AUTOSAR and the LDRA tool suite®

A more complete solution

AUTOSAR Classic Platform、Adaptive Platform を活用するシステムの

機能安全規格 ISO 26262 への準拠をどのように支援できるかについて紹介します。

www.ldra.com

目次

背景

AUTOSAR（Automotive Open System Architecture）は、大手自動車 OEM およびサプライヤのグループによる標準化イニシアチブで、2003 年秋に設立されました。その継続的な使命は ECU ソフトウェアのための参照アーキテクチャを開発することであり、それは現代の自動車におけるソフトウェアの増大する複雑さを克服することができます。

2017 年 12 月に AUTOSAR 規格ファミリは、既存の「Classic Platform」に新しい「Adaptive Platform」が追加拡張されました。（図 1）

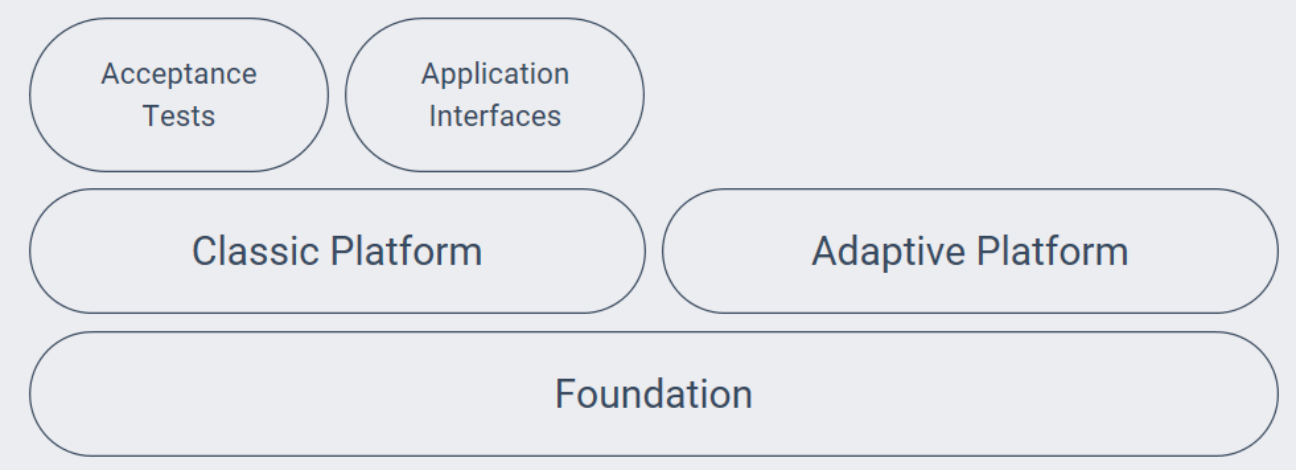

図 1: AUTOSAR 規格ファミリ

Classic Platform[1]は、2005 年に最初に発表された、リアルタイムと安全性に厳しい制約がある組み込みシステム向けの AUTOSAR の定評あるソリューションです。

Adaptive Platform[2] は、高度に自動化された自動運転などのユースケースのための安全関連システムを構築するための高性能コンピューティング ECU 用の AUTOSAR のソリューションです。その発表時に、元 AUTOSAR スポークスマンの Thomas Rüping 氏は次のように言及しました「新しい要求は新しいソリューションを要求します…　AUTOSAR はまた、コネクティビティと高度に自動化された自動運転の分野における新しいアプリケーションに最適な標準化ソフトウェアフレームワークを提供することを目指します。」[3]
Adaptive Platform は AUTOSAR Classic Platform を補完するものであり、それに代わるものではありません。

---

[1] AUTOSAR Classic Platform https://www.autosar.org/standards/classic-platform/
[2] AUTOSAR Adaptive Platform https://www.autosar.org/standards/adaptive-platform/
[3] http://www.ai-online.com/Adv/Previous/show_issue.php?id=6881#sthash.Cvo3ABXn.0xLSMTMG.dpbs

Foundation[4] 規格の目的は、AUTOSAR プラットフォーム間の相互運用性を強化することです。通信プロトコルなど、AUTOSAR プラットフォームに共通の要件と技術仕様が含まれています。

この資料では LDRA tool suite® のような統合された包括的なツールのセットが、コンプライアンスへの道を容易にするのをどのように支援できるかについて紹介します。AUTOSAR 規格に関連する文書は多くの巻に及んでおり、それらすべてをこの資料で参照することはできません。

AUTOSAR と ISO26262

どちらの AUTOSAR プラットフォーム規格に準拠していても、それ自体が自動車の機能安全規格「ISO 26262」[5] に準拠していることを意味するわけではありませんので、ISO 規格も同時に検討する必要があります。

ISO 26262：2011「自動車の機能安全」は、自動車の E/E/PE システムの複雑さが急増し、それに関連して公共の安全性が損なわれるリスクに対応して公開されました。それ以前の鉄道、医療機器、産業システムと同様に、自動車業界でも業界標準の機能安全規格 IEC 61508[6] に基づいて規格が定められました。結果 ISO 26262 は、自動車の機能安全規格の主流となり、その要件とプロセスは業界全体でますます一般的になりつつあります。

ISO 26262：2011 は 10 のパートで構成され、3 つは製品開発に重点を置いています。システムレベル（Part 4）[7]、ハードウェアレベル（Part 5）[8]、およびソフトウェアレベル（Part 6）[9] です。 ISO 26262-6：2011 は、安全性が重要であるかどうかにかかわらず、自動車用システムおよび機器用のすべてのソフトウェアの製造に関する詳細な業界固有のガイドラインを提供します。

システム全体の ISO 26262-4：2011 と、ISO 26262-6：2011 にあるソフトウェア固有のサブフェーズとの間の関係は、V モデルで表すことができます。図 2 は、開発ライフサイクルを通して LDRA と他の補完的なツールをどのように使用できるかを示しています。

---

[4] AUTOSAR Foundation https://www.autosar.org/standards/adaptive-platform/

[5] International standard ISO 26262 Road vehicles — Functional Safety

[6] IEC 61508:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems

[7] ISO 26262-4:2011 Road vehicles -- Functional safety -- Part 4: Product development at the system level

[8] ISO 26262-5:2011 Road vehicles -- Functional safety -- Part 5: Product development at the hardware level

[9] ISO 26262-6:2011 Road vehicles -- Functional safety -- Part 6: Product development at the software level

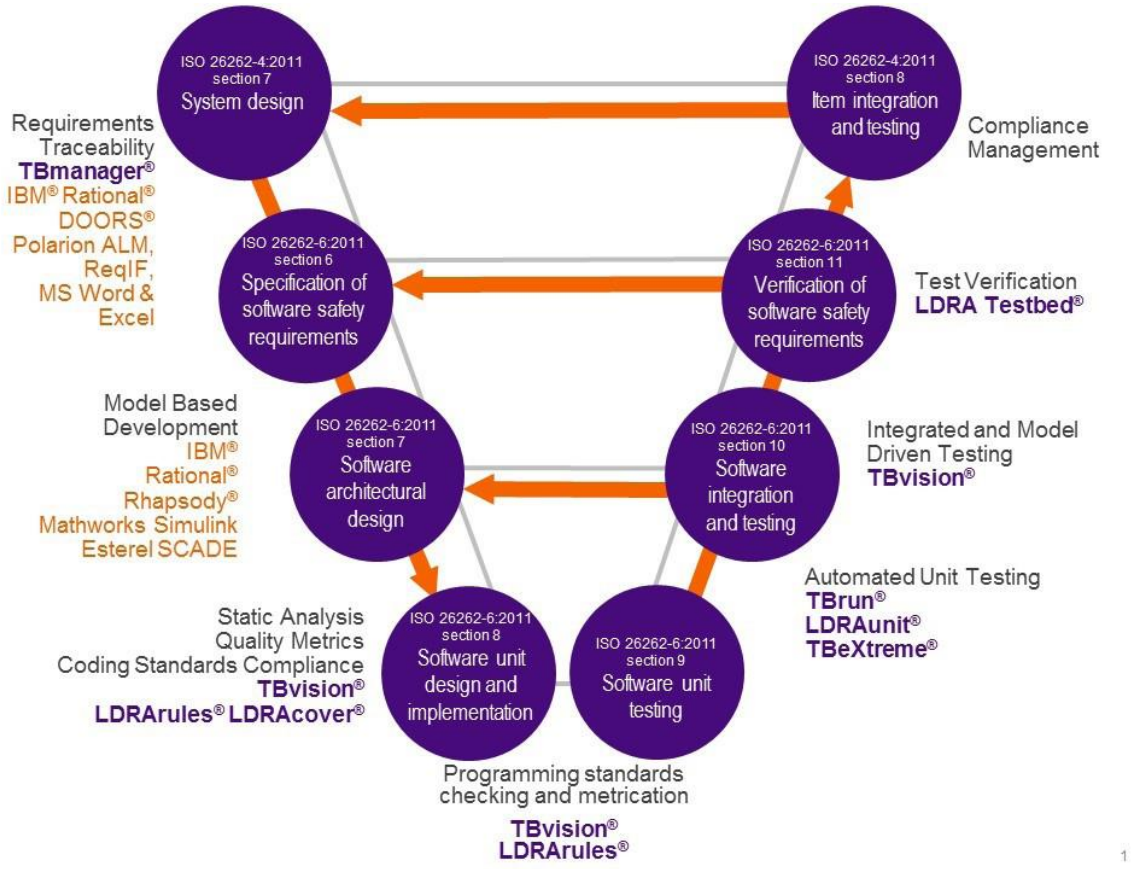

図 2: ISO 26262 と標準の開発ツールに関連付けたソフトウェア開発の V モデル

AUTOSAR Classic Platform：前提

従来、自動車用アプリケーションのソフトウェア開発は、OEM またはそのサプライヤによって個別に行われていました。これは自動車分野の関係者間で多くの重複した努力、長期の開発サイクル、そして特にハードウェアかソフトウェアのどちらかへの更新が必要とされるときの商業的柔軟性の欠如をもたらしました。

AUTOSAR Classic Platform は、ハードウェアとの間に明確に定義された抽象化レイヤを提供し、選択されたマイクロコントローラに関係なく共通のソフトウェア基盤を提供することによって、これらの欠如に対処しようとしています（図 3）。

