

3. ソフトウェア開発ライフサイクル活動 (Software Development Life Cycle Activities)

開発プロセスは多くのステージで構成され、それらのステージは慎重に検討された複数のステップにさらに分解されます。個々のステージの名称や、結果として作成されるドキュメントの形式にかかわらず、すべてのソフトウェア開発ライフサイクルは似たような特徴を持ちます。

ステージは以下のとおりです。

要求 (Requirement)	対象の環境を理解し、要求を収集して解析することが含まれます。
設計 (Design)	システムの論理的な側面と物理的な側面の両方が含まれます。
実装 (Implementation)	設計の詳細を定義し、システムのコードを書き、テストを行い、インストールすることが含まれます。
リリース (Release)	システムを生産で使用すること、ならびに修正、改良、および追加の保守および管理が含まれます。

3.1 計画フェーズ (Planning Phase)

計画フェーズではプロジェクトの基本的な構造の決定、プロジェクトの実現性とリスク評価、および適切な管理手法と技術手法の記述をおこないます。

プロジェクト計画ステージのアウトプットは、ソフトウェア開発プラン (SDP) 、ソフトウェア検証プラン (SVP) 、ソフトウェア設定管理プラン (SCMP) 、およびソフトウェア品質保証プラン (SQP) であり、この後に来る要求フェーズのために活動、責任リストを準備し、さらにアウトプットフェーズ向けの作業を高いレベルで見積もることです。

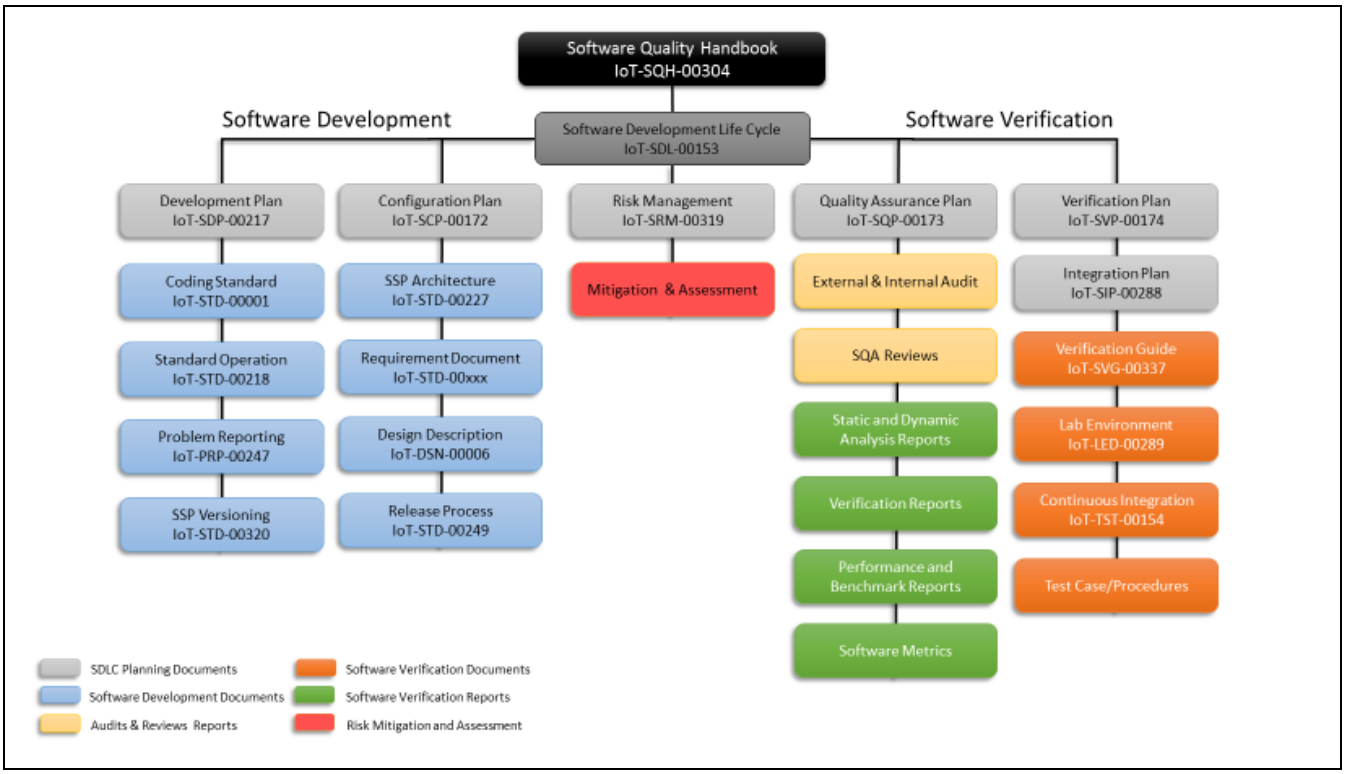


図1　Renesas Synergyソフトウェア SDLC文書

3.2 ソフトウェア開発プラン（Software development Plan）（IoT-SDP-00217）

SSPソフトウェア開発プランにおいて、ソフトウェア開発プロジェクトは管理可能なタスクに分割され、階層的で細密な詳細に整頓されて記述されます。ソフトウェア開発プラン（SDP）は、プログラム開発に関連するすべての技術的および管理的活動を特定します。以下の項目が規定されます。

- 活動の記述
- 活動の成果物および関連する完成基準
- 従来の活動から得られた前提となる成果物
- 活動の相互関係性

開発プロセスのアウトプットドキュメントは以下のとおりです。

- ソフトウェア要求ドキュメント（SRD）
- ソフトウェア設計記述（SDD）
- 要求トレーサビリティマトリックス（RTM）
- ソースコード
- 実行オブジェクトコード
- 開発単体テスト
- 開発単体テスト結果

注：トレーサビリティは、市場ユースケース→市場要求ドキュメント（MRD）→ソフトウェア要求ドキュメント（SRD）→ソフトウェア設計記述（SDD）→ソースコードセクションまたは実行オブジェクトコード、検証ケースおよび結果です。

すべてのソフトウェア製品要求は、要求定義ステージ中に開発され、1つまたは複数の市場要求から発生します。各市場要求の最小の情報は、タイトルと文章による記述から成りますが、追加情報や外部ドキュメントへの参照を含めることもあります。

3.3 ソフトウェア検証プラン（Software Verification Plan）（IoT-SVP-00174）

SSPソフトウェア検証ドキュメントには、要求検証を実行するためのプランとガイダンスが記載されています。このドキュメントには、ドキュメント中に定義されるプロセスに適合していることが示され、さらにハイレベル（機能テスト、統合テスト、性能テスト、および回帰テスト）とロウレベル（単体テスト）のソフトウェア要求にソフトウェアが適合していることが示されています。

検証プロセスのアウトプットドキュメントは以下のとおりです。

- ソフトウェア検証ケースおよび手続き（SVCP）
  - 機能モジュールテスト
  - 結合テスト
  - 性能テスト
  - 回帰テスト
- ソフトウェア検証結果（SVR）：
  - レビュー要求、設計、コード、テストケース、およびテスト結果
  - 実行オブジェクトコードのテスト
  - レビュー要求のカバレッジ
  - コードカバレッジ解析

3.4 ソフトウェア構成管理プラン（Software Configuration Management Plan）（IoT-SCP-00172）

ソフトウェア設定管理プラン（SCP）は、ソフトウェア設定確認、ソフトウェア設定管理、ソフトウェア設定状態、計算記録、およびソフトウェア設定監査を開始および維持するための運用プランを示します。

ソフトウェア設定管理活動に含まれるのは、設定確認（Configuration Identification）、変更管理（Change Control）、設定システム（Configuration System）におけるベースライン確立、およびソフトウェアライフサイクルデータのアーカイブ保存です。SCPはこれらの活動がどのように行われるのかを詳しく記述します。このプランはまた、プロジェクトメディア、設定状態計算、および設定監査の保存、処理、および納入の方法も扱います。

本プランによって確立されたSCM（Software Configuration Management）要求は、製品の生涯を通して、また今後のソフトウェアリリース活動を通して、すべてのソフトウェア製品/開発フェーズに適用されます。

Renesas設定システムは、ソフトウェアデータリポジトリの設定された部分を管理します。この設定システムは、進化するソフトウェアシステムの複数の変形を管理可能であり、ソフトウェアビルドでどのバージョンが使用されたかを追跡し、ユーザが定義したバージョン仕様に基づいて個々のプログラムまたはリリース全体のビルドを行い、サイト固有の開発方針を実行します。

Renesas設定システムのツールセットは、下記のために使用されます。

- ベースラインソフトウェアとドキュメントを追跡するためのメカニズムの提供
- ソフトウェアまたはドキュメントの各変更の履歴と記録を維持
- 行動要求を使用することで、ベースラインに対する変更を開始
- ソフトウェアのリリースされたすべての設定の特定および管理

3.5 ソフトウェア品質保証プラン（Software Quality Assurance Plan）（IoT-SQP-00173）

SSPソフトウェア品質保証プランは、開発プロセス保証（DPA）が使用する活動に対応しておりすべてのソフトウェア開発が、対応するソフトウェアプランに記述された認証済みの方法に従って行われるようにします。

品質保証プロセスのアウトプット文書は以下のとおりです。

- ソフトウェア品質保証記録（SQAR：Software quality asssurance records）
- ソフトウェア適合レビュー（SCR：Software conformity review）
- ソフトウェアリリースノート（SRN：Software Release Notes）

3.6 ソフトウェア開発フェーズ（Software Developmemt Phase）

ソフトウェア開発フェーズは以下で構成されます。

1. 要求仕様ステージ。コンピュータプログラムが満たすべき要求を特定します。
2. 設計記述ステージ。ソフトウェアの設計を決定します。
3. ソフトウェア実装ステージ。ステージ1と2を実行します
4. ソフトウェア開発検証ステージ。ステージ3が要件を満足するかを検証します

最初の2ステージでプロジェクトの目標の一覧、システム機能仕様、および設計の制限が作成されます。

3.6.1 ソフトウェア要求仕様（Software Requirements Specification）（IoT-STD-00xxx）

要求収集プロセスはそのインプットとして、市場要求ドキュメント（MRD）中で特定される要求を使用します。各市場要求は整理されて、1つまたは複数のソフトウェア要求になります。

要求は、本ステージの主要ドキュメントである要求ドキュメントと要求トレーサビリティマトリックス（RTM）の中で詳細に記述されます。要求ドキュメントには、各要求の完全な記述が収められています。必要に応じて図および外部ドキュメントへの参照が含まれます。

ソフトウェアリリース計画フェーズでは、ソフトウェア要求レビューが行われ、またソフトウェア要求仕様が作成されます。ソフトウェア要求レビューはSRD評価の構成要素です。SRDで記述されている要求の適切

さ、技術的実行可能性、および完全さを確実にするためにソフトウェア要求レビューは行われます。ソフトウェア要求レビューを行う目的は、SRDを評価して、それが運用フェーズおよび保守フェーズにおいて完全、検証可能で、一貫性があり、保守可能、変更可能、追跡可能、および利用可能であることを確実にするためです。このレビューは、ソフトウェア設計を完成させるための詳細情報が十分に入手できることを保証します。ソフトウェア要求レビューの結果はドキュメント化され、それには特定されたすべての欠陥の記録と是正措置の計画およびスケジュールが含まれます。これら欠陥を修正すべくソフトウェア要求レビューが更新された後は、ドキュメントは設定管理下に置かれます。またベースラインは、ライフサイクルと通じて、ソフトウェア設計や他の用途のために使用されます。SRDに対する追加の変更は、ソフトウェア設計とその実装期間中は許可されています。

SSP（Synergy Software Package）のSRDは、市場要求ドキュメントおよびRenesas Synergy MCUファミリデータシートに基づいて作成され、以下を確実にします。

1. 全要求がソフトウェアの検証可能な動作を独自に特定
2. 要求は設計との間に前方および後方互換性がある
3. 要求はハードウェア機能に基づいて、必要なソフトウェアインタフェースを特定
4. 要求は内部および外部通信インタフェース用データ結合を特定
5. 要求は内部および外部制御結合を特定
6. 安全要件とエラー管理シナリオは、必須アプリケーションが運用可能であるように満たす

3.6.2 ソフトウェア設計記述（Software Design Description）（IoT-STD-00006）

設計ステージでは、初期インプットとして、承認された要求ドキュメントで特定された要求が使用されます。各要求に対して、1つまたは複数の設計要素が生成されます。

設計要素は、求められるソフトウェア機能を詳細に記述します。設計要素は、機能階層図、画面レイアウト図、ビジネス規則表、ビジネスプロセス図（business process diagram）、疑似コード、および完全なデータディクショナリーを備えた完全なエンティティ関係図（complete entity-relationship diagram）を含みます。これらの設計要素は、ソフトウェアを十分詳細に記述することで、プログラマが最小の追加インプットでソフトウェアを開発できるようにさせるものです。

ソフトウェア設計記述（SDD：Software Design Description）は、すべてのソフトウェア要求（SRD）の検証可能な設計詳細を捕らえ、必要な機能だけが実装され、ドキュメント化されるようにします。

予備デザインレビューが、機能仕様ステージの最後に実施されます。予備デザインレビューでは、詳細設計の前段階として、予備設計の技術的適切さが評価されます。このレビューでは、選択された設計技法の技術的適切さが評定されます。また、SRDの機能要求と性能要求との設計互換性がチェックされ、ソフトウェア、ハードウェア、およびユーザの間のインタフェースの存在と互換性が検証されます。

要求を出すすべての組織と設計によって影響を受ける組織は、このレビューに代表者を参加させます。このレビューで特定されたすべての欠陥の記録と欠陥の是正措置の計画・スケジュールが結果のドキュメントに含まれます。更新されたSDDドキュメントは構成制御（configuration control）下に置かれ、詳細ソフトウェア設計作業のためのベースラインを確定します。詳細設計、実装、またはテスト期間中に高レベル設計に対する変更が必要になった場合、これらの変更は設計ドキュメントに含められます。さらにこれらの変更が与える影響を見極めるための適切なレビューが行われます。

クリティカルデザインレビュー（crotical design review）が、詳細ソフトウェア設計フェーズの最後に実施されます。クリティカルデザインレビューでは、実際のコーディングが始まる前に、詳細設計の技術的適切さ、完全性、および正確さが評価されます。クリティカルデザインレビューの目的は、ソフトウェア設計記述に書かれた詳細設計が受容可能かを評価し、詳細設計がSRDの要件を満たしていることを確認することです。製品が相互作用する必要のある他のソフトウェアおよびハードウェアとの互換性をレビューすること、そして製品設計の技術的リスク、コストリスク、およびスケジュールリスクを評定することです。