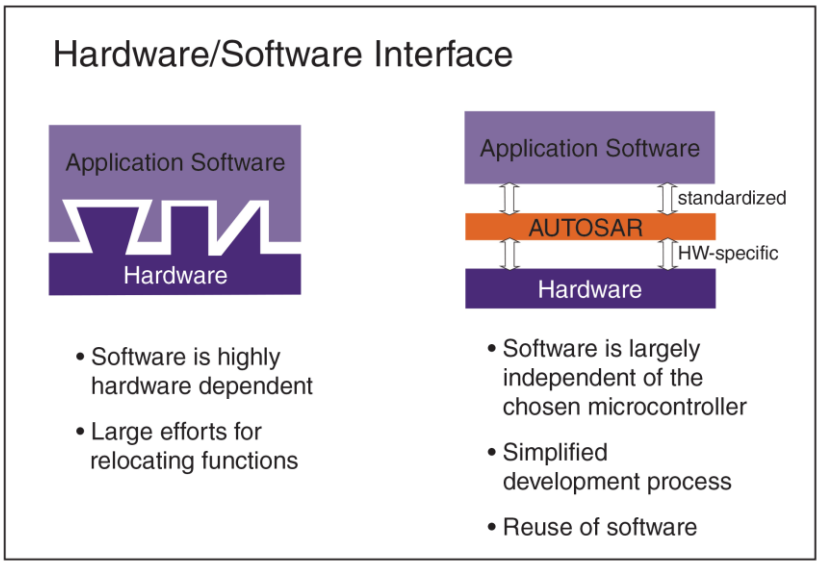

図 3: AUTOSAR Classic Platform[10]の基本的な前提

AUTOSAR Classic Platform での作業

AUTOSAR Classic Platform のアプリケーション開発

3 つの相互接続ソフトウェアユニットからなるシステム例を考えてみましょう（図 4)。各ソフトウェアコンポーネントはアプリケーションの一部をカプセル化し、コンポーネントはハードウェアに依存しない仮想機能バス（Virtual Function Bus）を介して互いに通信します。

各 ECU は 1 つまたは複数のソフトウェアユニットをホストでき、それぞれが仮想機能バスと AUTOSAR ベーシックソフトウェア（Basic Software）を実装するためのランタイム環境（Runtime Environment）をホストします。

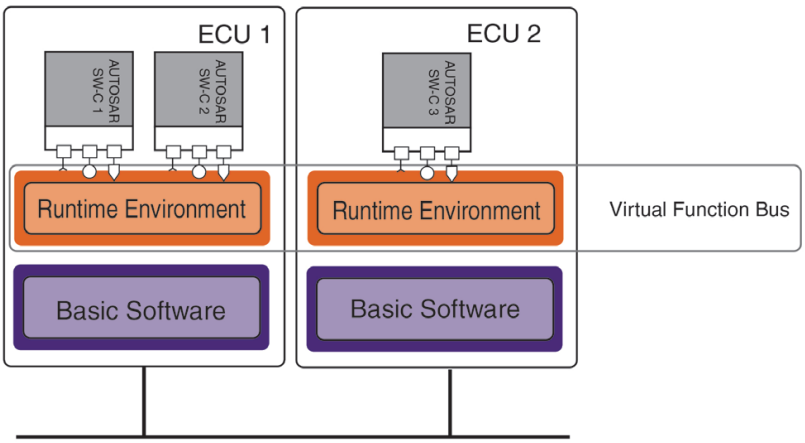

図 4: ECU でのソフトウェアの物理的な配置

[10] AUTOSAR – A First Glance https://www.youtube.com/watch?v=F27jtKkxbAo

この BASIC software は、オペレーティングシステムサービス、ネットワーク通信および管理サービス、メモリサービス、診断サービスなどのサービス機能をアプリケーション開発者に提供します。このように、ハードウェア環境はソフトウェアユニットから抽象化されているため、アプリケーション開発者にとってはほとんど透過的です。

BASIC software は層に細分されています。(図 5)

- Microcontroller Abstraction Layer（MCAL）は、マイクロコントローラのメモリ、通信および入出力（I/O）へのアクセスのためのドライバを提供する
- ECU Abstraction Layer（ECUAL）は、通信、メモリ、I/O など、ECU のすべての機能への統一的なアクセスを、これらの機能が物理的に実装されている場所に関係なく提供する
- Service Layer は、アプリケーション層にオペレーティングシステムと、ネットワークサービス、メモリ管理、バス通信などのバックグラウンドサービスの両方を提供する
- Runtime Environment（RTE）は、ベーシックソフトウェアからアプリケーション層を抽象化し、アプリケーション層の実行時動作を制御し、データ交換を実装する

アプリケーション層では、ECU のアプリケーション機能は個々の AUTOSAR ソフトウェアコンポーネント（SWC）の形で実装されます。

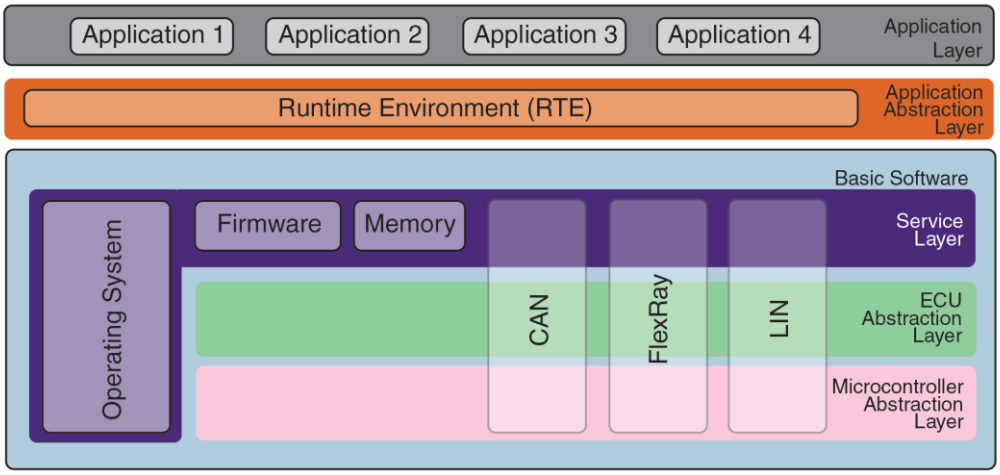

図 5: AUTOSAR Classic Platform のアーキテクチャ

実際には、このアーキテクチャは、これらの AUTOSAR ソフトウェアコンポーネント（SWC）が移転可能であり、開発プロセスの非常に遅い時期に ECU に展開できるように、あるレベルの抽象化を提供します[11]。AUTOSAR コンポーネントの依存関係は、仮想機能バスを参照してインターフェイスおよびポートの形式で説明されています。内部の隠れた依存関係は許されません。

[11] An introduction to AUTOSAR https://retis.sssup.it/sites/default/files/lesson19_autosar.pdf

その結果、同じロジックを実装し、同じパブリック通信インターフェイスを残りのシステムに提供するコンポーネントは交換可能です。そのようなコンポーネントはすべて同じコンポーネントタイプであると言われ、各コンポーネントタイプは Software Component Template[12] － それ自体が 800 ページを超える長さの文書 － に従って定義されます。

AUTOSAR Classic Platform と ISO 26262、そして LDRA tool suite

ISO 26262 の V モデル（図 2）の AUTOSAR の影響を特に受ける要素を参照して、完成したソフトウェアアーキテクチャ設計から開始して、その後の作業に焦点を当てた開発ライフサイクル（ソフトウェアユニットの設計と実装、ソフトウェアユニットのテスト、ソフトウェアの統合とテスト）を検討することが役立ちます。

ソフトウェアユニットの設計と実装（ISO 26262-6:2011 section 8）

図 6 は、ISO 26262-6：2011 の典型的な例です。実装中に適用されるコーディングおよびモデリングのガイドラインを示し、自動ツールを使用してコンプライアンスを確認できる場所を示しています。

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity	++✓	++✓	++✓	++✓
1b	Use of language subsets	++✓	++✓	++✓	++✓
1c	Enforcement of strong typing	++✓	++✓	++✓	++✓
1d	Use of defensive implementation techniques	o	+	++	++
1e	Use of established design principles	+✓	+✓	+✓	++✓
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+✓	++✓	++✓	++✓
1h	Use of naming conventions	++✓	++✓	++✓	++✓

”++” The method is highly recommended for this ASIL.
“+“ The method is recommended for this ASIL.
“o“ The method has no recommendation for or against its usage for this ASIL.
✓ Satisfied by the LDRA tool suite

図 6: LDRA tool suite の機能と ISO 26262-6:2011[13]「Table 6: Methods for the verification of the software architectural design”」との対応

これらのガイドラインを組み合わせることで、生成されるコードの信頼性が高まり、エラーが発生しにくくなり、テストが容易になり、メンテナンスが容易になります。ピアレビューはそのようなガイドラインに従うことを実施するための伝統的なアプローチですが、他にも多くの重要な作業を抱える開発者に

[12] AUTOSAR Software Component Template https://www.autosar.org/fileadmin/user_upload/standards/classic/4-1/AUTOSAR_TPS_SoftwareComponentTemplate.pdf

[13] ISO 26262-6:2011, Copyright© 2015 IEC, Geneva, Switzerland の Table 6 に基づく。了承済み

とって、退屈なチェックをツールで自動化することははるかに効率的で、間違いにくく、何度でもできて、明白な方法です。図 7 は、言語サブセットの違反がどのように表示されるかを示す一例です。

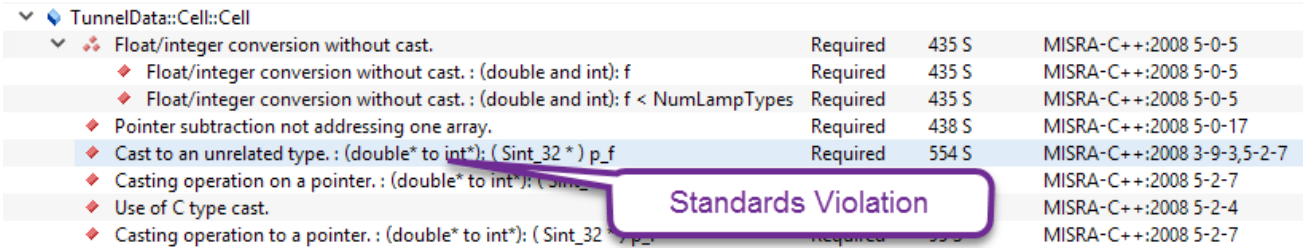

図 7: LDRA tool suite でのコーディングガイドライン違反の強調表示

ISO 26262-6：2011 は、使用される可能性があるものの例として MISRA[14] 言語サブセットを参照しています。特定のアプリケーションにより適したものにするために、そのような標準セットを操作、調整、追加することや、社内で独自のルールセットを使用することも認められています。(図 8)