要求を出す組織または設計によって影響を受ける組織は、このレビューに参加します。このレビューで特定されたすべての欠陥の記録と、欠陥の是正措置の計画・スケジュールが結果のドキュメントに含まれます。

更新されたSDDは構成制御（configuration control）下に置かれ、次のフェーズである実装とコーディングのためのベースラインを確定します。

ソフトウェア設計を作成する際は、以下の目標も設定されます。

1. すべてのインタフェース（ハードウェア、ソフトウェア、およびシステム）に概念的ソフトウェア製品アーキテクチャを提供
2. すべてのレイヤー（BSP、HAL、アプリケーションフレームワーク、インタフェース等）向けの独自ソフトウェアアーキテクチャ
3. ソフトウェアコンポーネントの初期化
4. 統計的決定のための方程式および計算論理を使用した手順の記述
5. データフローと制御の結合に関し、標準の設計原理に従う
6. 共通の業界設計パターン - 複数のSynergy MCUグループをサポートするためのコンポーネントドリブン設計
7. Renesas Synergyソフトウェアコーディング基準を満たす標準API記述、データ構造手法
8. 明確なエラー管理
9. モジュール間、プロセッサ間、コンポーネント間のインタフェースの定義
10. メモリへの要求とパーティショニングの詳細
11. 通信プロトコルの記述
12. 設計の制限

3.6.3 ソフトウェア実装（Software Implementation）

設計ドキュメントが完成し受領されると、RTMが更新され、各設計要素が具体的な要求と正式に関連していることが示されます。設計ステージのアウトプットは設計ドキュメントです。1つの設計要素に対して、1つもしくは複数のセットのソフトウェア中間生成物（artifact）が作成されます。ソフトウェア中間生成物（artifact）に関連する各機能セットに対し、適切な単体テストケースが開発されます。

開発ステージのアウトプットに含まれるのは、事前にドキュメント化された要求と設計要素を満足するソフトウェアの完全な機能セット、ソフトウェアの正確さと完全性を検証するのに使用される単体テストケース、および更新済みのRTMです。

SSP（Synergy Software Package）コード実装はC言語で行われます。ソフトウェア開発者は、開発環境（保全性および使い勝手）に関し、ソフトウェア開発者ガイド（Software Developer Guide）に従います。

すべてのソフトウェアコンポーネントと中間生成物（artifact）は、セキュアなバージョン管理（整合性と信頼性）を用いて維持されます。新規開発と変更は、Renesas Synergyソフトウェアコーディング基準に準拠する必要があります。コーディング基準に準拠しているかどうかは、静的コード解析（使い勝手と移植性）を用いてチェックされます。さらに、新規開発されたコードや変更されたコード、またはそれらに関連する中間生成物（artifact）は、関連開発者レビュー（peer-review）と独立したレビューの対象となり、変更が要求、設計、および検証結果まで追跡可能であることを確認します（互換性と正確さ）。

3.6.4 ソフトウェア開発検証（Software development verification）

ソフトウェア開発検証は要求ベースのテスト方法です。基本的に、すべての開発検証ケースは少なくとも1つのソフトウェア要求まで追跡され、またすべてのソフトウェア要求は1つの検証ケース（100%の要求カバレッジ）まで追跡され、検証ケースを実行することで100%のデータと制御の結合（coupling）がもたらされます。この目標が達成されるのは、検証ケースがソフトウェア要求まで完全に追跡可能で、かつ検証ケースが機能化されたコードに対して実行され、カバレッジデータが収集されたときです。

ソフトウェア開発終了の基準は以下のとおりです。

1. すべてのSRD項目が少なくとも1つのSDD項目まで追跡される。
2. すべてのSRDとSDDがレビューされ、ベースライン化された。
3. すべての計画された要求が実装された。

4. すべての実装は、少なくとも1つのSDD項目まで追跡可能である。
5. すべての要求は検証ケースまで追跡可能であり、検証ケースは追跡元の要求を完全に検証する。
6. すべての実装がRenesas Synergyソフトウェアコーディング基準に準拠している
7. すべてのソフトウェアコンポーネント、仕様、および検証がバージョン管理され、ベースライン化された。
8. すべての検証ケースが実行され、合格/不合格基準を満たしている。
9. すべてのテストケース、検証結果、およびカバレッジ報告がレビューされた
10. すべての欠陥が報告され、処置された。
11. すべてのソフトウェアコンポーネントが、警告（Warning）無しにコンパイルされ、リンクされた。
12. 100%の要求カバレッジが、100%のデータ結合カバレッジと制御結合カバレッジをもたらしている。

3.6.5 コード運用/保守基準（Code Operation/Maintenance Standards）

保守には修正と強化の2種類があります。修正は、ソフトウェア内で見つかった不良を正すか、もしくは環境の変化により必要になった変更を組み込みます。強化は、要求仕様に何らかの機能を追加します。運用/保守基準を考えるとき、予防的保守として知られる新しい種類の保守を考慮する必要があります。

プログラムまたはモジュールが予防的保守の対象候補として特定されると、ピアレビューが行われ、冗長コードがないこと、そして判定の複雑度が許容範囲内にあることを確認します。

3.6.6 基準（Standards）

開発されたすべてのコードはRenesas Synergyソフトウェアコーディング基準に準拠しています。基準とは、「ドキュメント化された合意事項であり、規則、ガイドライン、または特徴の定義として一貫して使用される技術的仕様またはその他の正確な基準を含んでおり、資料、製品、プロセス、およびサービスがそれらの目的を満たすことを確実にする」ものです。

3.6.7 レビュー（Review）

ドキュメントまたはコードの初版はフルレビューの対象となり、変更は部分レビューの対象となります。レビューの目的は、システムソリューションに対する概念的および技術的手法を評価し、プロジェクトのために前もって定義された品質要因が満足されていることを確認することです。ピアレビューは、ソフトウェア製品内でエラーとして現れる前に問題を特定する取り組みをします。

3.6.8 監査（Audit）

監査（Audit）とは、ドキュメントおよび関連する開発方法を調べ、プロセスとそのドキュメントがSQAプランならびに組織の方針と手続きに適合していることを検証するものです。

3.7 ソフトウェア設定管理フェーズ（Software Configuration Management Phase）

本項では、ソフトウェア設定管理（SCM）とコード管理に関する一般的事項を述べます。問題報告と是正措置にも言及します。

3.7.1 問題報告および是正措置（Problem REporting and Corrective Action）（IoT-PRP-00247）

Renesasは全てのソフトウェアに対して、ソフトウェアの問題報告と是正措置に関する正式な手続き方法を確立しています。ソフトウェアの不具合、誤作動、欠陥、逸脱、不良材料および装置、ならびに不適合を測定し、ただちに特定します。問題報告システムはソフトウェア構成管理手続きと連携し、これらの問題を正式な処理で確実に解決します。

ソフトウェアの開発中または運用中に発生した問題は、ソフトウェア、ハードウェア、もしくはシステム運用内の不具合による可能性があります。発生する可能性のある不具合、不具合元、および検出方法の数が多大なため、ソフトウェアの不具合を監視するシステムが必要です。

ソフトウェア問題報告および追跡システムの目的は以下のとおりです。

- 不具合がドキュメント化され、修正され、記憶されることを確実にする。
- 不具合の妥当性が評定されることを確実にする。
- ソフトウェア構成に対する変更を行う前に、すべての不具合修正がレビューチームもしくは変更管理委員会による承認を受けることを確実にする。
- 不具合修正プロセスの測定を円滑にする。
- 設計者とユーザに不具合の状態を通知する。
- 不具合修正の優先順位を決める方法と、適切な措置のスケジュールを作成する方法を提供する。
- ソフトウェア不具合の状態に関する知見を管理者に提供する。
- ソフトウェアの品質と信頼性の測定および予測のためのデータを提供する。

問題報告およびソフトウェアに対する変更提案を行う際には、標準フォームまたは標準ドキュメントを利用することを推奨します。標準フォームまたは標準ドキュメントには、最低限、下記の項目が含まれます。

- 問題と是正措置提案の記述
- 変更を実行するための認可
- 変更により影響を受ける可能性のあるすべての項目のリスト
- 変更のために必要なリソースの見積もり
- 問題報告と問題解決の作成と処置に関わる担当者が特定される。
- 識別番号と日付
- レビューで判明したことは、問題追跡システム内でアクションアイテムとして報告されます。アクションアイテムは、レビューされる中間生成物（artifact）に対して具体的です。アクションアイテムは、何をするべきかを明確に指定します

3.7.2 是正措置手続き（Corrective Action Procedures）

是正措置手続きは、ソフトウェアの不一致を修正するプロセスに関係します。すべての是正措置はソフトウェア開発によってサポートされます。是正措置では、開発者の生産性と創造性が妨げられないよう、開発者に十分な自由裁量が与えられます。是正措置のタイミングが悪かったり、不適切に行われたりすると、ソフトウェアのコストと信頼性に重大な影響を起こす可能性があります。システムが実装されるまでソフトウェアエラーが修正されないままだと、ソフトウェア開発の最中に発見されたエラーよりも、修正にはるかにコストがかかります。是正措置プロセスは、ソフトウェア開発計画フェーズ（Software Development Planning phase）の初期に確立されます。ソフトウェアの欠陥をいち早く検出し、早い段階で修正することの意義は強調してもし過ぎることがありません。

是正措置手続きは、ソフトウェアの不一致と例外に対する体系的な特定と修正を妨げるのではなく、むしろ助けるものです。SCMシステムで確立されたベースラインにより、是正措置手続きの体系的な統合が可能になります。これらの手続きに含まれるのは、文書中の矛盾を特定する手順、提案された変更をドキュメント化する手順、提案された変更の適切さを独立してレビューする手順、ならびに影響を受けるコードおよび相互作用するすべてのモジュールを再テストする手順です。

是正措置手続きは、個々の問題のエラー解析および再発する問題に関する情報について、ユーザにフィードバックを与える仕組みを確立します。逆に言えば、是正措置手続きでは、コンピュータプログラムにエラーが発見された場合、ソフトウェアユーザがプログラム開発者にエラーを知らせる必要があります。それにより、開発者はエラーの全体的な影響を調べ、評価をできるからです。

ソフトウェア開発と使用時に見つかったエラーの解決には、プログラム開発者が最終的な責任を負います。さらに開発者は、エラーがマイナーな変更で修正可能なのか、それともソフトウェアの再検証を必要とする大きな改訂（revision）が必要なのかを決定します。

効果的な是正措置手続きのためには、ソフトウェア設計者、開発者、テスト技術者、ならびにSQAおよび構成管理団体からのインプットが必要です。このインプットにより、元の開発プロセスにおいて何が上手くいかなかったのかが分かります。プロジェクト管理により既存の方法論を再検討することで、そのような不良の

再発を抑える措置が決定できます。特に、ソフトウェア開発ライフサイクル内のエラーが発生しやすい箇所を特定できるようになります。

3.7.3 SCM活動 (SCM Activities)

SCM活動は以下で構成されます。

3.7.4 構成識別 (Configuration Identification)

ソフトウェアと関連するすべてのコンポーネント、ユニット、またはドキュメントは、バージョン管理され、リリースごとにラベルが付けられます。SSPパッケージで使用されるバージョン形式は、「X.Y.Z-成熟レベル+ビルドメタデータ (X.Y.Z-maturitylevel+buildmetadeta) 」ので、それぞれの要素の意味は以下のとおりです。

X：メジャー（アップグレード）リリース番号、Y：マイナー（アップデート）リリース番号、Z：パッチ（バグ）リリース番号

成熟レベル文字列：各リリースは様々なレベルの成熟度があります。たとえば、アルファ、ベータ、カスタム、リリース候補 (RC) 、そして最終的にゴールドまたは一般公開用(GA)

ビルドメタデータ文字列：Renesas内部において、開発チームの外に発行されたリリースごとに増分されます。これは内部用の追跡番号であり、Renesas外部に必ずしも知らせる必要はありません

この業務においてベースラインの概念は重要です。なぜなら、あるプロジェクトに関与している各人にとって、ソフトウェア製品を定義、開発、または変更する際に共通の参照点となるからです。

3.7.5 構成変更管理 (Configuration Change Control)

設定変更管理は、変更プロセスを管理および制御する際に必要な管理手段を提供します。変更を処理する仕組みはSCMプランで定義されます。適切な承認手続きが組み込まれます。各リリースについて、変更管理審査委員会 (CCRB：Change Control Review Board) が変更を監視および承認します。

3.7.6 構成現状記録および報告 (Configuration Status Accounting and reporting)

構成現状記録は、ソフトウェアがソフトウェアライフサイクルを進行する際の、ソフトウェア状態の記録の作成と維持をするために使用されます。設定状態記録 (CSA) は記録システムと考えてもいいかもしれません。

3.7.7 構成監査およびレビュー (Configuration Audits and Reviews)

SCMプロセスでは、各リリースにおいて監査とレビューが行われます。ベースラインがリリースされると、設定項目が監査されます。実行される監査数はリリースされるベースラインによって変動します。監査参加者の役割を含め、監査の基準はSCMプランで設定されます。最低でも、製品ベースラインが確立、変更、またはソフトウェアの新バージョンがリリースされるたびに監査が実行されます。

3.7.8 コード管理 (Code Control)

コード管理は、動作ソフトウェアとその関連ドキュメントの配布と保護、およびその妥当性を確認するために必要な手続きを含みます。コードベースラインが確立されると、動作コードと関連中間生成物 (artifacts) はコンピュータプログラムライブラリに置かれます。SQAは、ソフトウェア変更のためと、コードがベースライン化された後の不注意による変更を防ぐために、適切な管理手段とセキュリティ手段が確実に確立されるようにします。

ドキュメント化された手続きの後に、新規バージョンの実装が続きます。コンピュータプログラムのすべてのバージョンに対して、正確で他と間違えない確認が確実に行われます。ソースコード、オブジェクトコード、または関連資料の変更を記録するための管理手段が確立されます。ソフトウェアライブラリは、コンピ

ュータプログラムと文書に識別番号を割り当て、追跡をおこないます（リビジョンを含む）。ライブラリは、リリースを許可するドキュメントを提供します。ソフトウェアライブラリは、ソフトウェア出荷の際のマーク付け、ラベル付け、および梱包の手配を助け、提出物の配布、目録作成、および構成管理/構成現状記録のログと記録を保持します。インデックスが、プロジェクトファイルを構成するドキュメントを一覧表示します。