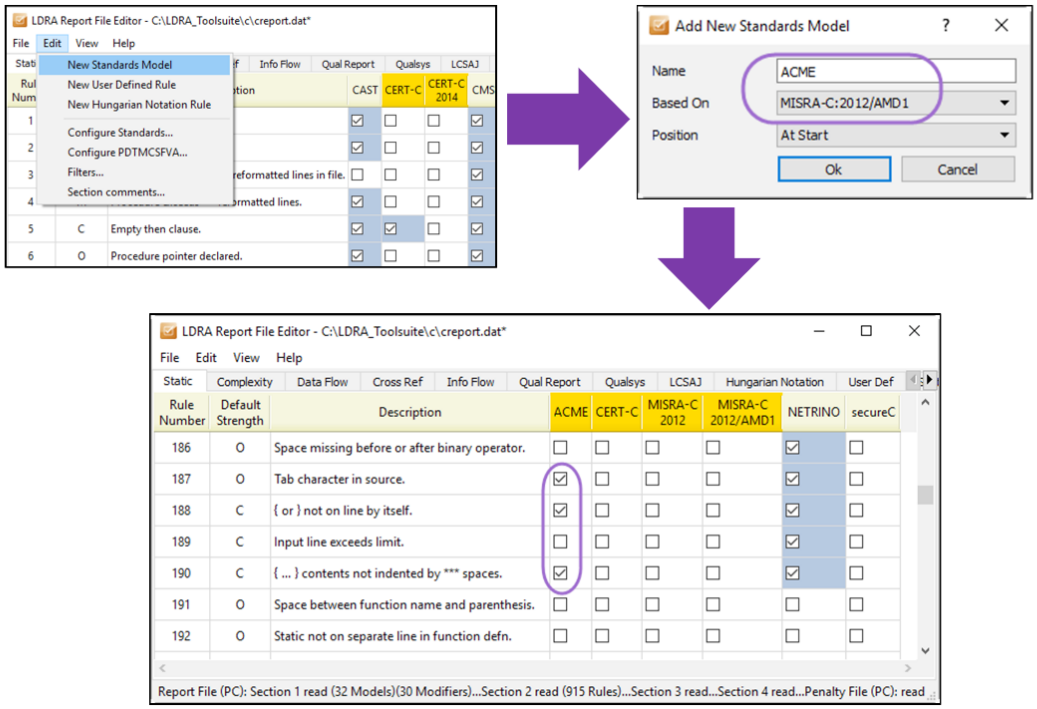

図 8: LDRA tool suite を使用して既存の標準モデルを基礎に新しい標準を作成

AUTOSAR Classic Platform と MISRA 標準の両方を使用する場合、克服すべき 2 番目の課題として、AUTOSAR システムコールの標準形式が MISRA に準拠していないという点があります。LDRA tool suite は、AUTOSAR システムコールを認識し、MISRA 関連の違反として強調表示しないように構成されています。

[14] MISRA – The Motor Industry Software Reliability Association https://www.misra.org.uk/

ソフトウェアアーキテクチャ設計とユニット実装

コーディング、アーキテクチャ設計、ユニット実装に関する適切なプロジェクトガイドラインの確立は、明らかに 3 つの個別のタスクですが、設計から実装を行うソフトウェア開発者は、それらすべてを同時に意識する必要があります。

これらのガイドラインは、結果として得られるコードの信頼性を高め、エラーを起こしにくく、テストしやすく、保守しやすくするという考え方にも基づいています。 たとえば、アーキテクチャガイドラインには以下が含まれます。

- 大規模でまとまりのない関数では読み取り、保守、およびテストが難しく、エラーが発生しやすいため、**ソフトウェアコンポーネントの制限されたサイズ**と**インターフェイスの制限されたサイズ**が特に推奨される。

- **各ソフトウェアコンポーネント内の高い凝集**　ソフトウェアプログラムのモジュール間の緊密なリンクにより、高い凝集力が生じる。これは、割り当てられたさまざまなタスクをどれだけ迅速に実行できるかに影響する。

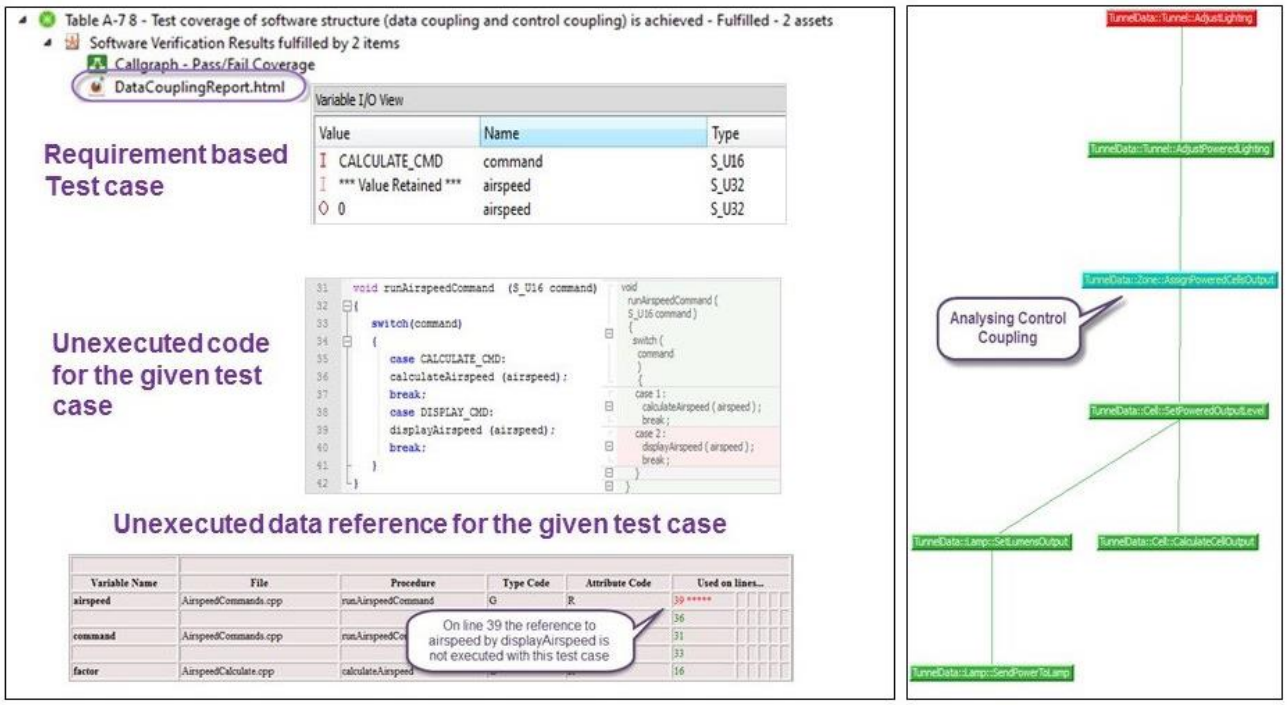

図 9: LDRA tool suite が表示する制御結合、データ結合解析の出力

静的解析ツールは、インターフェイス分析の成果として複雑性メトリック、データオブジェクト分析を通じて評価された凝集メトリック、データおよび制御結合分析による結合メトリックなど、標準への準拠を保証するメトリックを提供できます。(図 9)

より一般的には、静的解析は、ISO 26262：2011 で要求されるグッドプラクティスがコーディングルール、設計原則、またはソフトウェアアーキテクチャ設計の原則に準拠していることを確認するのに役立ちます。

実際には、ISO 26262 に新参である開発者にとって、このようなツールの役割は、違反を強調するメカニズムから、違反がないことを確認する手段へと進化することがよくあります。

ソフトウェア単体テスト（ISO 26262-6:2011 section 9）およびソフトウェア統合とテスト（ISO 26262-6:2011 section 10）

静的解析手法（ソースコードの自動検査）がコーディング、アーキテクチャ設計、ユニット実装のサブフェーズ全体に適用できるように、動的解析手法（コードの一部またはすべての実行を含む）はユニット、統合、システムのテストに適用できます。単体テストは、特定のソフトウェア手順または機能に単独で焦点を当てるように設計されますが、統合テストでは、ソフトウェアアーキテクチャ設計に従ってユニットが連携して動作することを確認することで、安全性要件と機能要件が満たされます。

ISO 26262-6：2011 の表には、ターゲットハードウェアで単体テストと統合テストを実行するためのテクニックとメトリックがリストされており、安全性要件と機能要件が満たされ、ソフトウェアインターフェイスが検証されていることを確認します。フォールトインジェクションとリソーステストは、堅牢性と復元力をさらに証明し、該当する場合は、モデルとコードの Back-to-Back テストが設計の正しい解釈を証明（一致性の検証）することに役立ちます。これらの手法に伴う成果物は、それらの管理の参照と完了の証拠の両方を提供します。これらには、ソフトウェアユニットの設計仕様、テスト手順、検証計画、検証仕様が含まれます。各テスト手順を完了すると、合格/不合格の結果が報告され、要件への準拠が適切に検証されます。

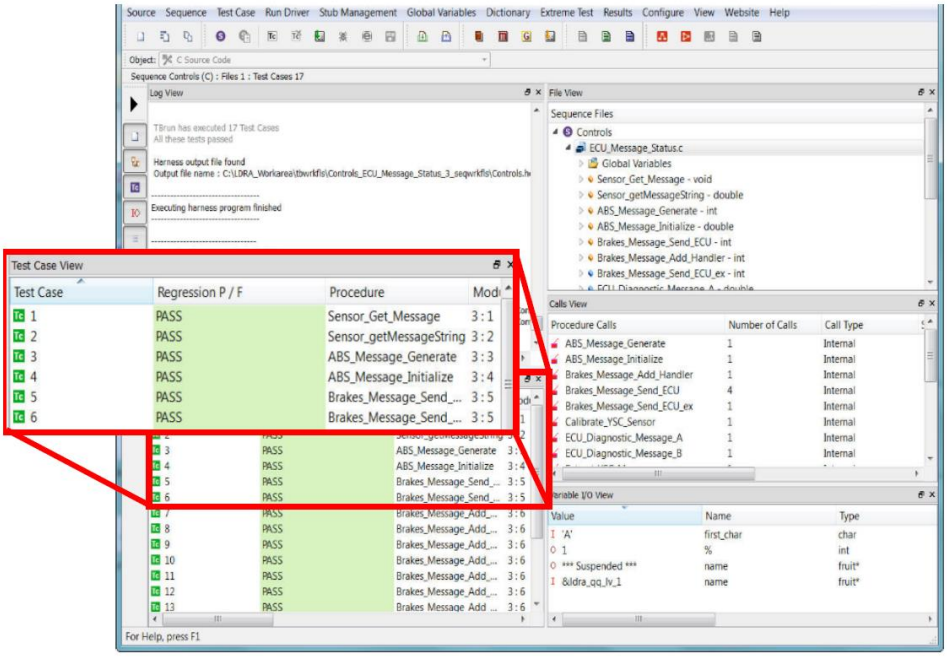

図 10: LDRA tool suite を使用した要件ベース単体テストの実施

図 10 の例は、関数スコープでソフトウェアインターフェイスが表示され、ユーザーが入力値と期待値を入力してテストハーネスの基礎を形成する様子を示します。このテストハーネスがコンパイルされて、ターゲットハードウェアで実行され、実際の出力と期待値が比較されます。

ユニットを「スタブ化」するのではなく、コールツリーの一部として導入すると、ユニットテストは統合テストになります。両方のケースでコードの検証にまったく同じテストデータを使用できます。

最小値、下限値を下回る値、下限値、上限値、上限値を超える値などの同値クラスの境界値は、関連する入力データを備えた一連の単体テストケースを自動生成することで分析できます。

テストの失敗や、顧客からの要件変更に応じる必要がある場合、影響を受けるすべてのユニットおよび統合テストを再実行（回帰テスト）する必要があります。確立された機能が変更によって損なわれないことの確認を、ツールを使用してテストを自動的に再適用することで行えます。

ISO 26262：2011 で促進されるいずれのテストでも、ソフトウェアテストツールを活用することが求められている訳ではありませんが、静的解析と同様に、動的解析ツールは、特に大規模なプロジェクトの場合、テストプロセスをはるかに効率的にするのに役立ちます。

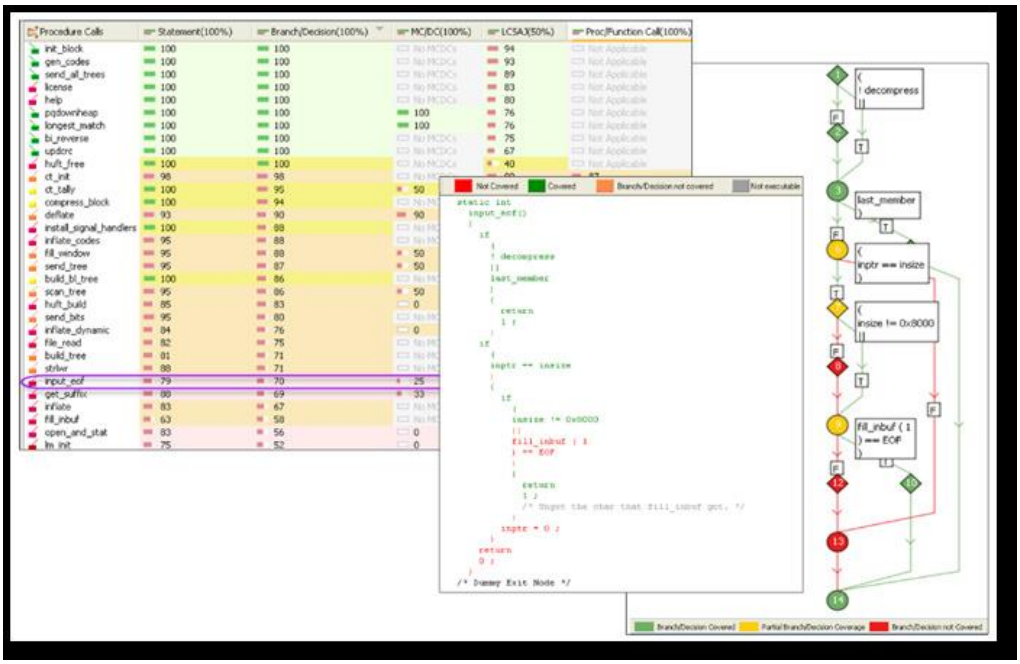

図 11: LDRA tool suite での構造カバレッジの表示例

構造カバレッジメトリック

ソフトウェアが正しく機能することを示すことに加えて、動的解析を使用して、構造カバレッジメトリックを生成します。ソフトウェアユニットレベルでの要件のカバレッジと併せて、これらのメトリックは、テストケースの完全性を評価し、意図しない機能がないことを実証するために必要な情報となります。（図 11）

ISO 26262：2011 で推奨されるカバレッジのメトリックには、関数、コール、ステートメント、ブランチ、および MC/DC があります。ユニットおよびシステムテストの実行には同じ環境が利用できるため、たとえば、動的システムテストを通じてほとんどのソースコードのカバレッジデータを生成して、通常のシステム操作で到達しない防御的な機能などにユニットテストを使用して補完することができます。

これら静的および動的機能は、IBM Rational Rhapsody[15]、MathWorks Simulink[16]、Esterel SCADE[17] などのいくつかの異なるモデルベースの開発ツールと統合できます。開発フェーズ自体は、通常の方法でモデルが作成されますが、モデルからソースコードが自動生成されてから統合されます。

図 12 は、Back-to-back テストに適したアプローチを使用して、Simulink との統合を展開する方法を示しています。設計モデルは Simulink で開発され、Simulink 上でのテストで検証されます。 次に、Simulink から生成されるコードに対して、LDRA tool suite によってカバレッジ測定のためのインスツルメントが行われたあと、Software In the Loop（SIL かホスト）または Processor In the Loop（PIL かターゲット）モードで実行されます。そして構造化カバレッジの結果が収集され、Simulink およびソースコードの動的解析により、ソースコードレベルで構造化カバレッジレポートが生成されます。

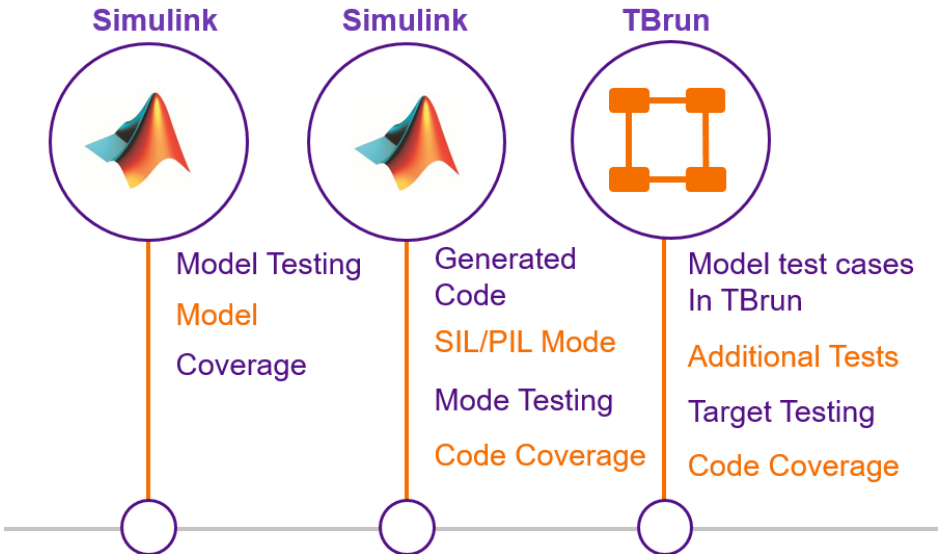

図 12: MathWorks Simulink と LDRA tool suite による自動生成コードの構造カバレッジデータ生成

この統合では、他のいくつかのテストも利用できます。生成されたソースコードを静的に解析して、MISRA C：2012 付録 E[18] などの適切なコーディング標準に準拠していることを確認できます。追加の動

[15] http://www-03.ibm.com/software/products/en/ratirhapfami/

[16] https://www.mathworks.com/products/simulink.html

[17] http://www.esterel-technologies.com/products/scade-suite/

[18] https://www.misra.org.uk/tabid/72/Default.aspx

的テストは、LDRA tool suite からソースレベルで実行できます。要件ベースのテストを作成して、機能を検証し、構造的カバレッジを照合できます。テストデータを Simulink からインポートして、LDRA tool suite に移行して効率を高めることもできます。

自動生成されるコードがベースになるリアルタイム組み込みシステムでも、通常、従来のレベルで人手により実装されたコードもある程度含まれています。ボードサポートパッケージ、割り込みハンドラー、ドライバ、およびその他の低レベルコード用のソフトウェアは、通常、手動でコーディングされます。ほとんどの場合、レガシーコードもシステムの一部に利用されます。システムのこれらの部分は、自動生成されるコードとともに、LDRA tool suite を従来どおりの方法で使用して検証することができます。

AUTOSAR Adaptive Platform：変化する世界への適応

AUTOSAR Classic Platform は実績のあるものとなりましたが、コネクテッドカーの出現によりその限界も明らかになりました。現在、BMW グループの Learning, Reasoning and Knowledge Representation のゼネラルマネージャーであり、元 AUTOSAR のスポークスマンの Simon Fürst 氏は、次のように言及しています「AUTOSAR Classic Platform…は、自動車の従来の E/E ドメインの制御ユニットに関して、引き続き最適なシステムです。」

「このプラットフォームは、安全要件を満たし、ハードリアルタイムの機能を提供しながら、ハードウェアリソースへの要求が限られているアプリケーションに適したソリューションであり続けます。」「今後の新しい機能は、特別な要件に合わせて設計されたソフトウェアプラットフォームによって、より効率的に実現されます。一方で、例えば複雑な数値演算アルゴリズムや高い相互接続性が求められますが、他方では、アプリケーションとシステムの信頼性、または「無線経由の更新」などの新しい技術に焦点が当てられます。」[19]

AUTOSAR Adaptive Platform での作業

もちろん、AUTOSAR Adaptive Platform 用に作成されたアプリケーションも ISO 26262 準拠を必要とする可能性が高く、図 2 で概説したすべてのサポートツールと、Classic Platform アプリケーションを参照して説明したテストプロセスの多くは、ここでも等しく適用することができます。

ただし、AUTOSAR Adaptive Platform には、アプリケーション開発者の観点から強調すべき重要な要素がいくつかあります。図 13 は、階層化されたアーキテクチャを示しています。

[19] http://www.ai-online.com/Adv/Previous/show_issue.php?id=6881#sthash.Cvo3ABXn.0xLSMTMG.dpbs

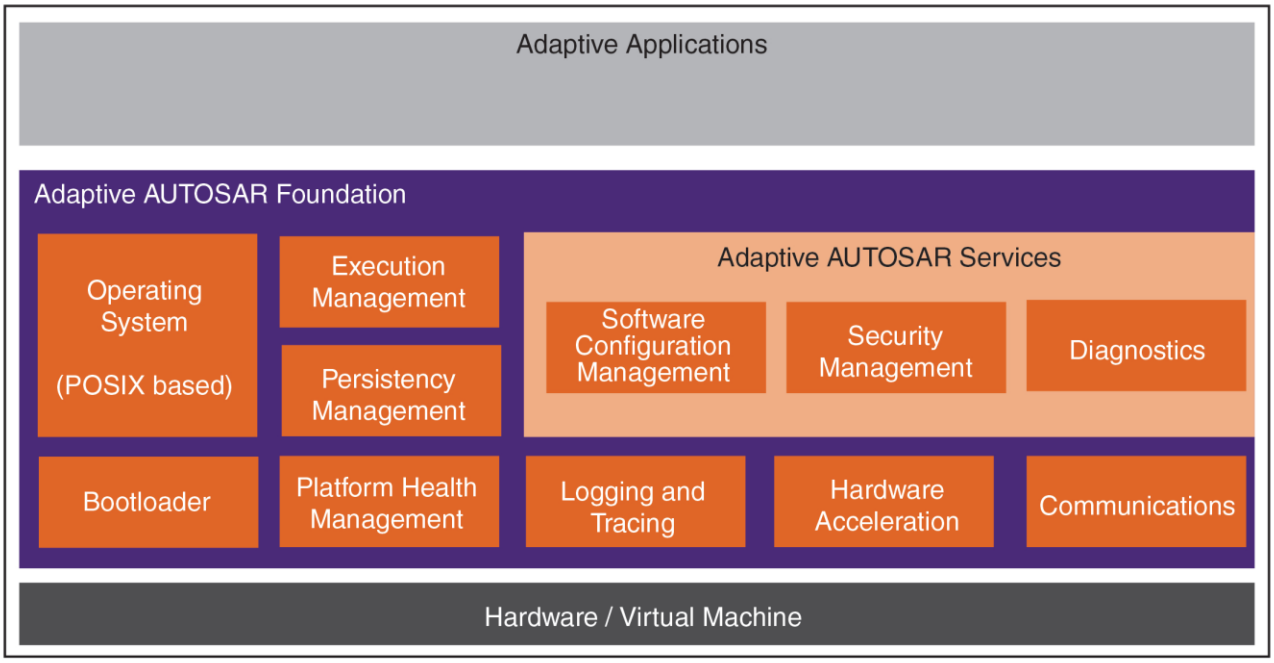

図 13: Adaptive AUTOSAR のアーキテクチャ

通信ミドルウェアは、Adaptive Platform を補完し、アプリケーション間の通信メカニズムを提供するように設計された新しい AUTOSAR 標準である ara::com[20] の形式をとります。ara::com は、Adaptive Applications が提供するサービスの 1 つに過ぎず、AUTOSAR の主要な要件のすべてが既存のソリューションで満たされているわけではないと考えられたために作成されました。