3.8 ソフトウェア検証フェーズ（Software Verification Phase）

ソフトウェア検証とは、「（1）「ソフトウェア開発サイクルの任意のフェーズの生成物（product）が、その前のフェーズ中で規定された要件を満たすかどうか」を決定するプロセス。（2）プログラムの正確さの正式な証拠。（3）項目、プロセス、サービス、またはドキュメントが指定された要求に適合するかどうかをレビュー、検査、テスト、チェック、監査、または確立およびドキュメント化する行為」。このアプローチは要求ベーステスト（RBT）として知られます。

3.8.1 仕様ごとの独立検証（Independing Verification per Specification）

検証の目的は、製品、サービス、またはシステム（もしくはシステムかセットの一部）が設計仕様を満足することをチェックすることです。開発フェーズにおいて、検証手続きは、製品、サービス、またはシステムの一部または全部をモデル化またはシミュレーションするための特別なテストを実行します。そしてモデル化の結果をレビューまたは解析します。開発後のフェーズで、この検証手続きは特別に開発されたテストを定期的に繰り返します。これらのテストは、製品、サービス、またはシステムが時間を経過しても最初の設計要求、仕様、および規則を満たし続けることを確認するためのものです。このプロセスは、製品、サービス、またはシステムが開発フェーズの開始時に課された規則、仕様、または条件に適合するかどうかを評価するために使用されます。

3.8.2 要求ベーステスト（Requirement Based Testing）（RBT）

RBTアプローチは、要求が正しく、完全で、あいまいな部分がなく、かつ論理的に一貫していることを確認します。また、膨大な数の潜在的なテストを合理的な数まで減らし、適切な理由で正しい答えが得られるようにテストします。

RBTの全体戦略は、開発ライフサイクルを通してテストを統合し、要求仕様の品質に焦点を当てることです。これは早い段階での不良の検出につながります。そして、結合テストもしくはそれ以降に不良を発見するよりも、はるかに安上がりであることが分かっています。RBTプロセスはまた、単に不良を検出だけでなく、不良の予防にも焦点を当てています。

RBTプロセスを見通すため、テストは以下の8つの作業に分割されています。

1. **テスト完成基準の定義（Define Test Completion Criteria）**テストには具体的な量的および質的目標があります。テストはゴールに達して初めて完了します。（たとえば、不良を検出するためにバリエーションが増やされたすべての機能、およびすべてのステートメントと分岐ベクタが、コード変更を行わない間に一回もしくは1セットで正常に実行された場合にテストは完了します）。
2. **テストケースの設計（Design Test Cases）**論理テストケースは5つの特徴で定義されます。テスト実行前のシステムの初期状態、データベース内のデータ、インプット、予想されるアウトプット、および最終的なシステム状態です。
3. **テストケースのビルド（Built Test Case）**論理テストケースからテストケースをビルドするために必要なことが2つあります。必要なデータを作成して、テストをサポートするためのコンポーネントをビルドすることです（例：ナビゲーションをビルドし、プログラムのテスト部分へ移動する）。
4. **テストの実行（Excute Test）**テスト対象のシステムに対してテストケース手順を実行し、結果をドキュメント化します。
5. **テスト結果の検証（Verify Test Reaults）**テスト結果が予想どおりであることを検証します。
6. **テストカバレッジの検証（Verify Test Coverage）**テストセットの正常な実行により達成された、機能カバレッジとコードカバレッジの量を追跡します。

7. **不良の管理と追跡（Manage and Track Defects）**テストプロセス中に検出された不良は解決まで追跡されます。全体的な不良の傾向と状態に関する統計が維持されます。
8. **テストライブラリの管理（Manage the Test Liability）**テスト管理者は、テストケースとテスト中のプログラムの関係を維持します。テスト管理者は、どのテストが実行され、実行されなかったか、および実行されたテストが合格か不合格かの記録を保持します。

3.8.3 テストおよびソフトウェア開発ライフサイクル（Testing and the Software Development Life Cycle）

ほとんどのソフトウェア開発ライフサイクルでは、大部分のテストはコードが利用可能になった場合にのみ行われます。RBTプロセスでは、ライフサイクルを通じてテストは統合されます（図2）。

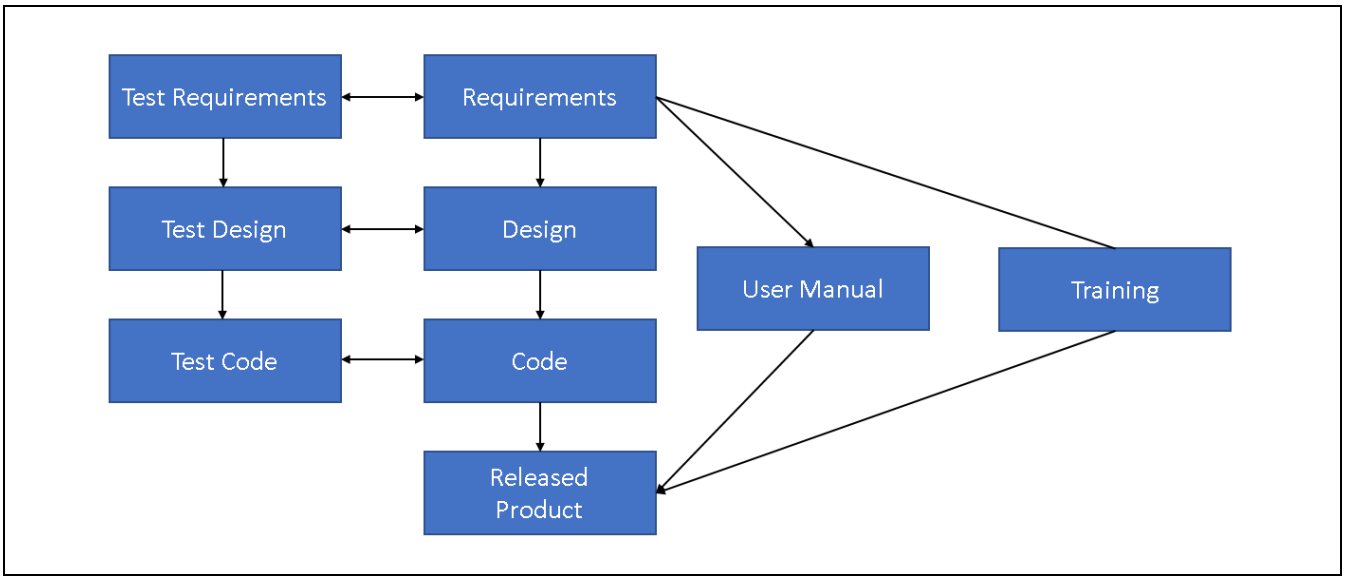


図2　テストが統合されたライフサイクル

3.8.4 テスト可能要求の特徴（Characteristics of Testable Requirements ）

要求がテスト可能となるには、要求が以下の特徴をすべて備えている必要があります。

1. **決定論的（Deterministic）**：初期システム状態にインプットがある場合、アウトプットがどのようになるかを正確に予測できる。
2. **あいまいでない（Unambiguous）**：プロジェクトメンバ全員が、要求を同じ意味に解釈する必要がある。そうでなければその要求はあいまいになる。
3. **正確である（Correct）**：原因と結果の関係が正確に記述されている。
4. **完全である（Complete）**：すべての要求が含まれていて、何も省かれていない。
5. **冗長でない（Non-redundant）**：データモデルが冗長でないデータセットを提供するために、要求も冗長でない関数とイベントセットを提供する。
6. **変更管理に適している（Lends itself to chage control）**：プロジェクトの他のすべての提出ドキュメント同様、要求も変更管理の下に置かれている。
7. **追跡可能（Traceable）**：要求は、目標・設計・テストケース、コードに対して追跡可能である。
8. **プロジェクトチームのすべてのメンバが読むことができる（Readable by all project team member）**：ユーザ、開発者、およびテスト技術者を含む、プロジェクトのすべての関係者はそれぞれ、要求を同じように理解する必要がある。
9. **一貫したスタイルで書かれている（Written in consistent style）**：要求は、理解し易いために、一貫したスタイルで書かれる必要がある。
10. **処理規則が一貫した基準を反映（Processing rules reffect consistent standards）**：処理規則は、理解しやすくするために一貫した形式で書かれている必要がある。

11. **明示的（Expricit）**：要求は決して暗示的であってはいけない。
12. **論理的な一貫性（Logically consistent）**：原因と結果の関係に論理的エラーが存在してはいけない。
13. **再利用に適している（Lends itself to reusability）**：良い要求は将来のプロジェクトで再利用可能。
14. **簡潔である（Terse）**：要求は、できるだけ少ない言葉で、簡潔に書かれている必要がある。
15. **重要度に関する注釈（Announced for criticality）**：すべての要求が重要なわけではありません。それぞれの要求は、その中に含まれる不良が生産に与える影響の程度を示す必要があります。これにより、各要求の優先度が決定でき、各要求の開発とテストに際して適切な注意が払われます。ほとんどのシステムでは不良をゼロにする必要はありません。テストの結果が「十分に良い」ものであればよいのです。
16. **実現可能である（Feasible）**：ソフトウェア設計で要求を実現できなければ、その要求は実現不可能です。

上記のリストの中で重要な特徴は「決定論的（Deterministic）」と「あいまいでない（Unambiguous）」です。定義上、テストとはあるシステムの予期される動作と観察される動作を比較することです。

3.8.5 ソフトウェア要件のトレーサビリティ（Tracebility of Software Requirement）

要求トレーサビリティは、ソフトウェア開発およびシステムエンジニアリング内の要求管理の下位の一分野です。要求トレーサビリティが関係するのは、要求生涯のドキュメント化と、様々な関連要求間での双方向トレーサビリティの提供です。これにより、ユーザは各要求の原点を見つけ、その要求に対して加えられる変更を1つ1つ追跡できます。この目的のため、要求に加えられた変更の1つ1つをドキュメント化する必要があるかもしれません。

トレーサビリティとは、「意味のある方法で、一意に特定できる存在（entity）どうしを、時間軸に沿って関係付ける能力」です。要求管理で重要なことは、時間的な進化ではなく、構造的な進化です。要求の発生元、要求がどのように満足されたか、要求がどのようにテストされたか、そして要求が変更された場合にどのような影響があるか、ということのトレースです。

RenesasのSynergyソフトウェアに対する要求は、様々なソースから来ます。市場セグメント、市場需要、顧客などです。要求トレーサビリティを使用することで、実装された機能を、要求導出の際にその機能を求めたソースまで追跡可能です。これを開発プロセスにおいても、要求の優先度を決めることに使用できます。要求がある特定ユーザに対してどの程度価値があるのかを知ることができるからです。また、機能が実現した後にも使用可能です。これは、ユーザがある機能を使っていないことが分かった場合、そもそもなぜその機能が要求されたかを知ることができるからです。

要求トレーサビリティが関係するのは、要求と他の開発中間生成物（artfacts）間の関係のドキュメント化です。その目的は以下の活動を円滑にすることです。

- 開発中の製品の全体的な品質
- 開発中の製品とその中間生成物（artfacts）の理解
- 変化を管理する能力

要求自体を追跡するだけでなく、要求と要求に関連するすべての中間生成物（artfacts）（モデル、解析結果、テストケース、テスト手続き、テスト結果、およびすべての種類のドキュメント）との関係も追跡する必要があります。要求に関係する人およびユーザグループも追跡可能である必要があります。

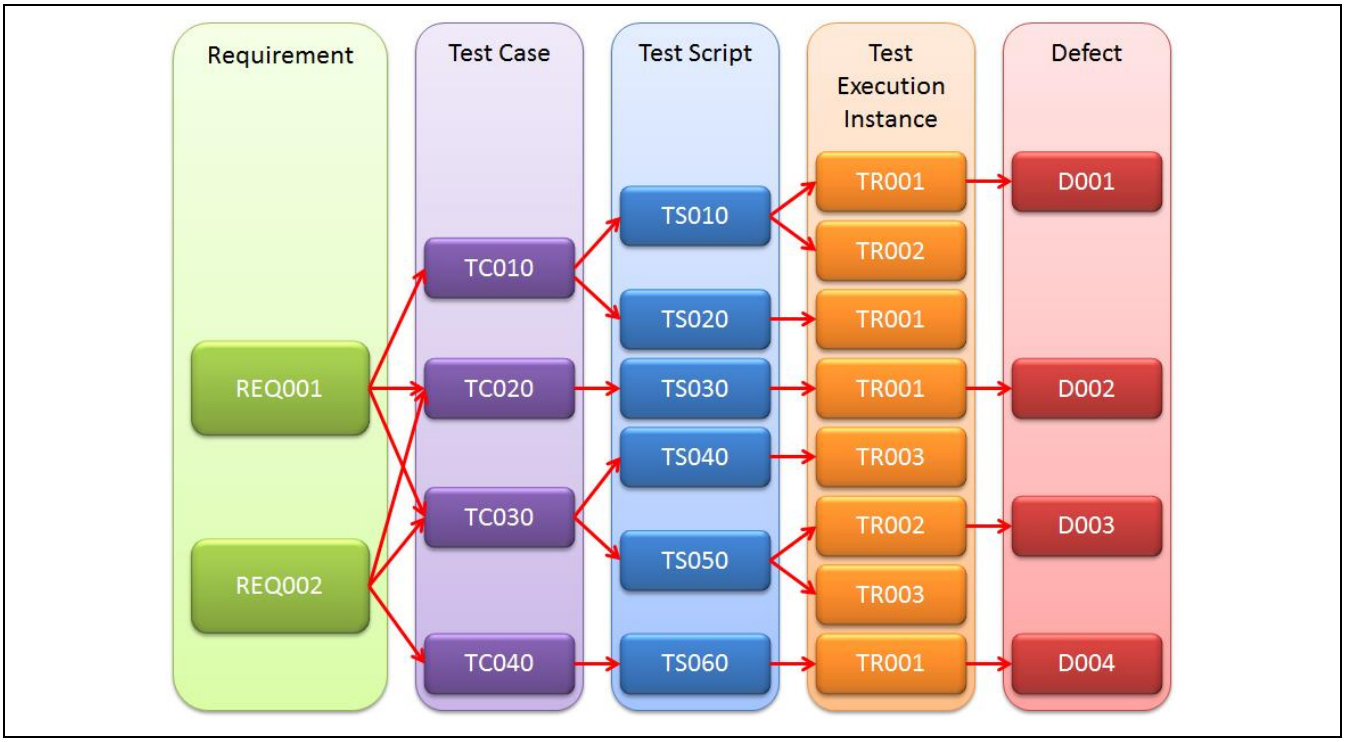


図3　ソフトウェア要件トレーサビリティ

3.8.6　ベースラインは矛盾しないこと（Baseline Must Be Congruent）

検証は、連続するベースライン（つまり連続するライフサイクルフェーズの製品）内および連続するベースライン間で連続する詳細レベル間の一貫性をチェックする必要があります。どの程度これが実行できるかは、該当するベースライン内の各レベルに含まれる情報に依存します。たとえば、設計仕様は、あいまいさのない、完全なSRDと比較することでのみ検証可能です。このように検証は、あるベースライン（ライフサイクルフェーズ）で意図されたものが次のベースラインで実際に実行されているかを確認します。言葉を変えれば、検証プロセスはライフサイクルフェーズ間のトレーサビリティを確立する必要があります。このトレーサビリティを実行する体系的な方法は、ソフトウェア構成管理プログラム（SCMP:software configuiration management program）にも含まれる必要があります。

3.8.7　ソフトウェア単体テスト（Software Unit Test）

開発環境で得られたソフトウェア中間生成物（artifacts）と単体テストデータは、別の継続的インテグレーション環境（Continuous Integration environment）に移行されます。すべての単体テストケースは、ソフトウェアコンポーネントの正確さと完全性を検証するために実行されます。

継続的インテグレーション環境は、LDRA社（Liverpool Data Research Associates）等の商用オフザシェル（COTS:Comercial-off-the-shelf）ツールと統合され、連続した動的および静的解析を行います。

3.8.8　機能テスト（Function Test）

機能テストはSDDをベースに、LDRA社のテストハーネス環境で開発されます。これらケースは、**境界（Boundary）ケース**、**範囲（Range）ケース**、および**頑強さ（Robuswtness）**ケースを含みます。この様なテストの目的は、名目（nominal）ケースおよび頑強さ（robustmess）ケースに対して、SDDに基づいて、ソフトウェアの動作を検証することです。

したがって、機能テストの目的は、すべての設計記述が独自にテストされた場合、100%の判定カバレッジを達成することです。境界（Boundary）ケース、範囲（Range）ケース、および頑強さ（Robustness）ケースの完全性の目的は、ピアレビューにより達成されます。

機能テストの終了基準は以下のとおりです。

1. すべてのSDDが少なくとも1つのSRDまで追跡される。
2. すべてのSDDが少なくとも1つのテストケースまで追跡される。
3. すべてのソフトウェアコンポーネント、仕様、および検証がバージョン管理され、ベースライン化される。
4. すべての検証ケースが実行され、合格/不合格基準を満たしている。
5. すべてのテストケースを実行して、100%の判定カバレッジを達成した。
6. すべてのテストケース、検証結果、およびカバレッジ報告がレビューされた。
7. すべての不具合が報告され、処置された。

3.8.9 ソフトウェア統合 (Software Integration)

ソフトウェア設定統合テスト（Software Configuration Integration Test）は、ハイレベルのユースケースを１セット提供し、ソフトウェアコンポーネントのすべてまたはそのほとんどを実行させます。ソフトウェアコンポーネントは組み合わされて、完全なソフトウェアシステムまたはシステムの大部分を形成します。

テストケースは、複数の異なるSynergy MCUハードウェアシステムで実行されます。このレベルのテストはほぼイベントドリブンであり、テストハーネス、ユーザインタフェース、または他のソフトウェアコンポーネントと相互に作用します。

インタフェースユースケースの種類は以下のとおりです。

1. MCU周辺回路に直接働きかけ、ドライバレベルで動作するユースケース
2. 周辺回路用HALレイヤーを使用して、フレームワークレイヤーで動作するユースケース
3. 抽象化された機能APIを使用して、フレームワークレイヤーで動作するユースケース
4. HALレイヤーにアクセスするRTOS対応ドライバのためのユースケース
5. アプリケーションコード向けハイレベルサービスを実装するために、ThreadX®サービスを使用するユースケース

ソフトウェア設定統合の終了基準は以下のとおりです。

1. すべてのユースケースがバージョン管理され、ベースライン化された。
2. すべてのユースケースがレビューされた。
3. すべてのケースが実行され、合格/不合格基準を満たしている。
4. すべての検証結果がレビューされた。
5. すべての不具合が報告され、処置された。

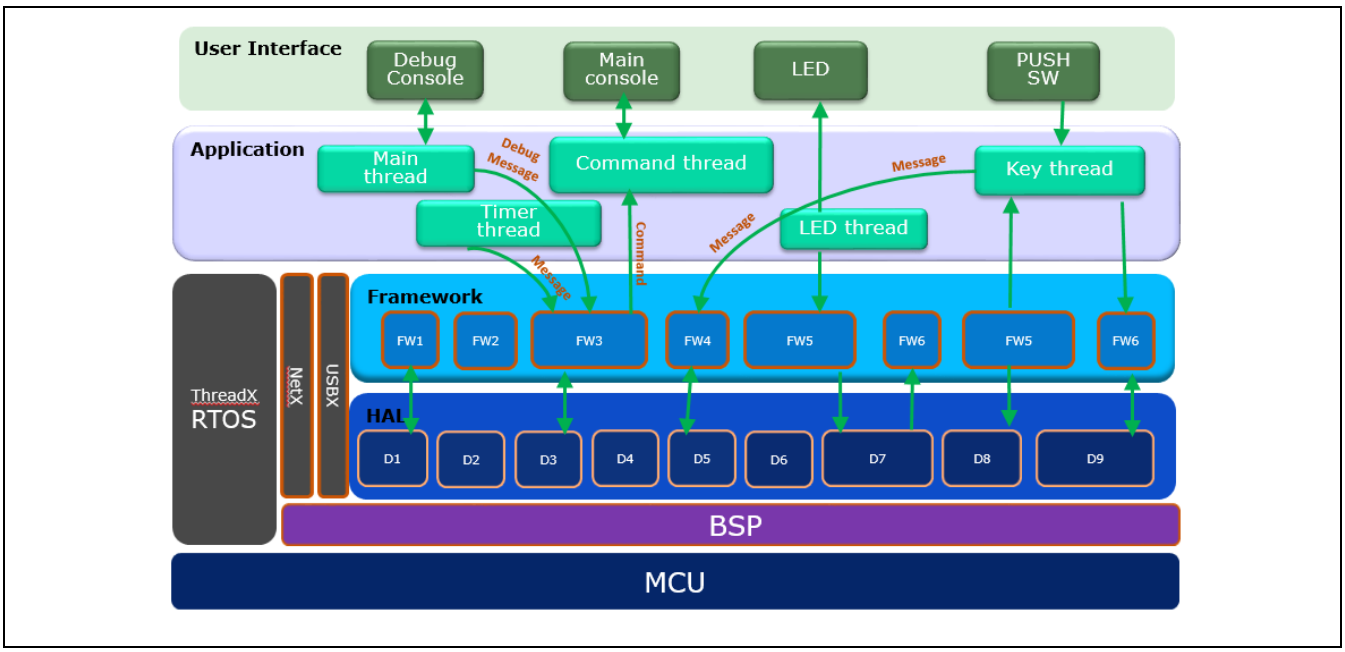


図4　ソフトウェア統合図

3.8.10　ソフトウェア性能テスト（Software Performance Tests）

市場要求を満たすかを確認するため、SSPコンポーネントは性能をテストされます。テストケースは複数の異なるSynergy MCUハードウェアシステムで実行され、ソフトウェアコンポーネントの拡張性（scalability）および安定性（stability）の特徴が報告されます。

ソフトウェア性能テストの終了基準は以下のとおりです。

1. すべてのテストが少なくとも1つのMRDまで追跡された。
2. すべてのテストがバージョン管理されている。
3. すべてのテストがレビューされた。
4. すべてのテストが実行され、合格/不合格基準を満たしている。
5. すべてのテスト報告がレビューされた。
6. すべての不具合が報告され、処置された。

3.8.11　回帰テスト（Regression Tests）（IoT-LED-00289）

継続的インテグレーション（CI：Continuous Integration）サーバが設定され、ある特定のベースラインを対象に、SSPがサポートするすべてのSynergy MCUハードウェアシステムに対して、テストが自動的にビルドされ、実行されます。CIサーバは、特定のベースラインに対する変更ごとにSPPの品質を測定および監視するために使用されます。CIサーバは、変更に互換性があり、品質が向上したことを確認します。

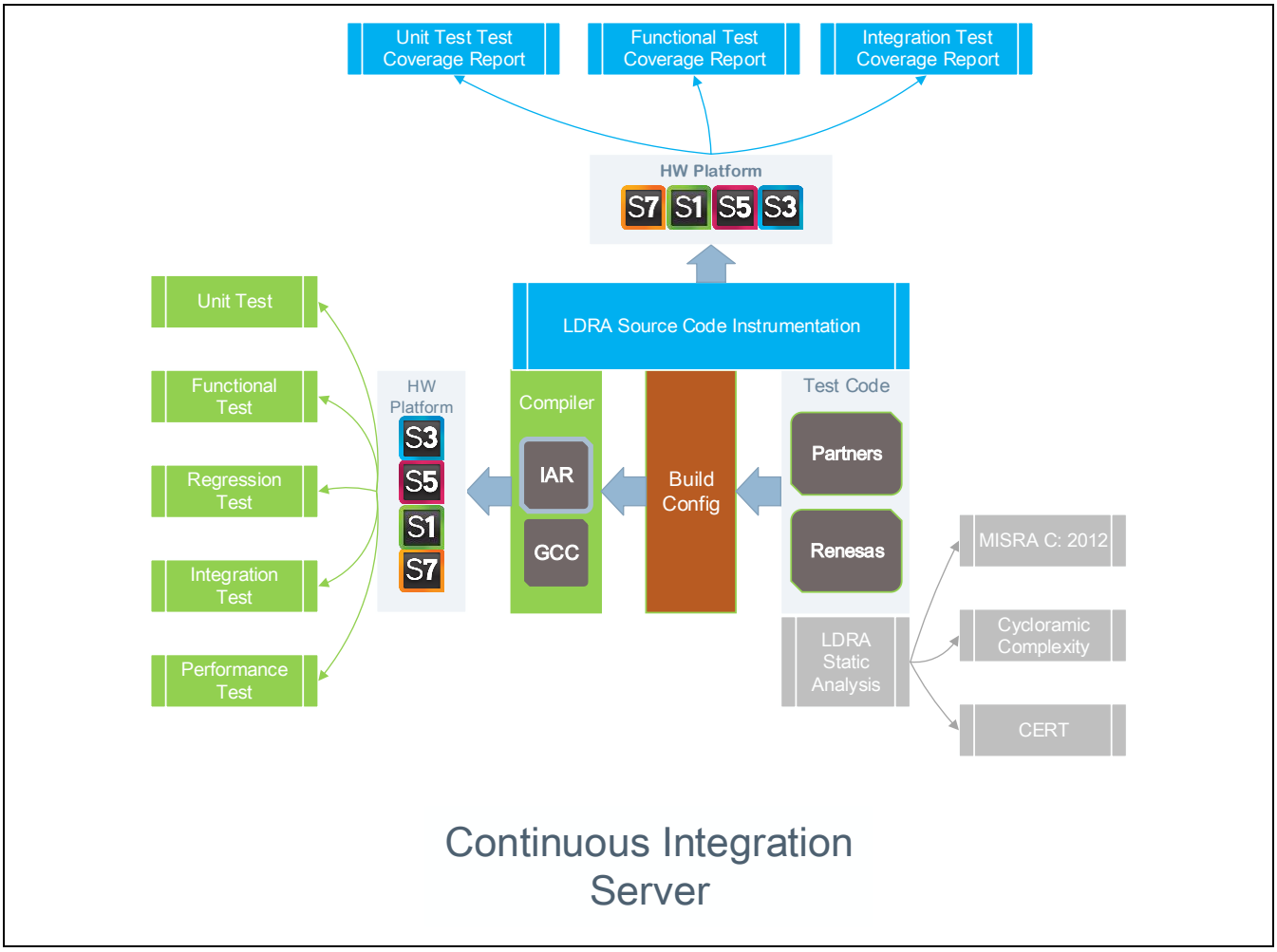


図5　連続統合サーバ

3.8.12　テストガイドライン（Test Guideline）

テストフェーズで使用する基準、慣例、および慣習は、単体テスト、結合テスト、回帰テスト、およびシステムテストについて、1セットのガイドラインに記述されており、そこではテスト繰り返し精度とテストカバレッジの基準が述べられています。それは、すべての要求、ユーザ手続き、およびプログラミングステートメントをテストすべきと記述する要求を含めることで行われます。

ガイドラインは、サポートソフトウェアが使用されるかどうかも示します。テストガイドラインには、プログラムテストの実行を管理する具体的な基準が含まれています。これにより、すべてのプログラマが一律にプログラムをテストすることが可能になります。

3.8.13　動的解析（Dynamic Analysis）

動的解析は、テストデータ選択によってテスト実行中のプログラムの意味付け（semantic）を探ります。動的解析は、管理およびデータフローモデルを使用し、プログラムが実行される際に実際の制御およびデータフローとの比較を行います。したがって動的解析時には、ソースコードの構造を探るテストデータを選択しなくていけません。LDRA社ツールは、動的カバレッジ解析モジュール（Dyanamic Coverage Analysis module）が含まれます。このモジュールの使用は、開発サイクルと保守サイクルの両方において、ソフトウェアの堅牢性と信頼性に有益な影響を与えます。

3.8.14　静的解析（Static Analysis）

静的解析は、単一のファイルまたは完全なシステムに対して、ソースコードの語彙（lexical）および構文（syntactic）解析を行うことで、LDRA社 Testbed作業を開始します。Renesas Synergyソフトウェアコーデ

ィング基準に準拠しているかどうかは、LDRA Testbedによって自動的にチェックされます。主要静的解析は、ソースファイルをチェックして、ソースコード内にコーディング基準違反があるか検索します。

LDRA社 Testbedは選択した基準セットでの違反を、テキストで報告するとともに、画面表示に対する注釈としても報告します。

3.8.15 レビュー (Review)

ドキュメントまたはコードの初版はフルレビューの対象となり、変更は変更部分レビューの対象となります。レビューの目的は、システムソリューションに対する概念的手法および技術的手法を評価し、プロジェクトのために事前に定義された品質要因が満たされていることを確認することです。このレビューは、検証手法での問題の特定も行います。

3.8.16 監査 (Audit)

ドキュメントと関連する検証方法を調べることで、プロセスとそのドキュメントがSQAプランならびに組織の方針・手続きに適合しているかを検証します。

3.9 ソフトウェアリリースフェーズ (Software Release Phase)