AUTOSAR Classic アプリケーションとは異なり、Adaptive アプリケーションは、モノリシックな実行可能ファイルを作成するために一緒にコンパイルされる多数のソースファイルでは構成されていません。代わりに、個別に実行可能なシングルまたはマルチスレッドのプロセスです。Adaptive アプリケーションは、コンパイル時ではなく、Manifest ファイルを使用してターゲット上に構成されます。そして C++ 言語が、オブジェクト指向の利点から C よりも優先されます。また、オーダーメイドの AUTOSAR Classic オペレーティングシステムとは異なり、Adaptive AUTOSAR は多くの POSIX PSE 51 準拠 OS のいずれかを使用します。

Classic Platform の開発ライフサイクルと比較して、これらの違いの 2 つを反映することは有用です。

[20] AUTOSAR Explanation of ara:comm API

POSIX 準拠

QNX Neutrino[21] や Lynx LynxOS[22] などのネイティブ POSIX 準拠の RTOS に加えて、Green Hills INTEGRITY[23] や Wind River VxWorks などの POSIX 準拠製品が数多くあります。また、Automotive Grade Linux[24] などの Linux プロバイダーからの Linux Standard Base 準拠オプションもあります。

これらはすべて、それ自体が十分に確立された RTOS であるため、LDRA から既存の実証済みのサポートがあり、開発ライフサイクル全体にわたってテスト支援機能を提供しています（図 14）。

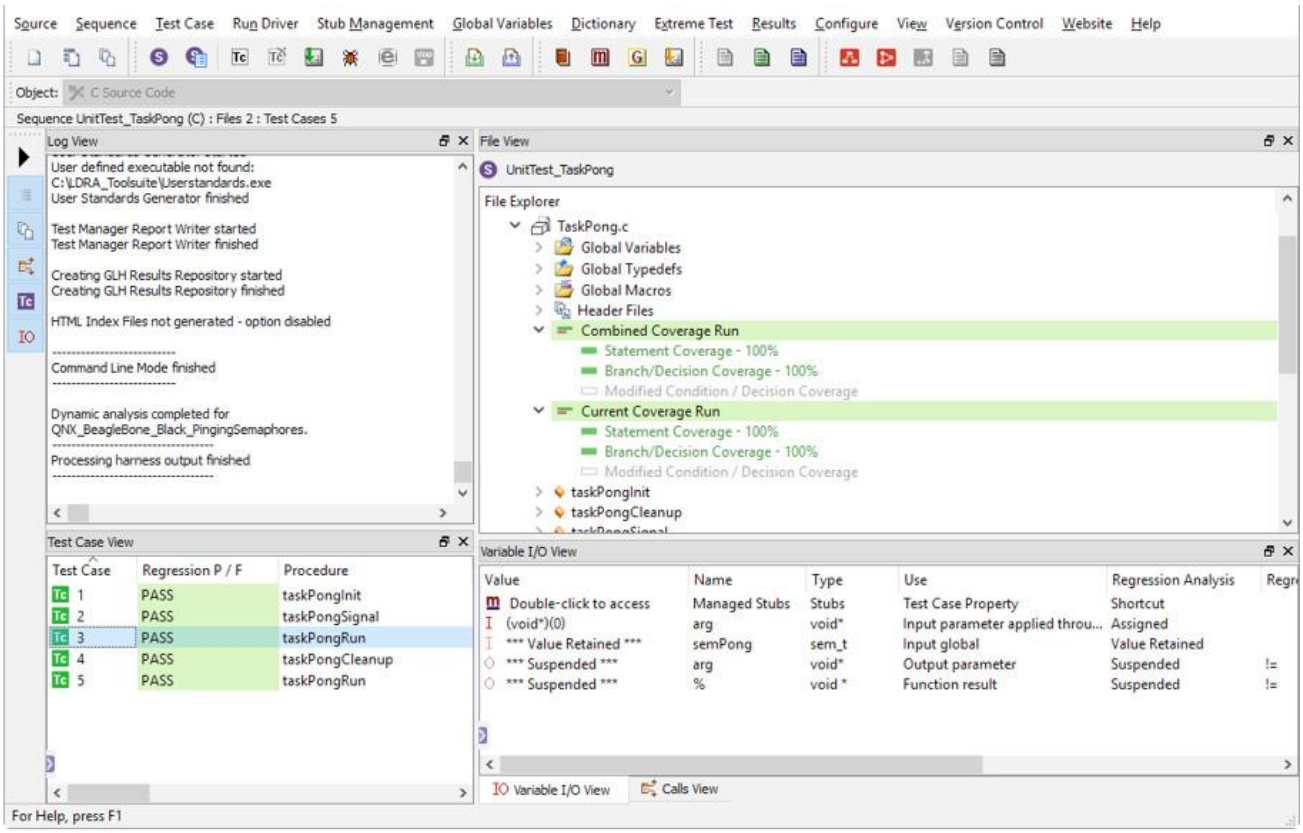

図 14: LDRA tool suite の QNX Momentics サポート

[21] QNX Neutrino http://blackberry.qnx.com/en/products/neutrino-rtos/index

[22] Lynx Software Technologies LynxOS RTOS http://www.lynx.com/products/real-time-operating-systems/lynxos-rtos/

[23] Green Hills INTEGRITY RTOS https://www.ghs.com/products/rtos/integrity.html

[24] Automotive Grade Linux https://www.automotivelinux.org/

AUTOSAR C++コーディングガイドライン

MISRA C++：2008 「クリティカルシステムでの C++言語の使用に関するガイドライン」[25]（つまり MISRA C++）が 2008 年 4 月に公開され、広く認知されました。AUTOSAR コーディング標準を確立する必要が生じた時点で、AUTOSAR グループはこの実証済みの言語サブセットに基づいて作業を行うことが賢明でした。

ただし、MISRA C++は、多くの場合 C++03 として知られている ISO / IEC 14882：2003[26] 標準に基づいていました。MISRA の公開後、図 15 に示すように、C++言語のメジャーアップデートが 2011 年 9 月に公開され（C++11[27]）、2014 年 12 月にさらにマイナーリビジョンが公開されました（C++14[28]）。

したがって、AUTOSAR ドキュメント「クリティカルおよび安全関連システムでの C++14 言語の使用に関するガイドライン」[29]（以降 AUTOSAR C++）は、最近導入された言語機能に対処するための MISRA C++の補遺として設計されました。これらには、ラムダ式、オーバーライド指定子、スマートポインター、変数テンプレート、および可変引数テンプレートが含まれます。初期の言語機能については、AUTOSAR C++は多くの MISRA C++ルールを変更および複製なしで参照します。したがって、AUTOSAR C++を適用する開発者は、MISRA C++ガイドラインも参照する必要があります。

AUTOSAR C++は明らかに自動車分野に主に焦点を当てていますが、他の組み込みアプリケーション分野にも適用できます。

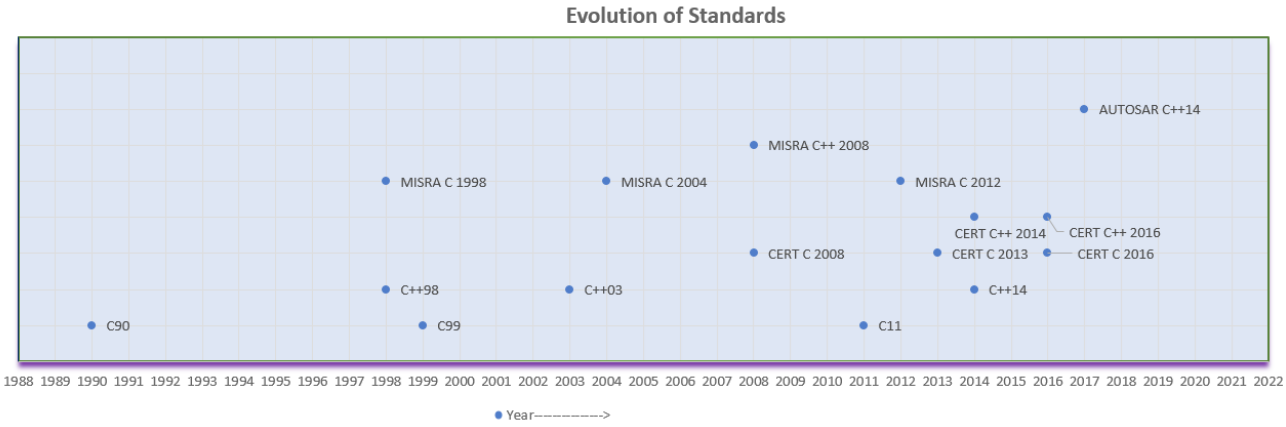

図 15: C、C++コーディング標準のリリース年表

---

[25] “MISRA C++ Guidelines for the Use of the C++ Language in Critical Systems”, ISBN 978-906400-03-3 (paperback), ISBN 978-906400-04-0 (PDF), June 2008. https://www.misra.org.uk/Publications/tabid/57/Default.aspx#label-cpp

[26] ISO/IEC 14882:2003, “The C++ Standard Incorporating Technical Corrigendum 1, International Organization for Standardization, 2003.

[27] ISO/IEC 14882:2011 Standard C++ Foundation, “C++11 Overview”, https://isocpp.org/wiki/faq/cpp11/

[28] ISO/IEC 14882:2014 Standard C++ Foundation, “C++14 Overview”, https://isocpp.org/wiki/faq/cpp14/

[29] AUTOSAR Document ID 838, “Guidelines for the use of the C++14 language in critical and safety-related systems” https://www.autosar.org/ fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_RS_CPP14Guidelines.pdf

AUTOSAR C++ と他の標準

MISRA C++との明確な連携に加えて、AUTOSAR 標準には、JSF AV C++[30]、SEI CERT C++[31]、C++ Core Guidelines[32] など他の C++標準との関係を示すトレーサビリティマトリックスが含まれています。

自動車の機能安全規格 ISO 26262[33] に関連して提供されるこのようなトレーサビリティマトリックスはありません。ただし、言語サブセットを使用するためのドキュメントの要件を満たすための基礎を明確に提供します（図 16）。

Topics		ASIL			
		A	B	C	D
1a	Enforcement of low complexity[a]	++	++	++	++
1b	Use of language subsets[b]	++	++	++	++
1c	Enforcement of strong typing[c]	++	++	++	++
1d	Use of defensive implementation techniques	o	+	++	++
1e	Use of established design principles	+	+	+	++
1f	Use of unambiguous graphical representation	+	++	++	++
1g	Use of style guides	+	++	++	++
1h	Use of naming conventions	++	++	++	++

[a] An appropriate compromise of this topic with other methods in this part of ISO 26262 may be required

[b] The objectives of method 1[b] are
- Exclusion of ambiguously defined language constructs which may be interpreted differently by different modelers, programmers, code generators or compilers.
- Exclusion of language constructs which from experience easily lead to mistakes for example assignments in conditions or identical naming of local and global variables.
- Exclusion of language constructs which could result in unhandled run-time errors.