結合ステージの主なアウトプットは、製品アプリケーション、完了した承認テスト一式、およびソフトウェアのリリースノートです。最終的に、DPAは最後の実労働データをプロジェクトスケジュールにインプットされ、プロジェクトを恒久的なプロジェクト記録としてロックします。このとき、DPAがプロジェクトを「ロックする」とは、すべてのソフトウェアアイテム、実装マップ、ソースコード、および将来の参照のためのドキュメントをアーカイブ保存することです。

3.10 範囲の制限 (Scope Restriction)

プロジェクトの範囲は、承認されたMRD内の要求リストによって決定されます。これらの市場要求は引き続き、システムおよびソフトウェア要求に、次に設計要素に、そしてソフトウェア中間生成物 (artifacts) に変換されていきます。要求、要素、および中間生成物 (artifacts) から成るこの階層構造は、要件トレーサビリティマトリックス (RTM) にドキュメント化されます。RTMはプロジェクトを、本来定義された範囲内に収めるための管理要素の役割を果たします。

プロジェクト参加者は、要求、要素、および中間生成物 (artifacts) が、MRDにリスト化された市場要求まで直接追跡可能になるように取り組む必要があります。これにより、ソフトウェアプロジェクト失敗の第1の要因である、要求の変更・追加 (scope creep) が多く発生することを防ぎます。

プロジェクト範囲に対する変更は、変更を実施する前に、変更管理審査委員会 (CCRB) による承認が必要です。

4. ドキュメント品質 (Documentation Quality)

開発プロジェクトの全体的な品質は、ドキュメントの品質に依存します。効果的なSQAプランは、すべてのプロジェクトおよびソフトウェアのドキュメントに対して、統一された要求を指定します。これらの基準は、各ドキュメントの範囲と形式を定義します。この基準はまた、技術ライティングスタイルの問題にも取り組み、ドキュメントの明瞭さと一貫性を向上させます。トレーサビリティで必要となるため、10進法で段落に番号を付けます。

4.1 ドキュメント基準 (Documentation Standards)

ドキュメントは適切な基準に従って書式設定されます。基準により、作成者はドキュメントに何を含めるべきか、そしてドキュメントの取るべき書式を正確に知ることができます。

4.2 ドキュメントレビュー (Documentation Review)

ドキュメント完成後は、すべてのドキュメントはその正確さがピアレビューされ、また完全さを調べるために、ドキュメント作成に関わらなかった独立したレビュー者によってレビューが行われます。

4.3 ドキュメント維持 (Documentation Maintenance)

SSP (Synergy Software Package) に関連するすべてのドキュメントはオンラインで維持されます。オンラインでドキュメントを維持することで、開発者は必要なときに最新版を入手できます。

4.4 ドキュメント管理 (Documentation Control)

SSP (Synergy Software Package) ドキュメントは、コンピュータプログラム自体同様、ソフトウェア製品とみなされ、プログラムと同様に構成管理と構成制御の対象となります。すべてのドキュメント変更は、適切なドキュメント管理者または責任者による承認プロセスを経ます。

5. 基準、慣例、および慣習（Standareds,Practices,and Conventions）

適切な基準・慣例・慣習の決定、実施、および施行は、プログラムの認定に欠かせません。ソフトウェア開発ライフサイクル（SDLC）基準には手続きと規則が含まれます。これらの手続きと規則は、ある特定のタスクを完成させるための規律がとれた統一アプローチを規定するために採用され、施行されます。

Renesas SynergyソフトウェアSDLCはIEC12207を基に作られました。IEC12207はソフトウェアライフサイクル開発のための業界共通規格であり、IEC61508やIEC62304等の他の高度に規制された規格に準拠しています。

Renesas Synergyソフトウェアプロセスは、ソフトウェアの開発または使用のための承諾された方法・技術です。これらの方法・技術は、Renesas製品の統一性を確実にするために確立されました。最良事例（best practice）が、データ配置や情報表示をするのに、統一されたパターンや形式を使った方法、技術、慣習を規定することで、それに一貫性をもたらし、またその理解を助けるのと同じです。

これらの基準は、技術的な役割と管理的な役割の両方を果たします。また、プログラムの読みやすさ、ソフトウェア検証および妥当性確認、インタフェース定義、そしてソフトウェア開発の管理レビューを促進します。Renesas Synergyソフトウェアプロセスは、これらの基準の目的を満たすために、必要な手続きや活動をはっきりと規定します。

SQAの主な役割は、ソフトウェア製品とソフトウェア開発プロセスを監視し、それらが採用された基準に準拠することを確認することです。採用された基準は、ソフトウェアライフサイクルにおいて、あるイベントを別のイベントへとつなぐ線であり、ある特定の要求が最終製品でどのように実施されているかを示します。

プロジェクト進行中に基準または手続きに見直しがおこなわれる場合、見直された基準がプロジェクトに与える影響が評価され、変更管理審査委員会（CCRB）に提出されます。そしてCCRBにおいて、前の基準に準拠し続けるのか、あるいは新しい基準に準拠するのかが決定されます。ただし、プロジェクトの過程において、どの手続きが実行中であるかは常に記録で明白に示されます。

6. レビュー、監査、および管理手段（Reviews,Audits,and Controls）

ソフトウェア開発、運用、および保守作業は、定期的にレビューおよび監査され、SQA要求に適合しているかが決定されます。技術レビューと監査は定期的に実施され、判断が出されます。これにより技術的な取り組みの状態と品質が評価され、要求される技術ドキュメントの作成と採用された基準への準拠を確実にします。

ソフトウェア開発プランとスケジュールに対する、具体的な技術レビューと監査はSQAプランで決められます。またレビューと監査で使用する手続きがガイドラインで記述されます。参加者と個々の責任も同様に明示されます。最低限、下記のレビューと監査が行われます。

1. 市場要求レビュー（Market requirements review）
2. ソフトウェア要求レビュー（Software requirements review）
3. 予備デザインレビュー（Preliminary design rfeview）
4. 重要デザインレビュー（Critical design review）
5. ソフトウェア検証レビュー（Software verification review）
6. 機能設定監査（Functional configuration audit）
7. 物理設定監査（Physicalconfiguration audit）
8. 中間監査（In-process audit）
9. 管理レビュー（Managerial review）

6.1.1 技術レビュー（Technical Reviews）

技術レビューはソフトウェア品質の確立する以上に、多くの目的があります。技術レビューにより、複数の人間がその経験を製品開発者と共有することができます。ソフトウェアレビューは、個人と開発プロジェクトに関わるチームの技術的能力を進歩させる効果があります。グループのメンバは、しだいに同僚のことを知り、理解するようになります。ある状況でどのように考えるのか、よくミスが発生するのはどこか、などです。このように互いを理解することで、技術チームはより良いものとなり、同じような問題が再発するのを防ぐことができます。人々をチームに振り分けることで、プロジェクトがスムーズに進行します。チームを編成し、仕事を割り振る過程で、個人個人の能力の違いを補うことができます。チームでは、個人では見過ごしてしまう不具合を見つけることがしばしばあります。

6.1.2 レビューチームのメンバ（Review Team Members）

レビューは、すべての活動を徹底的にレビューできる十分な技術的専門知識を持つ人物によって行われます。独立チェックはSQA組織ではなく、エンジニアリンググループまたは技術グループによって行われます。SQA組織が独立チェックを行わないのは、SQA組織は通常監査を行うことが役割だからです。レビュー参加者は、プログラムロジックを開発した人々とは別であり、プログラムタスクに関連する分野で十分な技術的能力がある人達です。

6.1.3 レビュー手続き（Review Procedures）

レビューと監査が明確にされた後、スケジューリングされ、適切な順番に並べられます。レビューと監査の手続きは、参加者とその具体的な責任、そして収集してレビューする情報の種類を決定します。この手続きは、各レビューの報告書を作成することも規定し、報告書を作成する人物も決定します。さらに、報告書の書式、報告書を受け取る者、関連する管理責任、ならびにフォローアップ活動も記述します。これにより、レビューと監査時に出された勧告が適切に実施されることを確実にします。また、レビューとフォローアップ活動の間の時間間隔と、フォローアップ活動を行う責任者も規定されます。

技術レビューの過程で、チェックリストは効果的に使用されます。レビューの参加者は、正式なレビュー会議の前に、入手可能な全てのドキュメントを、チェックリストに照らして検査します。

6.2 監査 (Audits)

本項では、SQAプログラムの監査とSQAの機能を記述します。

6.2.1 機能設定監査 (Function Configuration Audit)

機能設定監査はソフトウェア納入の前に行われ、ソフトウェア要求仕様で明確に記述された要求がすべて満たされていることを検証します。機能監査ではコードを、現行のソフトウェア要求ドキュメントで示される要求と比較します。その目的は、コードがドキュメント化されたすべての要求に対応しているかどうかを見極めることです。結果のドキュメントに含まれるのは、一致しなかった点と、その解決のためのプランとスケジュールです。不一致が解決されると、ソフトウェアはユーザに納品可能になります。

6.2.2 物理設定監査 (Physical Configuration Audit)

物理設定監査の主な目的は、コンピュータプログラム開発作業によるすべての技術的成果物（product）が完全なものであるかどうか、そして承認されたプロセスに正式に則っていたかどうかを見極めることです。物理監査の際に監査される資料に含まれるのは、コンピュータプログラムに関連する技術的成果物（product）です。たとえば、最終SRD、ソフトウェア設計記述、そして各リリースのために正式に用意された、その他のすべての文書です。

6.2.3 中間監査 (In-Process Audits)

中間監査の目的は、開発で進化する製品の一貫性、または保守フェーズ中に変更される製品の一貫性を検証することです。すべての中間監査の結果はドキュメント化され、監査で発見されたすべての不一致は、その解決のためのプラン、スケジュールとともに明記されます。

6.2.4 SQA監査 (SQA Audits)

SQA監査は、SQAドキュメント内で規定される手続き、基準、および慣習に対する適合性と、効果を評価します。ライフサイクルを通じて、内部手続き、SQAプラン、設定管理、物理的および機能的観点の両方から契約に基づいて要求される成果物が監査されます。SQA監査はドキュメントの目視検査を含み、ドキュメントが承認された基準および要求に合致しているかどうかを見極めます。SQA監査は、問題解決策または設計に対する概念的な手法をレビューすることを意図していません。むしろ監査者は、各ドキュメントの書式を調べて、規定された概要に適合しているかどうか、そして抜け、明らかな矛盾、後の作業で混乱の元になりそうな項目があるかを調べます。監査者は、すべての必要なドキュメントが存在すること、そして各ドキュメントの品質が容認可能であることを検証します。正式なSQA監査報告書が作成され、監査権を持つプロジェクト管理者に、情報と対応のために提出されます。このような監査が、設計、コーディング、ドキュメント作成等と同時に行われ、大規模な作業のやり直しや、コスト超過につながりかねない見過ごしの可能性、うっかりした思い違いを減らします。

6.2.5 是正措置 (Corrective Action)

レビューおよび監査プロセスを完了するためには、不具合修正のプランとスケジュールが必要です。是正措置は変更管理審査委員会（CCRB）の承認を受けます。是正措置が不要または遅らせてもよいと判断された場合、問題はリリースノートに記載され、問題報告システムに既知の問題として記録されます。

7. ソフトウェア保守フェーズ（Software Maintenance Phase）

保守フェーズの目的は、ソフトウェアのリリース後に発見された問題を収集し、対応することです。これら問題はソフトウェアの製品への複製および発送の際に内部で発見される場合もあれば、顧客がソフトウェアをインストールした後に顧客から報告される場合もあります。以下の項では、それぞれの場合を簡潔に記述します。

7.1 内部問題（Internal problems）

問題が内部で発見された場合、追跡システムでは不具合として報告されます。このような問題は、**IOT-PRP-00247問題報告手続き**に従って報告および追跡されます。問題は以下の手続きに従って、解決まで追跡されます。

7.1.1 不良およびバグのライフサイクル（Defect and Bug Life Cycle）

以下に、問題追跡システムで報告された不具合またはバグのワークフローを説明します。この問題追跡システムにおける問題管理の手続きの詳細については、IOT-PRP-00247を参照してください。

1. 問題が報告されます。
2. 問題の状態は「Open（公開）」に変更され、作業が実行できるようになります。
3. 作業が開始されると、状態は「In Progress（進行中）」に変更されます。
4. さらに情報が必要な場合、または問題に対する作業の開始が遅れている場合、状態は「Waiting（待機中）」に変更されます。作業が再開すると、状態は「In Progress（進行中）」に戻ります。
5. 問題に対する解決策が決定すると、状態が「Review（レビュー）」に変更されます。
6. 問題の解決策が許容可能であれば、状態は「Resolved（解決済）」に変更されます。さらに検証が必要な場合、問題は検証のためにSQAに照会されます。これ以上検証が不要な場合、状態は「Closed（終結）」に変更されます。解決策が許容不可の場合、状態は「Reopen（再公開）」に変更され、作業が再開されて、許容可能な解決策を探します。
7. 問題が解決されると、SQAによる最終的な検証を受けます。SQAが解決策を検証すると、状態は「Closed（終結）」に変更されます。検証の際に問題点が見つかった場合、状態は「Reopen（再公開）」に変更され、さらに作業が必要となります。

7.1.2 不具合、バグの報告（Reporting-Defect, Bug）

不具合/バグは、それが再現できるように明確に報告される必要があります。それにより、調査と解決が可能になります。報告者は不具合/バグの区分の詳細情報を提供する必要があります。報告内容には以下の詳細が含まれます。

1. 概要
2. 不具合/バグが見つかった中間生成物（artifacts）のIDおよびバージョン
3. 不具合/バグを再現するための方法/手順
4. 結果が想定から逸脱した理由の説明テスト実行のために使用されたテストケースのID
5. より明確にするための、画像またはエラー情報の添付

7.1.3 不良、バグの種類（Defect, Bug-Types）

不具合/バグは以下の種類に分類されます。

1. ドキュメント
2. ビルド/パッケージ
3. レビューとピアレビュー
4. コーディング基準
5. 単体テスト（UT）
6. 結合テスト（IT）

7. 性能テスト
8. テスト/開発環境
9. 構造カバレッジとコード複雑度
10. 不要な機能的動作

7.1.4 不具合、バグの調査（Investigation-Defect,Bug）

問題追跡システムで報告された不具合/バグは、調査され、関係者に渡されます。不具合/バグ調査は以下の手順で行います。

1. 不具合/バグの確認
2. 根本原因解析（RCA）
3. 影響解析
4. 解決策の提案

問題の調査の際、他の既知または潜在的問題との関連性および類似性を考慮する必要があります。類似の問題は明示的なリンクで追跡するか、主問題のサブ問題と位置付けて追跡します。

7.1.5 不具合、バグの解決および終結（Resolve and Close-Defect,Bug）

不具合は変更管理審査委員会（CCRB）の承認を必要とせず、任された人物によって修正され、報告者（独立レビュー者）によって検証/終結されます。

バグを修正するには変更管理審査委員会（CCRB）の承認が必要です。バグが修正されると、報告者によって検証され、SQAによって終結されます。

1. バグ記録に対する変更管理審査委員会（CCRB）の承認
2. 提案された解決策の実施
3. 中間生成物（artifact）に対する変更は固有の不良/バグIDまで追跡
4. 報告者による独立レビュー
5. 不具合は、レビュー後、報告者によって終結可能
6. バグは、レビュー後、SQAによって終結可能

従うべき手続きの詳細な記述については、**IOT-PRP-00247問題報告手続き**を参照してください。

7.2 顧客から報告された問題（Customer reported problems）

顧客が発見した問題は、保証請求またはサポート要請を伴う場合があります。

7.2.1 保証請求（Warranty Claims）

顧客が保証請求を提出した場合、まず該当地域のRenesasサポートチームによって調査されます。顧客は保証請求を提出する前に、Synergy開発キット（DK）を使用して問題を再現する必要があります。これは地域のサポートチームと共同で行われるかもしれません。保証請求ができるのは、SSPデータシートの性能セクションに規定されているSSP動作に対してのみであり、しかも同じく性能セクションに規定されているMCUハードウェアシステム上で動作中のSSP動作に限ります。RenesasがSSPデータシートの性能セクションに規定されている動作と照らして、問題がSSPにおける不具合であると確認した場合、顧客は24時間（週末を除く）以内にRenesasから、この確認の知らせを受けます。Renesasは不具合を解決するための作業を続け、7営業日以内にRenesasは顧客に解決策を提供するか、不具合の解決策に関する最新情報と解決策を提供するまでの追加時間の見積もりを通知します。

7.2.2 サポート要請（Support request）

SSPソフトウェア問題に関して顧客がサポートを要請した場合、その問題はRenesasの問題報告および追跡システムにインプットされ、チケットが作成されます。

サポートチケットの目標は以下のとおりです。

1. すべてのチケットは、24時間以内に要請者（requester）に対する最初の応答を受信すること。
2. すべてのチケットは72時間以内に「Solved（解決済）」または「Pending（保留中）」になること。
3. 1週間以上「Open（公開）」のチケットは、マネージャーに上げられること。

チケットはチケット内のキーワードに基づき、システムによって自動的にエージェントに割り当てられます。この割り当てが間違っている場合は、チケットは正しいエージェントに再割り当てされます。

チケットは以下の状態をとります。

- エージェントがチケットを受信すると、チケットの状態は「NEW（新規）」となっています。
- エージェントがチケットを開いて読んだ場合、またはサポート要請者とやり取りを開始した場合、状態は「OPEN（公開）」に変わります。チケットの状態は手動で変更しないかぎり、「OPEN（公開）」のままです。
- エージェントが情報を待っている場合は、エージェントが状態を「PENDING（保留中）」に変更します。
- 問題が解決されると、状態は「SOLVED（解決済）」になります。
  - チケットが「SOLVED（解決済）」になってから24時間以内に顧客が応答すると、状態は「OPEN（公開）」に変更され、作業が継続します。
  - 24時間以内に顧客が応答しない場合、状態は「CLOSED（終結）」に変更されます。
  - 顧客が24時間後以降に応答した場合、システムは新しいチケットを作成します。

チケットを解決する際、エージェントはナレッジデータベースを検索し、問題の答えがないか調べます。答えがない場合、問題が解決した後で、エージェントはナレッジデータベースに追加すべき情報があるかどうかを決めます。

一部のチケットは、さらなるフォローアップのために、内部問題報告システムに追加されます。

8. ソフトウェア指標（Software Metrics）

ソフトウェア指標は、ソフトウェア品質測定を数値で表すための手段です。その意図は、ソフトウェアエンジニアリング手法と改善されたソフトウェア品質の相関関係を示すことです。その長期的な目標は、様々なSQA手続きと構造化開発手法が、品質全体にどう影響するかを知ることです。指標データを収集することで、品質改善が測定できます。このような品質指標が重要なのは、測定しないものを管理して改善するのが困難だからです。

8.1 ソフトウェアの複雑度（Complexity of Software）

ソフトウェア複雑度は、コード完成による機能的複雑さの副次的結果です。複数のシステムインタフェースと複雑な要求のため、ソフトウェアシステムの複雑度は時に制御不可能となり、そのためアプリケーションと製品ポートフォリオの保守が過剰に高価になり、その拡張にはリスクが伴うようになります。そのままにしていると、ソフトウェア複雑度は納入プロジェクト内で度を超えて高まり、残るのは肥大化した、扱いにくいアプリケーションになります。

ソフトウェアエンジニアリング分野は、ソフトウェア複雑度の共通の測定指標を幾つか確立しています。最も一般的な測定指標がMcCabe本質的複雑度指標です。これは循環的複雑度（cyclomatic complexity）と呼ばれることもあります。これは1つのコード内での、ルーチンの深さと量の測定指標です。

信頼に値するソフトウェア複雑度指標の使用無しでは、リスクとコストが発生する構造上のホットスポットを突き止めるのは困難で、時間がかかります。さらに重要なことは、連続的なソフトウェア複雑度解析によって、プロジェクトチームと技術管理部門が問題の先を行き、過剰な複雑さの定着を防げます。

複雑度を測定するとき、結合、まとまり、フレームワークの使用、およびアルゴリズムの複雑さを全体的に見ることが重要です。また、アプリケーション製品群の技術レイヤーの全てにおいて、一貫性のある、正確かつ繰り返し可能な一連の複雑度指標を使用することも重要です。これは、ビジネスニーズに適合するために変更を実施する際の継続的な評価に基準を提供するためです。堅牢なソフトウェア複雑度測定プログラムは、組織に以下を実現する機会をもたらします。

- コード品質の向上
- 保守コストの削減
- 生産性の向上
- 堅牢さの向上
- アーキテクチャ基準への準拠

定義されたソフトウェア複雑度アルゴリズムに基づいた自動解析は、アプリケーションのサイズや解析の頻度にかかわらず、包括的な評価を可能にします。自動化は客観的で、繰り返し可能、一貫性があり、コスト効果が高くなります。ソフトウェア複雑度測定体制は、より迅速にソフトウェアを納入したいと思う組織にとって重要です。

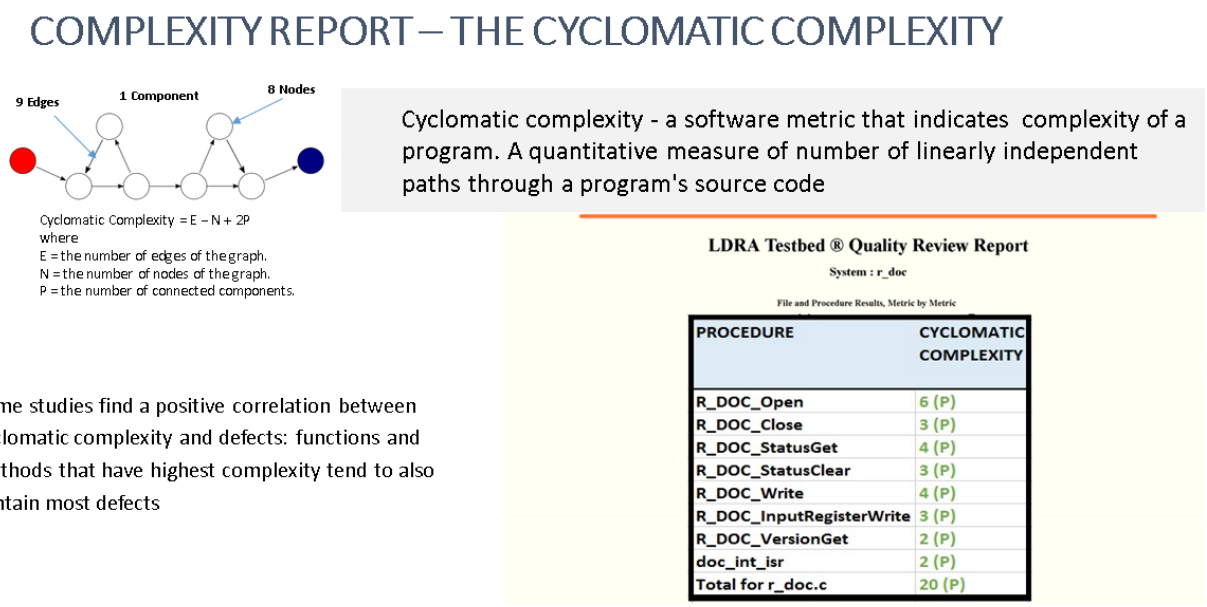


図6　循環的複雑度（Cyclomatic Complexity）

8.1.1　循環的複雑度限度（Cyclomatic Complexity）

コードの複雑度が高いとデバッグ作業が増え、コードの信頼性が低下します（セキュリティ欠陥が増えるため）。

複雑度	信頼性に関する危険性
1～10	単純な機能、低い危険性
11～20	複雑度が増す、中程度の危険性
21～50	複雑、高い危険性
51以上	テスト不可能、非常に高い危険性

コードの複雑度が高いと、既存の欠陥を固定するとともに、新しい欠陥が挿入される危険性が高くなります。

複雑度	変更を加える場合にバグが挿入される危険性
1～10	5%
20～30	20%
50より高い	40%
100近く	60%

注：McCabe、Thomas Jr. 「Software Quality Metrics to Identify Risk（危険性を特定するソフトウェア品質メトリック）」。国土安全保障省ソフトウェア保証ワーキンググループへのプレゼンテーション、2008年（http://www.mccabe.com/ppt/SoftwareQualityMetricsToIdentifyRisk.ppt#36）
および、Laird、Linda、Brennan, M. Carol「Software Measurement and Estimation: A Practical Approach（ソフトウェア測定および見積もり：実際的アプローチ）」。カリフォルニア州ロスアラミトス、IEEEコンピュータソサエティ、2006年

8.1.2　モジュールソフトウェア複雑度指数（Icmx：Module Software Complexity Index）

モジュールの各機能の循環的複雑度は、静的解析ツールを使用して計算します。モジュールの計算された循環的複雑度の最大数は、以下のように割り振られます。

複雑度の最大数	指数
0より大きく、10以下	5
10より大きく、15以下	4
15より大きく、20以下	3
20より大きく、25以下	2
25より大きく、30以下	1
30より大きい	0

8.2 クリーンビルド (Clean Build)

8.2.1 警告指数 (Iwar : Module Code Coverage Index)

異なるツールチェーンを使用した様々なビルド設定から生成された警告の最大数は、以下のように割り振られます。

警告の最大数	指数
ゼロ	5
ゼロより大きく、5以下	4
5より大きく、10以下	3
10より大きく、15以下	2
15より大きく、20以下	1
20より大きい	0

8.3 構造判定カバレッジ (Structual Decision Coverage)

8.3.1 モジュールコードカバレッジ指数 (Icov : Module Code Coverage Index)

モジュールコードカバレッジ指数は以下のように計算されます。

総合判定カバレッジ (%)	指数
100	5
95以上100未満	4
80以上95未満	3
60以上80未満	2
50以上60未満	1
50未満	0

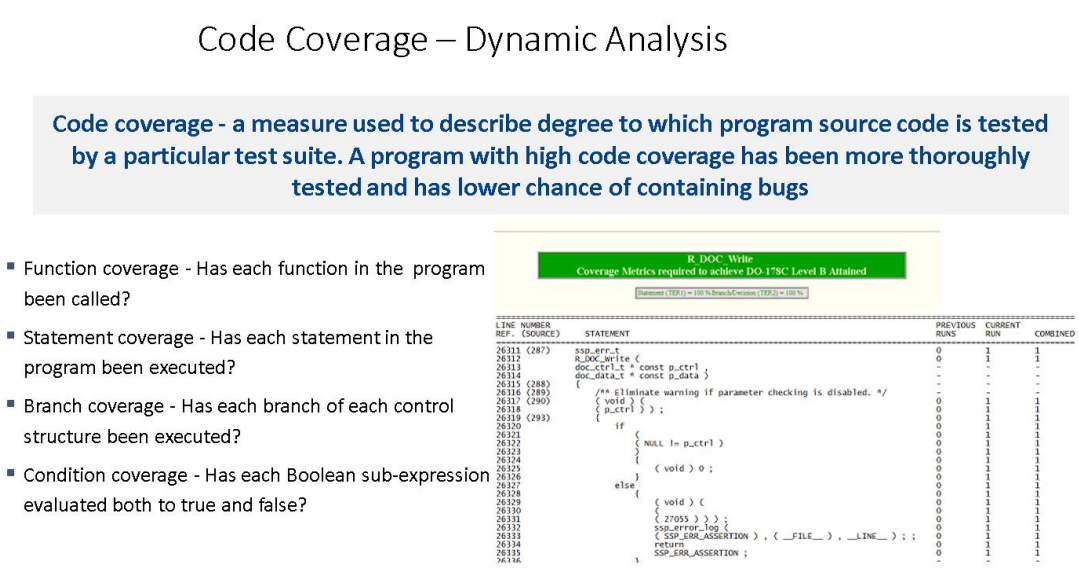


図7　コードカバレッジ - 動的解析（Code Coverage - Dynamic Analysis）

8.4　コーディング基準（Coding Standards）

実装フェーズとコーディングフェーズ中に使用する慣例と慣習は、コーディング基準に記述されています。コーディング基準は、テスト可能な方法で、品質属性が規定されるようにします。構造化されたコードまたは事前コンパイルの構造化の使用、ローカル/グローバルデータアクセス、およびパラメータの受け渡しに基準を実施することで、コーディングエラーの数が減少します。コーディング基準を使用することで、コード保守も改善されます。特にコードの体裁、配置ならびにコメントに関連するコーディング基準が改善されます。基準に含まれるのは、モジュールサイズ、命名および番号付け、ヘッダーコメント、インラインコメント、ローカル/グローバルデータアクセス、パラメータ受け渡し、そしてコード書式です。プログラミング基準への準拠を検証するための自動または手動の手段は、ソフトウェアツールを使用して実施可能です。本ハンドブック執筆者の経験では、このような手段を利用するコスト効果が高くなります。