[c] The objective of method 1[c] is to impose principles of strong typing where these are not inherent in the language

図 16: ISO 26262:「Topics to be covered by modelling and coding guidelines」の Table 1

AUTOSAR C++でのコーディングルールの分類

AUTOSAR C++は MISRA C++の補遺なので、補完的な AUTOSAR C++ルール用に MISRA 形式が保持されます。両方の標準は、義務レベル、適用、および実施割り当てによるルール分類のための共通のアプローチを採用しています。これらの分類は、静的解析によってルールを自動的にチェックできるかどうか、ツールチェーンのどこで適用されるか（たとえば、ツールチェーンにコンパイラ警告が適用される）などの問題を明確に示します。

---

[30] Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program, Document Number 2RDU00001 Rev C, Lockheed Martin Corporation, 2005.

[31] Software Engineering Institute CERT C++ Coding Standard, Software Engineering Institute Division at Carnegie Mellon University, 2016.

[32] Bjarne Stroustrup, Herb Sutter, C++ Core Guidelines, 2017.

[33] International standard ISO 26262 Road vehicles — Functional safety

図 17 は、ルール分類の 3 つのカテゴリと、それらの適用方法の例を示しています。

Category 1: Obligation Level			
Required		**Advisory**	
Rules must be followed to comply If not followed, then formal deviation is required		If Rules are not followed, then formal deviation is not required	
Category 2: Enforcement by Static Analysis			
Automated	**Partially Automated**	**Non-automated**	
Rules can be checked by Static Analyser	Rules can be partially checked by Static Analyser, but manual inspection is also required	Static Analyser cannot provide any help here or can provide limited help. Rules need to be tested by other means also, for example, manual review of the code.	
Example Rules			
Rule A10-1-1 Class shall not be derived from more than one base class which is not an interface class.	*Rule A15-5-2 Program shall not be abruptly terminated*	*Rule A1-1-2 A warning level of the compilation process shall be set in compliance with project policies*	
Category 3: Allocated Target			
Implementation	**Verification**	**Toolchain**	**Infrastructure**
These rules are implemented in code, software design	Rules are applied to the verification activity like manual code review, testing	Some rules can be applied to the toolchain like compiler, linker etc.	Rules can be applied to OS running on Target system.
Example Rules			
Rule M0-1-9 There shall be no dead code.	Rule A15-0-6 An analysis shall be performed to analyse the failure modes of exception handling.	Rule A1-1-2 A warning level of the compilation process shall be set in compliance with project policies.	Rule A0-4-1 The floating-point implementation shall comply with IEEE 754 standard.

図 17: AUTOSAR C++と MISRA C++に共通のルール分類

LDRA tool suite と AUTOSAR C++14

LDRA は、言語サブセットをサポートするという長年の実績と経験から、AUTOSAR C++のベースとなっている標準をまとめる MISRA C および C++委員会で重要な役割を果たしています（2019 年現在、各委員長は LDRA の社員）。したがって LDRA 静的解析ツールが AUTOSAR C++をサポートすることは、これまで MISRA C：2016 with Amendment 1、MISRA C++：2008、CERT C、および CERT C++などをサポートしてきたことの必然的な延長です（図 18）。この機能は、AUTOSAR C++が選択された言語のサブセットである ISO 26262 への準拠を検討している開発者にとって非常に重要です。

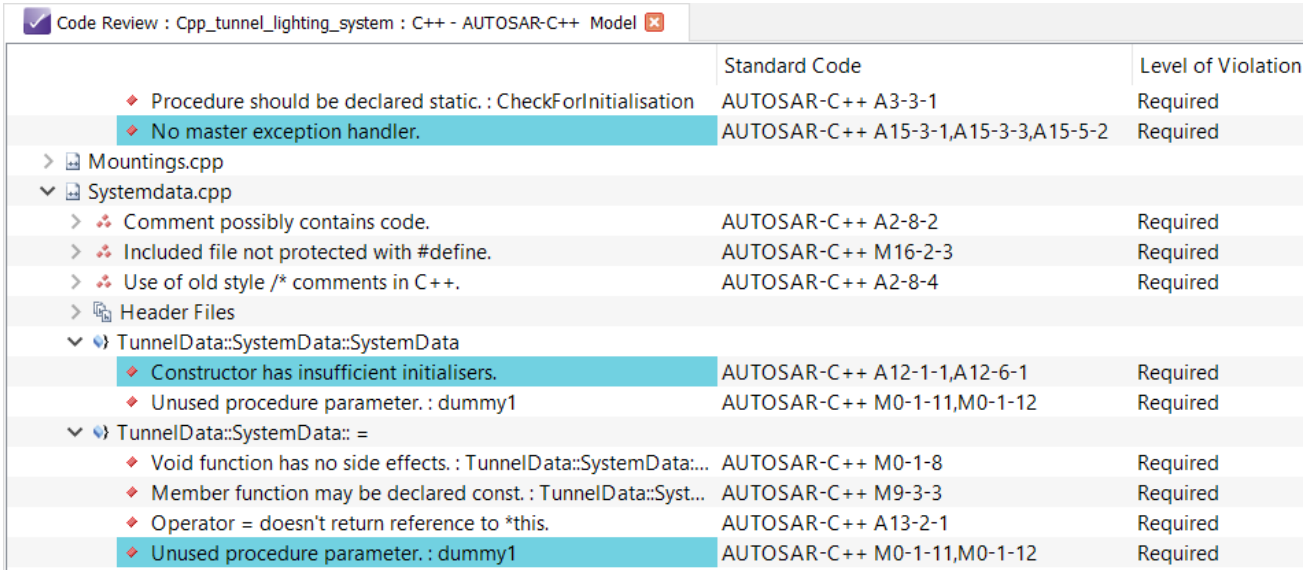

図 18: LDRA tool suite で示される AUTOSAR C++違反

AUTOSAR C++には、MISRA C++と等価なものを単に引用するだけのルールを含む 337 のルールがあり、そのうち 308 は静的にチェック可能です。それらのうちの 78%が LDRA ツールに実装されています。

Classification	Enhanced Enforcement	Fully Implemented	Partially Implemented	Not yet Implemented	Not statically Checkable	Total
Required	22	168	39	56	23	308
Advisory	1	8	1	11	2	23
Document	0	1	1	0	4	6
Total	23	177	41	67	29	337

図 19: LDRA tool suite における AUTOSAR C++コーディング準拠

例　AUTOSAR C++ルールと違反

ルール A10-1-1：「菱形継承問題」の対処

「クラスは、インターフェイスクラスではない複数の基底クラスから派生してはならない」

このルールは、菱形継承問題[34]として一般に知られている、多重継承に関連する C++の問題に対処します。これは、複数の親クラスがその機能を実装している場合に、特定の機能がどの親クラスから継承されるかがあいまいになる状況を説明しています。

図 20 に、この規則に違反するコードの例を示し、自動的に識別する方法を示します。

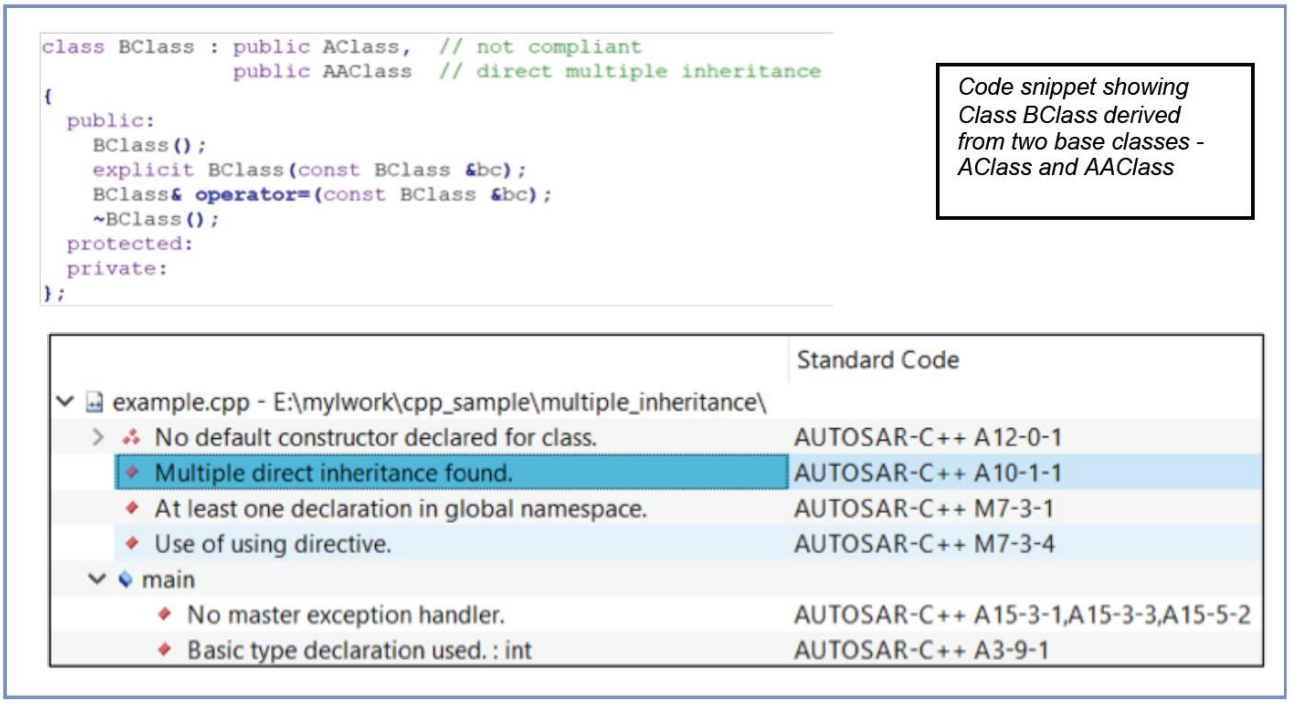

図 20: LDRA tool suite が検出した AUTOSAR ルール A10-1-1 違反

ルール A27-0-1：AUTOSAR C++とセキュリティ

「独立したコンポーネントからの入力は、検証されなければならない」

AUTOSAR C++ではセキュリティは明示的に言及されていませんが、この標準内にはセキュリティに該当する多くのルールがあり、このルールはその一例です。外部入力は攻撃者のエントリポイントであるため、外部入力の検証は重要な防御戦略です。 チェックしない場合、不正なデータ入力によりメモリが破損し、任意のコードが実行され、システムが制御される可能性があります。

図 21 は、このルールに違反するコードの例を示し、自動的に識別する方法を示しています。

[34] https://medium.freecodecamp.org/multiple-inheritance-in-c-and-the-diamond-problem-7c12a9ddbbec