均一性をもたらすため、ソフトウェア各バージョンとコンポーネント向けのネーミングとラベリング方式が規定されています。プログラム名はすべてのソースコードに含まれ、またバージョンはすべての要素（element）から発生します。プログラムの各バージョンは、一意（unique）なバージョン番号を与えられます。バージョン番号は、そのプログラムがテストされた日付ともに、プログラムから得られた検証結果に結び付けられます。ネーミングとラベリング方式は各プロジェクトを一意（unique）にします。Renesas全体でその形式（format）は統一されています。

ソフトウェア品質を高めるため、Renesas Synergyソフトウェアコーディング基準は、各ソフトウェアモジュールに対して、適切なレイアウト方式を用いることを命じています。プログラミング方式の詳細仕様が規定されています。プログラミング方式とは、プログラムステートメントの字下げと間隔、コメントの使用、およびプログラミング言語の特定の機能の使用またはその制限などです。特に保守において、レイアウト方式が要求されます。その結果、Renesas Synergyソフトウェアは常に同じ形式であり、特定の形式に慣れ親しんでいるというだけで、保守プログラマは大量の情報を得ることができます。

コーディング技術はプログラミング言語によって差がでる傾向にありますが、すべてこの基本的な構造的に構成された概念は、どんなプログラミング言語において標準的なやり方で実施可能です。

標準的な構成概念は、厳密に従うならば、詳細設計の際に作成された、いわゆる疑似コードやその他の設計表現を直接的にプログラミング言語に容易に変換できるよう助けてくれます。

Renesas Synergyソフトウェアコーディング技術は、アサーション技術（assertion technology）を利用してインライン検証を行います。アサーションはコメントの形でコード内に埋め込まれます。そして後で起動さ

れ、コードの任意の場所で処理変数の状態を知ることができます。アサーションは、物理的に実行可能な変数の限度（範囲などの）をチェックできます。それにより、コード実行中でもある程度の検証が行えます。プリプロセッサとポストプロセッサを使用して、ほとんどの実装言語にアサーションを埋め込むことができます。

インラインコメントのためのRenesas Synergyソフトウェア基準は、コメントに含まれる詳細記述の量を均一にします。コメントは、単にロジックフローを見れば簡単に知ることのできる情報を反映しているわけではありません。たとえば、コンピュータのコードで明白なのに、「プラスで分岐」などのコメントは必要ありません。コメントはプログラミング論理に関する情報を付け加えるものであり、プログラムリストに詳細設計を埋め込む方法として使うことができます。

8.4.1 モジュールコーディング基準指数（Icstd：Module Coding Index）

SSPコーディング基準への準拠は、LDRA社 Testbedによって自動的にチェックされます。違反の総数は以下のようにスケーリングされます。

コーディング基準違反総数	指数
ゼロ	5
0より多く、20以下	4
20より多く、40以下	3
40より多く、60以下	2
60より多く、80以下	1
80より多い	0

8.5 ソフトウェア検証（Software Verification）

8.5.1 モジュールソフトウェア検証指数（Ivnv:Module Software Verification Index）

モジュールソフトウェア検証指数は以下のように計算されます。

項目	説明	はい	いいえ
E	すべてのテストが実行済み	1	0
F	失敗なし	1	0
I	無視なし	1	0
T	追跡可能	1	0
R	完全な要件カバレッジ	1	0

注： Ivnv = E + F + I + T + R

8.6 ソフトウェア後方互換性（Software Backward Compatibility）

8.6.1 モジュール後方互換性指数（Ibck：Module Backward Compatubility Index）

モジュール後方互換性指数は、API、データ、および制御結合に変更がなく、性能低下がないことを意味します。

9. 要件管理解決ツール (Requirements Management Solution Tool)

要求管理解決ツールは以下の特徴をもち、Renesasの複雑な製品ニーズを満たすために使用されます。

- すべての要求を、中央で一元的に記録、検討、レビュー、承認、および管理します。
- カバレッジを理解し、要求、ユースケース、テストケース、不具合、およびその他の関連項目間の関係を追跡して、準拠を示し、品質を保証します。
- ツール共同レビューセンタを使用して、製品納入のためのオープンで現代的な方法を受け入れます。
- 既存の戦略的資産を再利用することで、新しいプロジェクトの設定時間を減らします。
- 組織内の関係者にリアルタイムに製品状態を知らせます。
- 完全なバージョン履歴により、範囲を管理し、変更要求を追跡します。

10. SSPソフトウェア認定および検証パッケージ（SSP Software Qualification and Verification Package）

SSP認定パッケージには以下のものが含まれます。

1. ソフトウェア品質プレゼンテーション
2. ソフトウェア品質ハンドブック
3. ソフトウェア品質サマリレポート

SSPソフトウェア検証パッケージは、秘密保持契約（NDA）により顧客の要求ごとに入手可能です。以下が含まれます。

1. ソフトウェア開発ライフサイクル
2. ソフトウェアコーディング基準
3. ソフトウェア仕様報告
4. ソフトウェア静的解析報告
5. ソフトウェア動的解析報告
6. ソフトウェア性能およびベンチマーク報告
7. ソフトウェア指標（metric）

Appendix A – 模擬検証報告

11. 機能テスト報告（Function Tests Reports）

11.1 カバレッジ報告（Coverage Report）

R_CGC_SystemClockFreqGet
Coverage Metrics required to achieve Decision Coverage Attained

Statement (TER1) = 100 % Branch/Decision (TER2) = 100 %

Statement Coverage Profile

LINE NUMBER REF. (SOURCE)	STATEMENT	PREVIOUS RUNS	CURRENT RUN	COMBINED
27204 (582)	ssp_err_t	0	1	1
27205	R_CGC_SystemClockFreqGet (	0	1	1
27206	cgc_system_clocks_t clock ,	-	-	-
27207	uint32_t * p_freq_hz)	-	-	-
27208 (583)	{	-	-	-
27209 (584)	/* return error if invalid system clock or not supported by hardw	-	-	-
27210 (585)	* p_freq_hz = 0x00 ;	0	1	1
27211 (586)	{	0	1	1
27212	if	0	1	1
27213	(	0	1	1
27214	(	0	1	1
27215	(	0	1	1
27216	(HW_CGC_SystemClockValidCheck (clock))))	0	1	1
27217	)	0	1	1
27218	{	0	1	1
27219	(void) 0 ;	0	1	1
27220	}	0	1	1
27221	else	0	1	1
27222	{	0	1	1
27223	(void) (	0	1	1
27224	(	0	1	1
27225	(	0	1	1
27226	(& g_cgc_version)))) ;	0	1	1
27227	ssp_error_log (	0	1	1
27228	(	0	1	1
27229	(	0	1	1
27230	(SSP_ERR_INVALID_ARGUMENT))) , (	0	1	1
27231	(g_module_name)) , __LINE__) ; ;	0	1	1
27232	return	0	1	1
27233	(	0	1	1
27234	(SSP_ERR_INVALID_ARGUMENT)) ;	0	1	1
27235	}	-	-	-
27236	} ;	0	1	1
27237 (588)	* p_freq_hz = HW_CGC_ClockHzGet (clock) ;	0	1	1
27238 (589)	return	0	1	1
27239	SSP_SUCCESS ;	0	1	1
27240 (590)	}	-	-	-

11.2 検証結果 (Verification Results)

Test Case 1 : Procedure R_CGC_SystemClockSet (r_cgc.c) - PASS

Description

TC_CGC_00018_002_NM_A_02

Verify that the System Clock is successfully set when the specified clock is system clock and the internal clock divider is configured as fastest. is configured as fastest.

Number of Input Parameters : 2
Number of Input Globals : 7
Number of Output Parameters : 0
Number of Output Globals : 9

Return Value

	Type	Expected Value	Actual Value	Status
Procedure Returns	ssp_err_t	SSP_SUCCESS	SSP_SUCCESS	PASS

Input Parameters

Name	Type	Value
clock_source	cgc_clock_t	CGC_CLOCK_MOCO
p_clock_cfg	cgc_system_clock_cfg_t*	&L_sysp_clock_cfg

Input Globals

Name	Type	Value	UsrG
L_sysp_clock_cfg.iclk_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	Yes
L_sysp_clock_cfg.fclk_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	Yes
L_sysp_clock_cfg.bclk_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	Yes
L_sysp_clock_cfg.pclkd_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	Yes
L_sysp_clock_cfg.pclkc_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	Yes
L_sysp_clock_cfg.pclkb_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	Yes
L_sysp_clock_cfg.pclka_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	Yes

Output Globals

Name	Type	Expected Value	Actual Value	Status	UsrG
L_sysp_clock_cfg.fclk_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	CGC_SYS_CLOCK_DIV_1	PASS	Yes
L_sysp_clock_cfg.bclk_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	CGC_SYS_CLOCK_DIV_1	PASS	Yes
L_sysp_clock_cfg.pclkd_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	CGC_SYS_CLOCK_DIV_1	PASS	Yes
L_sysp_clock_cfg.pclkc_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	CGC_SYS_CLOCK_DIV_1	PASS	Yes
L_sysp_clock_cfg.pclkb_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	CGC_SYS_CLOCK_DIV_1	PASS	Yes
L_sysp_clock_cfg.pclka_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	CGC_SYS_CLOCK_DIV_1	PASS	Yes
L_sysp_clock_cfg.iclk_div	cgc_sys_clock_div_t	CGC_SYS_CLOCK_DIV_1	CGC_SYS_CLOCK_DIV_1	PASS	Yes
clock_source	cgc_clock_t	CGC_CLOCK_MOCO	CGC_CLOCK_MOCO	PASS	
R_SYSTEM->SCKSCR_b.CKSEL	uint8_t	0x01	1	PASS	

12. 性能およびベンチマーク報告 (Performance and Benchamrk Report)

Class	Operation	Mode	SSPX.Y.0(e2studioM.N.1.010)						
			IAR						
			S124		S3A7		S5D9	S7G2	
			OHB	OHSpeed, no size constrains	OHB	OHSpeed, no size constrains	OHSpeed, no size constrains	OHB	OHSpeed, no size constrains
EEMBC CoreMark@	S7-DK	CPU 240Mhz Iterations 10000 (time sec)							21
		CPU 160Mhz Iterations 10000 (time sec)							27
		CPU 80Mhz Iterations 6000 (time sec)							32
	S3-DK	CPU 48Mhz Iterations 10000 (time sec)							80
		CPU 24Mhz Iterations 10000 (time sec)							159
	S124-DK	CPU 32Mhz Iterations 10000 (time sec)							167
TM_ThreadX Metrics	basic processing	Cycle count per 30 second		63,841		175,595			638,717
	cooperative scheduling	Cycle count per 30 second		3,353,310		8,179,504			32,934,167
	interrupt preemption proces	Cycle count per 30 second		1,291,687		2,999,190			12,630,826
	interrupt processing	Cycle count per 30 second		4,442,511		10,743,624			45,857,288
	memory allocation	Cycle count per 30 second		5,177,587		12,410,857			51,425,800
	message processing	Cycle count per 30 second		2,787,673		7,121,126			29,236,962
	preemptive scheduling	Cycle count per 30 second		1678080		3,853,070			16,251,622
	synchronization processing	Cycle count per 30 second		5,899,047		13,842,831			61,534,943
FilexX	File number: 200 Transaction number: 500	Time (sec)			28			24	

13. 統合報告（Integration Report）

Module	Class	Description	Compiler	Time Cost(hour)	Margin (Time to solve problem unexpected)	Total Cost(hour) with Margin	Actual Time Cost(hour)	Estimate Completion	Review	Reviewer	Compatibility	Latest SSP Modified on	Ready for ISDE Version	Overall Result
L2_UC_1	ADC Periodic, USBX (Full Speed), Filex(FAT32), EL_GX	Use ADC Periodic framework to detect the frequency of a signal applied to a GPIO on a channel. Write all raw data from ADC during sampling to a FAT32 file on a USB thumb drive in Full Speed. Display detected frequency on the connect LCD. Draw the signal on the connect LCD.	IAR	8	30%	10.4	2	-	Yes	Feder	DK-S7G2	1.1.3	5.0.0.043	PASS
L2_UC_3	FileX(FAT16), Block_media_Interface, SDMMC, Flash, SPI Flash, USBX, jpeg_decode,EL_GX	User select a media SD or MMC User save three jpeg images on selected media from a connected thumb drive User set a time interval in sec User select number of repetition (1, 2, endless) User start a manual slide show where pictures and be changed by a user action on the screen User start a automate slide show where pictures changes within selected time interval and continues for selected number of repetition	IAR	8	30%	10.4	4.5	-	Yes	Feder	DK-S7G2	1.1.3	5.0.0.043	PASS
L2_UC_4	L2_UC_4:EL_GX, 2D Drawing, jpeg_decode, Filex	User select a file format from (FAT12, FAT16, or FAT32) Use select a media to save User Draw a not straight line on screen User save the image to the selected media as a jpeg file	IAR	6	30%	7.8	3	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
L2_UC_7	L2_UC_7: Power Mode Profile, Console, RTC, LPM, IOPORT, and CGC, Messaging	User utilize messaging framework to send command to listed driver using Messaging Framework User receives an acknowledgment for each command through the console	IAR	6	30%	7.8	2	-	Yes	Feder	DK-S7G2	1.1.3	5.0.0.043	PASS
L2_UC_8	L2_UC_8: Netx Client - DHCP, DNS, FTP, TFTP, Telnet, SMTP, POP3, SNTP	BSD IPv4	IAR	2	20%	2.4	2	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		AUTO IP IPv4 only	IAR	2	20%	2.4	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		SMTP Client IPv4	GCC	2	20%	2.4	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		DHCP Client IPv4	GCC	3	20%	3.6	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		HTTP Client IPv4	GCC	2	20%	2.4	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		POP3 Client IPv4	GCC	2	20%	2.4	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		DNS Client IPv4	GCC	2	20%	2.4	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		FTP Client IPv4	GCC	2	20%	2.4	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		TFTP Client IPv4	GCC	2	20%	2.4	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		SNTP Client IPv4	GCC	2	20%	2.4	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		TELNET Client IPv4	GCC	3	20%	3.6	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
		PPP IPv4 only	GCC	6	20%	7.2	1	-	No		DK-S7G2	1.1.3	5.0.0.043	PASS
L2_UC_13	L2_UC_13: NetxDuo Client - DHCP, DNS, FTP, TFTP, Telnet, SMTP, POP3, SNTP	AUTO IP IPv4 only	IAR	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		SMTP Client IPv4	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		SMTP Client IPv6	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		DHCP Client IPv4	GCC	3	20%	3.6	0.5		No		DK-S7G2	1.1.3		PASS
		DHCP Client IPv6	GCC	3	20%	3.6	1		No		DK-S7G2	1.1.3		PASS
		HTTP Client IPv4	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		HTTP Client IPv6	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		POP3 Client IPv4	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		POP3 Client IPv6	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		DNS Client IPv4	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		DNS Client IPv6	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		FTP Client IPv4	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		FTP Client IPv6	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		TFTP Client IPv4	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		TFTP Client IPv6	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		SNTP Client IPv4	GCC	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		SNTP Client IPv6	GCC	2	20%	2.4	0.5		No		DK-S7G2	1.1.3		PASS
		TELNET Client IPv4	GCC	3	20%	3.6	1		No		DK-S7G2	1.1.3		PASS
		TELNET Client IPv6	GCC	3	20%	3.6	0.5		No		DK-S7G2	1.1.3		PASS
		PPP IPv4 only	GCC	6	20%	7.2	1		No		DK-S7G2	1.1.3		PASS
		BSD IPv4	IAR	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS
		BSD IPv6	IAR	2	20%	2.4	1		No		DK-S7G2	1.1.3		PASS