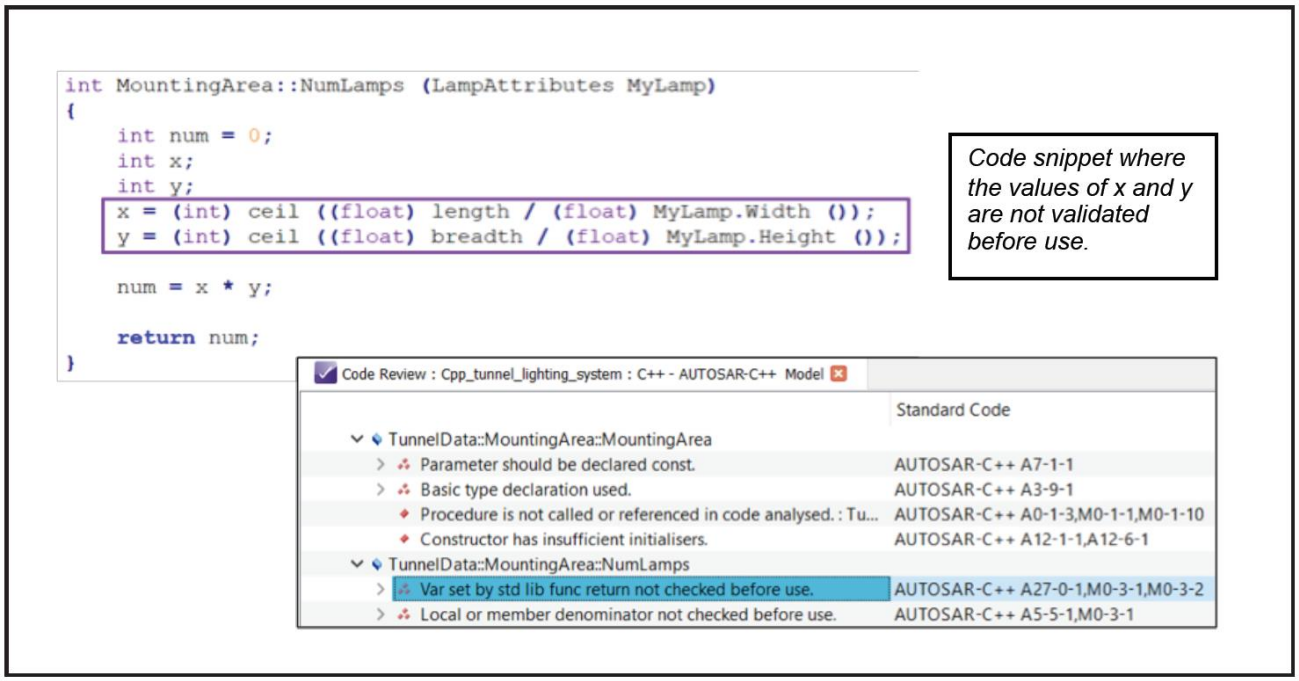

図 21: LDRA tool suite が検出した AUTOSAR ルール A27-0-1 違反

ルール M8-5-2：波括弧初期化

「波括弧は、配列と構造体のゼロ以外の初期化において、構造を示して一致させるために使用する」

波括弧（またはリスト）初期化は C++11 で導入されたので、C++14 の機能でもあります。これは、以前のバージョンで提唱されていたものよりもエラーが発生しにくい初期化メカニズムです。

初期化に対する従来のアプローチでは、図 22 に示すように「=」演算子を使用します。

```
int varint =2; // 2 will be stored in varint variable, no error
int myvar =2.7; // result in loss of information, as 2.7 will become 2 when stored in variable myvar
```

図 22: 従来の初期化

このアプローチは誤解を招く可能性があります。この例では、開発者は 2.7 を myvar に保存することを明確に意図しており、コードは問題なくコンパイルされます。ただし、myvar は整数型変数であるため、実際には 2 の値が割り当てられます。

図 23 は、波括弧初期化での同等の使用法を示しています。この場合、myvar を 2.7 に初期化しようとすると、コンパイルエラーが発生します。

```
int varint {2}; // recommended method
int myvar {2.7}; // compilation error as narrowing is happing in this line
double mydouble {2.7}; // correct method
```

図 23: 波括弧（またはリスト）初期化

AUTOSAR C++ルール M8-5-2 は、波括弧初期化の使用を推奨しており、配列または構造内のすべての要素の初期化は行わないことに起因する一般的な問題に対処しています。図 24 は、整数配列と文字配列に関連する 2 つの例を示しています。

```
INT_32 my_array[3] = { 1, 2 };          /* Not Compliant */
CHAR char_10[10] = "Hello";             /* Not Compliant */
```

図 24: 部分初期化された配列 – ルール M8-5-2 違反

ルール A4-10-1：*nullptr* の使用

「ナルポインタ定数には nullptr リテラルだけを使用する」

C++11 より前は、ナルポインタと整数値 0 の両方を表すために、従来から NULL が使用されていました。図 25 は、int と char *の両方を引数とするためにオーバーロードされた関数（f）を示しています。NULL でナルポインタを表したつもり（左）で呼び出しても、int にオーバーロードされたバージョンを呼び出します。代わりに nullptr を使用する（右）と、目的の動作が得られます。

```
void f(int);
void f(char*);

void g()
{
    f(NULL); //calls f(int)
}
```

```
void f(int);
void f(char*);

void g()
{
    f(nullptr); //calls f(char*)
}
```

図 25: ナルポインタ定数として NULL を使用することで生じる誤解

ルール A7-2-3：スコープ付き列挙の使用

「列挙は、スコープ付き enum クラスとして宣言する」

enum クラスは C++ 11 で導入されました。プレーン enum で宣言される場合とは異なり、enum クラスとして宣言される列挙子名は、その enum 宣言に対してローカルであり、値が暗黙的に他の型（別の enum や int など）に変換されません。

これはより制限的ですが、予期しない結果が発生する可能性がはるかに低いことを意味します。

AUTOSAR C++と ISO 26262

ISO 26262：2011 は、ASIL（Automotive Safety Integrity Levels）として知られる複数の安全性レベルを指定しています。ASIL に比例した不当な残留リスクを回避するために、開発プロセスのチェックと安全対策が指定されています。ASIL の範囲は A から D です。ASIL D は最も危険で要求レベルが高いため、安全性が重要な ASIL D システム（自動ブレーキなど）の生成に伴うオーバーヘッドは、高い安全性レベルが求められない ASIL A システム（車内エンターテイメントシステムなど）の生成に必要なオーバーヘッドよりも大きくなります。

ASIL は、システムまたはシステムコンポーネント全体のプロパティとしてではなく、個々の安全機能のプロパティとして割り当てられます。安全関連システムの安全機能に割り当てられた ASIL は、関連する危険なイベントのプロパティによって決定され、次の 3 つの属性の影響を受けます。

- 状況の頻度（すなわち「暴露」）
- 考えられる損傷の影響（すなわち「重大度」）
- 制御性

ISO 26262：2011-6 Section 5 は、図 26 に示すように、コーディングガイドライン（言語サブセットの使用を含む）と低複雑度の適用の両方を求めています。

ISO:26262-6 Section 5	Coding Guidelines	Applies to:
1a	Enforcement of Low Complexity	All ASILs
1b	Use of Language Subsets	All ASILs
1c	Enforcement of Strong Typing	All ASILs
1g	Use of Style Guides	ASILs B, C and D
1h	Use of Naming Conventions	All ASILs
ISO:26262-6 Section 7	**Software Design**	**Applies to:**
1a	Enforcement of Low Complexity	All ASILs
1b	Use of Language Subsets	All ASILs

図 26: ISO 26262-6:2011 におけるコーディングガイドラインとソフトウェアコンポーネントのサイズ

コード品質の管理を支援するために利用可能な多くのメトリックがありますが、多くの組織はプロジェクトの中でこれらの一部のみを採用しているに過ぎません（図 27）。

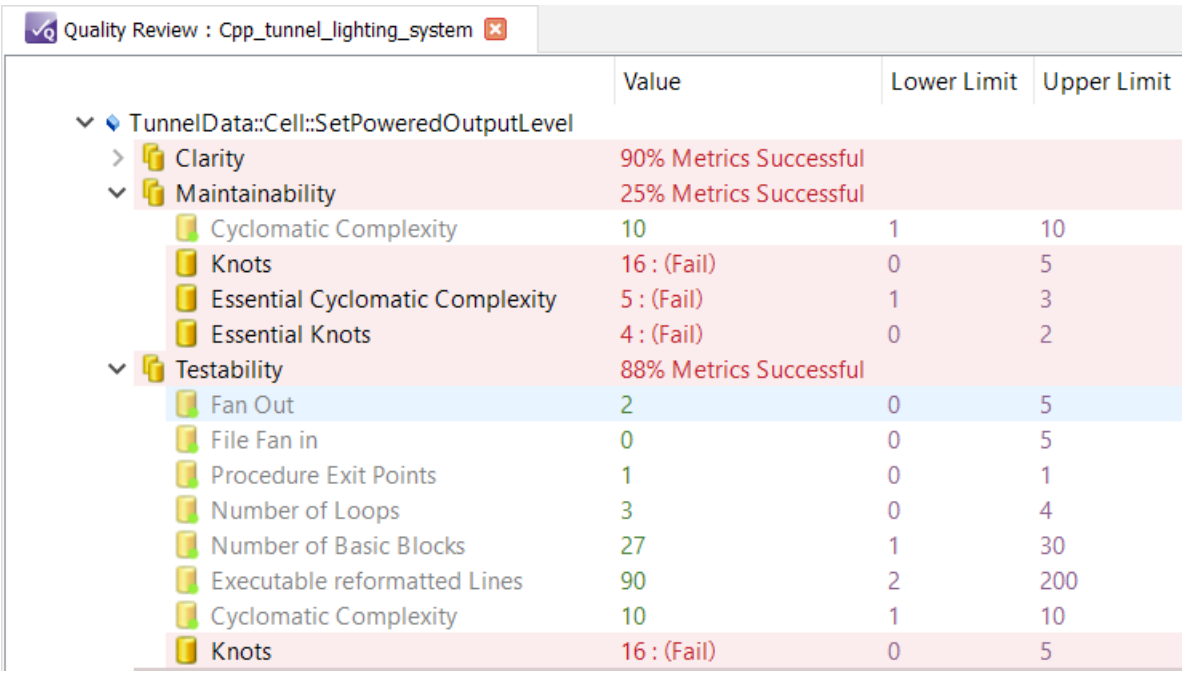

図 27: LDRA tool suite は 45 を超える品質メトリックを測定し、それぞれ上下限値を設定できます。この LDRA tool suite の品質調査レポートではそのようなメトリックのいくつかを示します。

複雑さは、コードの循環的複雑さ、基本ブロックの平均長、ループの入れ子の深さなどのメトリックを使用して定量化できます。これらは、到達不能な行、到達不能な分岐、LCSAJ[35]（パス密度のメトリック）、およびプロシージャ出口点などの数を含めた、コード品質のさまざまな側面を評価するために設計された他のコードメトリックによって補完されます。

AUTOSAR C++には、コーディングメトリックの使用に対処するいくつかのルールがあります。

[35] LCSAJ Linear Code Sequence and Jump Analysis, devised by Professor Michael Hennell (Founder of LDRA) http://www.professionalqa.com/lcsaj-testing

例　コーディングメトリックに関係する AUTOSAR C++ルールと違反

ルール A1-4-2：定義済み境界の使用

「すべてのコードは、コードメトリックの定義された境界に準拠しなければならない」

このルールの目的は、コードの複雑さを制御することです。これは、ISO 26262 に関連して前述したように、複雑なコードは保守とテストが難しいためです（図 28）。