14. 複雑度報告 (Complexity Report)

File and Procedure Results, Metric by Metric

Complexity Metrics (r_cgc.c)

Procedure	Cyclomatic Complexity
R_CGC_Init	4 (P)
R_CGC_ClocksCfg	26 (F)
R_CGC_ClockStart	31 (F)
R_CGC_ClockStop	20 (F)
R_CGC_SystemClockSet	48 (F)
R_CGC_SystemClockGet	1 (P)
R_CGC_SystemClockFreqGet	2 (P)
R_CGC_ClockCheck	11 (F)
R_CGC_OscStopDetect	47 (F)
R_CGC_OscStopStatusClear	9 (P)
R_CGC_BusClockOutCfg	1 (P)
R_CGC_BusClockOutEnable	1 (P)
R_CGC_BusClockOutDisable	1 (P)
R_CGC_ClockOutCfg	1 (P)
R_CGC_ClockOutEnable	1 (P)
R_CGC_ClockOutDisable	1 (P)
R_CGC_LCDClockCfg	7 (P)
R_CGC_LCDClockEnable	5 (P)
R_CGC_LCDClockDisable	5 (P)
R_CGC_SDRAMClockOutEnable	2 (P)
R_CGC_SDRAMClockOutDisable	2 (P)
R_CGC_USBClockCfg	2 (P)
R_CGC_SystickUpdate	4 (P)
R_CGC_VersionGet	1 (P)
r_cgc_stabilization_wait	6 (P)
r_cgc_clock_start_stop	6 (P)
Total for r_cgc.c	220 (P)

15. コーディング基準報告（Coding Standard Report）

Overall Code Review Summary

Totals of Violations for Selected Code Review Standards

'-' indicates required Analysis Phase results are not yet available.
'Off' indicates that the standard is switched off in the Penalty File (<lang>pen.dat).

Number of Violations	LDRA Code	(M) Mandatory Standards	MISRA-C:2012 Code
0	36 S	Function has no return statement.	MISRA-C:2012 R.17.4
0	54 S	Sizeof operator with side effects.	MISRA-C:2012 R.13.6
0	66 S	Function with empty return expression.	MISRA-C:2012 R.17.4
0	407 S	free used on string.	MISRA-C:2012 R.22.2
0	480 S	String function params access same variable.	MISRA-C:2012 R.19.1
0	483 S	free parameter is not heap item.	MISRA-C:2012 R.22.2
0	484 S	Attempt to use already freed object.	MISRA-C:2012 R.22.2
0	496 S	Function call with no prior declaration.	MISRA-C:2012 R.17.3
0	545 S	Assignment of overlapping storage.	MISRA-C:2012 R.19.1
0	591 S	Inappropriate use of file pointer.	MISRA-C:2012 R.22.5
0	614 S	Use of static keyword in array parameter.	MISRA-C:2012 R.17.6
0	631 S	Declaration not reachable.	MISRA-C:2012 R.9.1
0	2 D	Function does not return a value on all paths.	MISRA-C:2012 R.17.4
0	48 D	Attempt to write to unopened file.	MISRA-C:2012 R.22.6
0	51 D	Attempt to read from freed memory.	MISRA-C:2012 R.22.2
0	53 D	Attempt to use uninitialised pointer.	MISRA-C:2012 R.9.1
0	69 D	Procedure contains UR data flow anomalies.	MISRA-C:2012 R.9.1
0	98 D	Attempt to write to file opened read only.	MISRA-C:2012 R.22.4
0	113 D	File closed more than once.	MISRA-C:2012 R.22.6
0	62 X	Function prototype/defn return type mismatch (MR).	MISRA-C:2012 R.8.4
0	63 X	Function prototype/defn param type mismatch (MR).	MISRA-C:2012 R.8.3,R.8.4
0	5 Q	File does not end with new line.	MISRA-C:2012 R.1.3

16. 単体テスト報告 (Unit Test Report)

```
##SSP_VERSION##

Unity test run 1 of 1

TEST(R_AGT_TG1, TC_1_1_GetVersion) PASS
TEST(R_AGT_TG1, TC_1_2_OpenWithValidChan0) PASS
TEST(R_AGT_TG1, TC_1_3_OpenWithValidChan1) PASS
TEST(R_AGT_TG1, TC_1_4_OpenWithInvalidChannel) PASS
TEST(R_AGT_TG1, TC_1_5_OpenWithInvalidHdlPtr) PASS
TEST(R_AGT_TG1, TC_1_6_OpenWithOpenedChannel) PASS
TEST(R_AGT_TG1, TC_1_7_OpenWithOutputCompareDisabled) PASS
TEST(R_AGT_TG1, TC_1_8_OpenWithAGTO_Enabled) PASS
TEST(R_AGT_TG1, TC_1_9_OpenWithAGTIO_Enabled) PASS
TEST(R_AGT_TG1, TC_1_10_OpenWithAGTO_AGTIO_Enabled) PASS
TEST(R_AGT_TG1, TC_1_11_OpenWithValidChan0_one_shot_mode) PASS
TEST(R_AGT_TG1, TC_1_12_OpenWithValidChan0_callback_trigger) PASS
TEST(R_AGT_TG1, TC_1_13_StartStopClear) PASS
TEST(R_AGT_TG1, TC_1_14_SetGetDelay) PASS
TEST(R_AGT_TG1, TC_1_14_CalculateDuty) PASS
TEST(R_AGT_TG1, TC_1_15_CalculateDutyOutputPinCheck) PASS
TEST(R_AGT_TG1, TC_1_16_CascadedTimers) PASS
TEST(DRV_TIMER_TG1, TC_1_1_OpenSuccess) PASS
TEST(DRV_TIMER_TG1, TC_1_2_OpenHdl) PASS
TEST(DRV_TIMER_TG1, TC_1_3_OpenCallbackParameter) PASS
TEST(DRV_TIMER_TG1, TC_1_4_OpenConfigIsNull) PASS
TEST(DRV_TIMER_TG1, TC_1_5_OpenConfigChannelOutOfRange) PASS
TEST(DRV_TIMER_TG1, TC_1_6_OpenOneShot) PASS
TEST(DRV_TIMER_TG1, TC_1_10_OpenTwice) PASS
TEST(DRV_TIMER_TG1, TC_1_11_OpenCallbackIRQNotAvailable) PASS
TEST(DRV_TIMER_TG1, TC_1_12_OpenPeriodTooLarge) PASS
TEST(DRV_TIMER_TG1, TC_1_13_MultipleChannels) PASS
TEST(DRV_TIMER_TG2, TC_2_1_NullPointers) PASS
TEST(DRV_TIMER_TG2, TC_2_2_StartStopClear) PASS
TEST(DRV_TIMER_TG2, TC_2_3_SetGetDelay) PASS
TEST(DRV_TIMER_TG2, TC_2_4_GetVersion) PASS
TEST(DRV_TIMER_TG2, TC_2_5_ControlNotOpen) PASS
TEST(DRV_TIMER_TG2, TC_2_6_infoGet) PASS
TEST(DRV_TIMER_TG3, TC_3_1_CloseSuccess) PASS
TEST(DRV_TIMER_TG3, TC_3_2_CloseHdl) PASS
TEST(DRV_TIMER_TG3, TC_3_3_CloseNotOpen) PASS

-----------------------

36 Tests 0 Failures 0 Ignored 36 Pass

OK
```

17. 回復および品質報告 (Regression and Quality Report)

ソフトウェア品質サマリレポートは、SSPのマイナーリリースごとに入手可能です。報告の入手場所に関する情報については、Synergyソフトウェア品質ウェブページをご覧ください。

改訂履歴

		改訂内容	
Rev.	発行日	ページ	ポイント
1.14	2017.03.08	-	内部版、文書ID：IoT-SQH-00304
1.15	2017.03.13	-	初版
1.16	2017.03.14	すべて	書式改訂、表紙追加
		前付け	承認署名を追加
		40	回帰および品質報告を更新
1.17	2017.03.16	表紙	タイトルに変更
1.18	2017.09.05	40	SSP v1.2.0固有の回帰および品質報告を削除。任意のSSPバージョンの報告を探す指示と入れ替え

ご注意書き

1. 本資料に記載された回路、ソフトウェアおよびこれらに関連する情報は、半導体製品の動作例、応用例を説明するものです。お客様の機器・システムの設計において、回路、ソフトウェアおよびこれらに関連する情報を使用する場合には、お客様の責任において行ってください。これらの使用に起因して生じた損害（お客様または第三者いずれに生じた損害も含みます。以下同じです。）に関し、当社は、一切その責任を負いません。
2. 当社製品、本資料に記載された製品データ、図、表、プログラム、アルゴリズム、応用回路例等の情報の使用に起因して発生した第三者の特許権、著作権その他の知的財産権に対する侵害またはこれらに関する紛争について、当社は、何らの保証を行うものではなく、また責任を負うものではありません。
3. 当社は、本資料に基づき当社または第三者の特許権、著作権その他の知的財産権を何ら許諾するものではありません。
4. 当社製品を、全部または一部を問わず、改造、改変、複製、リバースエンジニアリング、その他、不適切に使用しないでください。かかる改造、改変、複製、リバースエンジニアリング等により生じた損害に関し、当社は、一切その責任を負いません。
5. 当社は、当社製品の品質水準を「標準水準」および「高品質水準」に分類しており、各品質水準は、以下に示す用途に製品が使用されることを意図しております。

   標準水準： コンピュータ、OA機器、通信機器、計測機器、AV機器、
   家電、工作機械、パーソナル機器、産業用ロボット等

   高品質水準： 輸送機器（自動車、電車、船舶等）、交通制御（信号）、大規模通信機器、
   金融端末基幹システム、各種安全制御装置等

   当社製品は、データシート等により高信頼性、Harsh environment向け製品と定義しているものを除き、直接生命・身体に危害を及ぼす可能性のある機器・システム（生命維持装置、人体に埋め込み使用するもの等）、もしくは多大な物的損害を発生させるおそれのある機器・システム（宇宙機器と、海底中継器、原子力制御システム、航空機制御システム、プラント基幹システム、軍事機器等）に使用されることを意図しておらず、これらの用途に使用することは想定していません。たとえ、当社が想定していない用途に当社製品を使用したことにより損害が生じても、当社は一切その責任を負いません。
6. 当社製品をご使用の際は、最新の製品情報（データシート、ユーザーズマニュアル、アプリケーションノート、信頼性ハンドブックに記載の「半導体デバイスの使用上の一般的な注意事項」等）をご確認の上、当社が指定する最大定格、動作電源電圧範囲、放熱特性、実装条件その他指定条件の範囲内でご使用ください。指定条件の範囲を超えて当社製品をご使用された場合の故障、誤動作の不具合および事故につきましては、当社は、一切その責任を負いません。
7. 当社は、当社製品の品質および信頼性の向上に努めていますが、半導体製品はある確率で故障が発生したり、使用条件によっては誤動作したりする場合があります。また、当社製品は、データシート等において高信頼性、Harsh environment向け製品と定義しているものを除き、耐放射線設計を行っておりません。仮に当社製品の故障または誤動作が生じた場合であっても、人身事故、火災事故その他社会的損害等を生じさせないよう、お客様の責任において、冗長設計、延焼対策設計、誤動作防止設計等の安全設計およびエージング処理等、お客様の機器・システムとしての出荷保証を行ってください。特に、マイコンソフトウェアは、単独での検証は困難なため、お客様の機器・システムとしての安全検証をお客様の責任で行ってください。
8. 当社製品の環境適合性等の詳細につきましては、製品個別に必ず当社営業窓口までお問合せください。ご使用に際しては、特定の物質の含有・使用を規制するRoHS指令等、適用される環境関連法令を十分調査のうえ、かかる法令に適合するようご使用ください。かかる法令を遵守しないことにより生じた損害に関して、当社は、一切その責任を負いません。
9. 当社製品および技術を国内外の法令および規則により製造・使用・販売を禁止されている機器・システムに使用することはできません。当社製品および技術を輸出、販売または移転等する場合は、「外国為替及び外国貿易法」その他日本国および適用される外国の輸出管理関連法規を遵守し、それらの定めるところに従い必要な手続きを行ってください。
10. お客様が当社製品を第三者に転売等される場合には、事前に当該第三者に対して、本ご注意書き記載の諸条件を通知する責任を負うものといたします。
11. 本資料の全部または一部を当社の文書による事前の承諾を得ることなく転載または複製することを禁じます。
12. 本資料に記載されている内容または当社製品についてご不明な点がございましたら、当社の営業担当者までお問合せください。

注1. 本資料において使用されている「当社」とは、ルネサス エレクトロニクス株式会社およびルネサス エレクトロニクス株式会社が直接的、間接的に支配する会社をいいます。

注2. 本資料において使用されている「当社製品」とは、注1において定義された当社の開発、製造製品をいいます。

(Rev.4.0-1 2017.11)

ルネサス エレクトロニクス株式会社

■営業お問合せ窓口

http://www.renesas.com

※営業お問合せ窓口の住所は変更になることがあります。最新情報につきましては、弊社ホームページをご覧ください。

ルネサス エレクトロニクス株式会社　〒135-0061　東京都江東区豊洲3-2-24（豊洲フォレシア）

■技術的なお問合せおよび資料のご請求は下記へどうぞ。
総合お問合せ窓口：https://www.renesas.com/contact/

Renesas Synergy™ Platform
Synergyソフトウェア品質ハンドブック