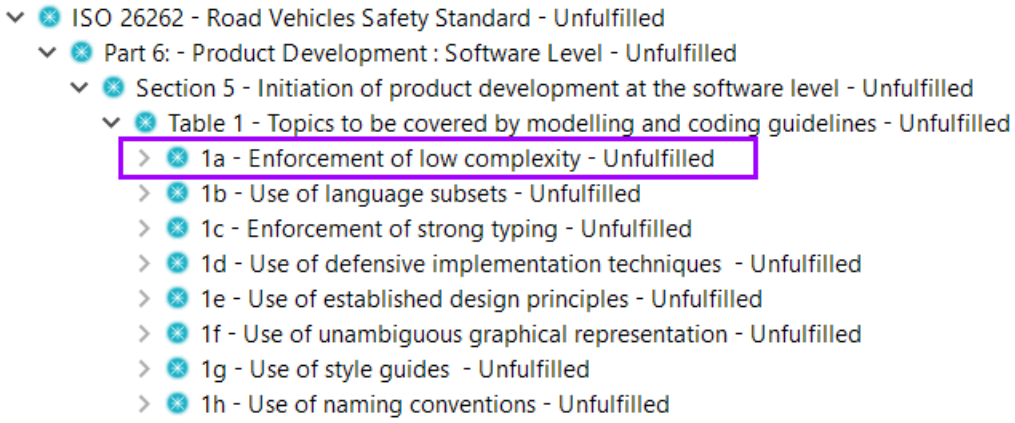

図 28: LDRA tool suite の TBmanager では、ここに示されるように「低複雑度の適用」を含んだ目的の達成を自動でモニタします。

より完全なソリューション

双方向のトレーサビリティ（ISO 26262-4:2011 と ISO 26262-6:2011）

静的および動的解析が重要であることは間違いないですが、ISO 26262 への準拠は AUTOSAR の有無にかかわらず、はるかに重要です。ISO26262：2011 全体で双方向のトレーサビリティは原則であり、各開発段階では一つ前の段階を正確に反映する必要があります。理論的には、V モデルの正確なシーケンスが守られていれば要件は変更されず、テストで問題が発生することはありません。しかし、現実はそう簡単ではありません。

もし統合テストの失敗に応じてコードが変更されると、どうなるでしょう？ その原因は、要件との一致性の欠如や、コーディングエラーの有無などでしょう。また、他のどのソフトウェアユニットが変更されたコードに依存していたでしょうか？このようなシナリオでは、ソフトウェア開発の成果物間のトレーサビリティが低下する状況に陥る可能性があります。トレーサビリティを手動で維持することは可能ですが、その自動化は非常に役立つことを強調します。

ソフトウェアユニットの設計要素は、ソフトウェアの安全要件とソフトウェアアーキテクチャの両方に双方向で追跡可能である必要があります。次に、ソフトウェアユニットを仕様どおりに実装し、設計仕様に追跡可能にする必要があります。

要件トレーサビリティの自動化支援ツールを使用して、さまざまなスコープの要件とテストケースを確立することにより、テストカバレッジを評価できます（図 29）。そこで失敗したテストケースは、その影響を評価して対処することができ、要件変更の影響や要件カバレッジのギャップも同様に評価することができます。また、トレーサビリティマトリックスなどの成果物を自動的に生成して、ISO 26262：2011 への準拠へのエビデンスを提示することもできます。

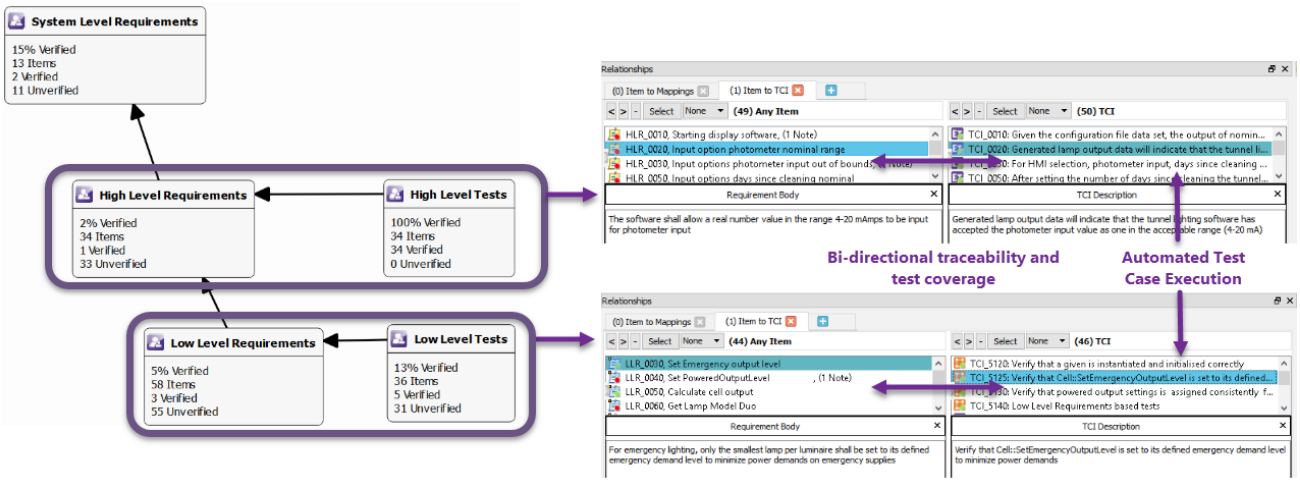

図 29: 要求ベーステストの実施。テストケースは要件にひもづけられ、LDRA tool suite 内で実行されます。

通常、実践的な構造カバレッジ解析は、解析のためにインストルメントされたコードの機能テストの実行から開始されます。そしてさらなる解析を必要とするコードの未実行部分が明らかになり、最終的に、テストケースの追加や修正、要件の変更、および/またはデッドコードの削除が行われます。そしてレビュー、修正、解析の繰り返し作業によって、設計仕様が保証されます。

従来の分離されたシステムの開発であれば、このやり方でも十分です。しかしながら接続されるシステムの場合、現場で特定される脆弱性への対応力が求められます。新たに発見される脆弱性によって要件の変更、あるいは新たな要件が求められ、すぐに対応しなくてはなりません。開発エンジニアがシステム自体に長い間関わっていなかったとしてもです。このような状況で、必要なものを見出して、影響を受けるコードのみを自動的にテストできることは非常に重要です。

接続されるシステムでは、製品の新発売時や生産が終了するときでさえ、開発プロセスの終了の概念が変更されます。市場で新しい脆弱性が発見されるたびに、迅速な対応は命や評判に関わるというプレッシャーの中、要件の変更が行われます。このような状況下では、自動化された要件トレーサビリティが新たな視点で重要なソリューションとなります。

結論

標準規格のサイズや範囲、詳細はさまざまであり、AUTOSAR を単なる別の標準と考えるのは簡単です。実際は、AUTOSAR は非常に巨大であるので、一担当者がすべての詳細な知識を持つことはできません。

Adaptive Platform は Classic Platform に取って代わるものではありません。それは AUTOSAR の機能を拡張し、結果的に AUTOSAR の様々な要素を全体として十分に把握することが難しくなりました。

静的解析、動的解析、単体テスト、要件トレーサビリティ、スタンダードで要求されるオブジェクティブの管理など、安全性が重視される業界で実績のあるツールは非常に有益です。しかしながら、開発サイクルに共通点のないツールのコレクションを適用することは、すでに困難な開発状況下に不必要な複雑さを追加する可能性があります。そのため最も重要なことは、LDRA tool suite のようなツールチェーンで 1 つのシームレスな全体に統合することです。

引用資料

An introduction to AUTOSAR
https://retis.sssup.it/sites/default/files/lesson19_autosar.pdf

ANSYS Esterel Scade Suite
http://www.esterel-technologies.com/products/scade-suite/

Automotive Grade Linux
https://www.automotivelinux.org/

Automotive Industries – March 2016 - New AUTOSAR adaptive platform on its way
http://www.ai-online.com/Adv/Previous/show_issue.php?id=6881#sthash.Cvo3ABXn.0xLSMTMG.dpbs

AUTOSAR – A First Glance
https://www.youtube.com/watch?v=F27jtKkxbAo

AUTOSAR Adaptive Platform
https://www.autosar.org/standards/adaptive-platform/

AUTOSAR Classic Platform
https://www.autosar.org/standards/classic-platform/

AUTOSAR Document ID 838, “Guidelines for the use of the C++14 language in critical and safety-related systems”
https://www.autosar.org/fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_RS_CPP14Guidelines.pdf

AUTOSAR Explanation of ara:comm API
https://www.autosar.org/fileadmin/user_upload/standards/adaptive/17-03/AUTOSAR_EXP_ARAComAPI.pdf

AUTOSAR Foundation
https://www.autosar.org/standards/adaptive-platform/

AUTOSAR Software Component Template
https://www.autosar.org/fileadmin/user_upload/standards/classic/41/AUTOSAR_TPS_SoftwareComponentTemplate.pdf

Bjarne Stroustrup, Herb Sutter, C++ Core Guidelines, 2017.

Embedded.com - Electronic Blogs Accelerating development with model-based design
https://www.embedded.com/electronics-blogs/say-what-/4440209/Accelerating-development-with-model-based-design

freeCodeCamp - Onur Tuna – Multiple Inheritance in C++ and the Diamond Problem
https://medium.freecodecamp.org/multiple-inheritance-in-c-and-the-diamond-problem-7c12a9ddbbec

Green Hills INTEGRITY RTOS
https://www.ghs.com/products/rtos/integrity.html

IBM Rational Rhapsody family
http://www-03.ibm.com/software/products/en/ratirhapfami/

ISO 26262:2011 Road vehicles — Functional safety

ISO 26262:2011 Road vehicles — Functional safety — Part 4: Product development at the system level

ISO 26262:2011 Road vehicles — Functional safety — Part 5: Product development at the hardware level

ISO 26262:2011 Road vehicles — Functional safety — Part 6: Product development at the software level

ISO/IEC 14882:2003, The C++ Standard Incorporating Technical Corrigendum 1, International Organization for Standardization, 2003.

ISO/IEC 14882:2011 Standard C++ Foundation, “C++11 Overview”
https://isocpp.org/wiki/faq/cpp11/

ISO/IEC 14882:2014 Standard C++ Foundation, “C++14 Overview”
https://isocpp.org/wiki/faq/cpp14/

IEC 61508:2010 Functional safety of electrical/electronic/programmable electronic safety-related systems

Joint Strike Fighter Air Vehicle C++ Coding Standards for the System Development and Demonstration Program, Document Number 2RDU00001 Rev C, Lockheed Martin Corporation, 2005.

LCSAJ Linear Code Sequence and Jump Analysis
http://www.professionalqa.com/lcsaj-testing

Lynx Software Technologies LynxOS RTOS
http://www.lynx.com/products/real-time-operating-systems/lynxos-rtos/

MathWorks Simulink
https://www.mathworks.com/products/simulink.html

MISRA C++ Guidelines for the Use of the C++ Language in Critical Systems, ISBN 978-906400-03-3 (paperback), ISBN 978-906400-04-0 (PDF), June 2008.
https://www.misra.org.uk/Publications/tabid/57/Default.aspx#label-cpp

MISRA – The Motor Industry Software Reliability Association
https://www.misra.org.uk/

QNX Neutrino
http://blackberry.qnx.com/en/products/neutrino-rtos/index

Software Engineering Institute CERT C++ Coding Standard, Software Engineering Institute Division at Carnegie Mellon University, 2016.