ISO/SAE 21434 規格に対応したソフトウェア開発モデル

EDN edn.com/software-development-model-for-the-iso-sae-21434-standard/

Mark Pitchford

2022 年 8 月 10 日

Mark Pitchford has over 30 years' experience in software development for engineering applications. He has worked on many significant industrial and commercial projects in development and management, both in the UK and internationally. Since 2001, he has worked with development teams looking to achieve compliant software development in safety- and security-critical environments, working with standards such as DO-178, IEC 61508, ISO 26262, IIRA, and RAMI 4.0. Mark earned his Bachelor of Science degree at Trent University, Nottingham, and he has been a Chartered Engineer for over 20 years. He now works as Technical Specialist with LDRA Software Technology.

機能安全の世界では、システムはひとたび開発されると、それが稼働している限り保護され続けます。一方、セキュリティの世界では、製品のリリース後も、絶えず進化する攻撃手法からシステムを守り続けることが、ソフトウェアに求められています。車載コネクティビティの需要が爆発的に増加する中、エンターテイメントなど重要でないシステムは、重要なステアリング、ブレーキ、制御システムと同じ通信インフラを共有するようになり、自動車システムの開発者は両方の世界の規格を満たすように迫られています。ベストプラクティスの指針として ISO/SAE 21434 では、ISO 26262「路上走行車－機能安全」で省略されている箇所を取り上げています。

ISO/SAE 21434 は ISO 26262 を定期的に参照しており、この 2 つの規格は製品ライフサイクルの各段階での統合に適していて、使い慣れた利用可能なツールを使用する同じテストチームを配備して両方の役割を果たすようにできる程です。たとえば、ハザード分析、安全リスク評価、脅威分析、そしてセキュリティリスク評価を単一の統合されたテンプレートと手法で同時に実施する技法を開発できます。

この 2 部構成の記事の第一部では、ISO/SAE 21434 にいたった技術の進化を説明し、ソフトウェア開発者にとっての意味について述べました。第二部（本記事）では、従来の開発での V モデルのステップを順に説明し、この規格で概説されている原則が各段階でどのように適用できるかを説明します。

ISO/SAE 21434 に準拠したソフトウェア開発

図 1 は、ソフトウェア開発に最も影響を与える ISO/SAE 21434 のセクション間の関係を示す修正 V モデルです。以下のセクションでは、規格で概説されている原則がどのように適用されるかについて説明します。

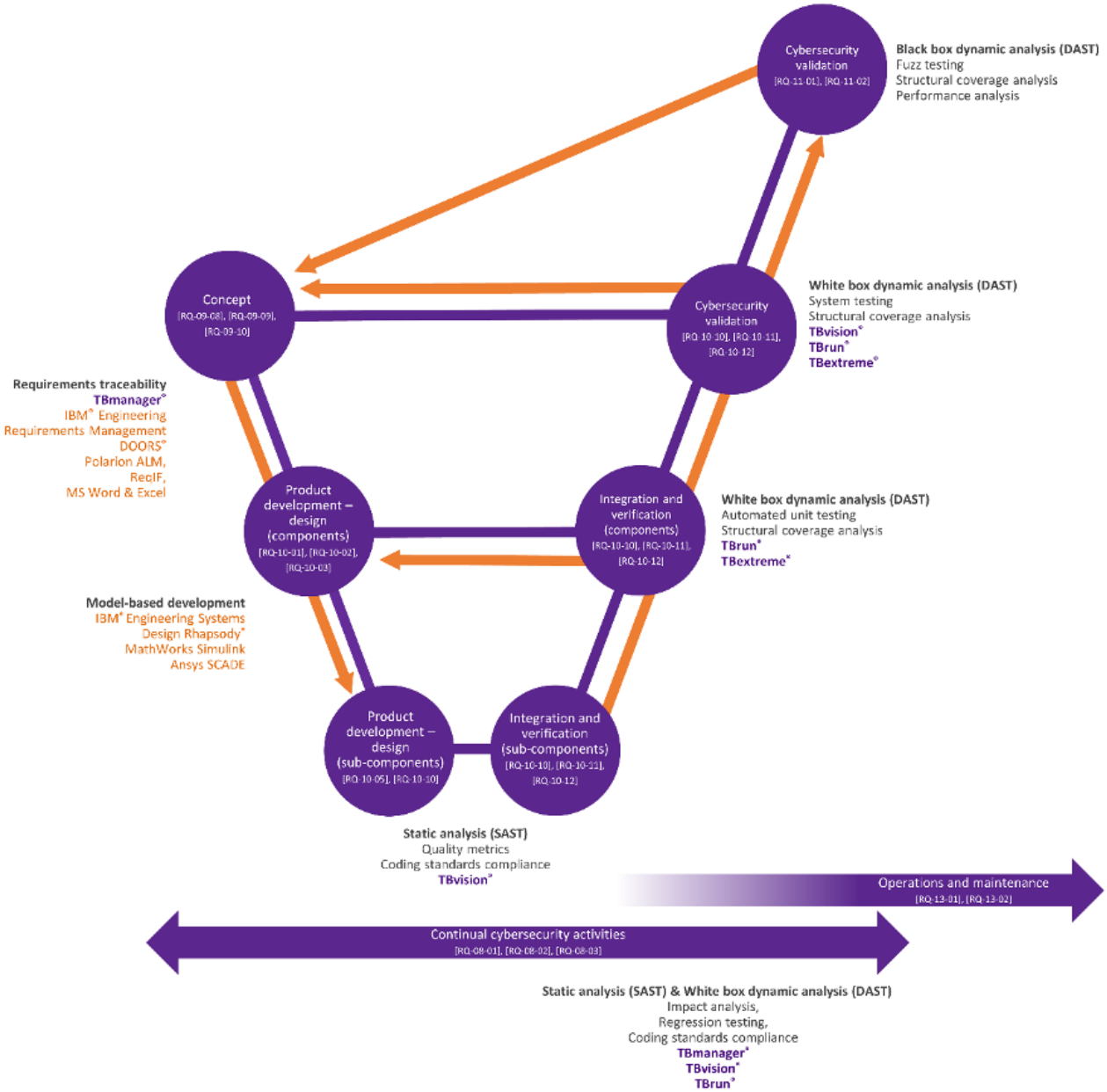

図 1　ISO/SAE 21434 のライフサイクルを修正 V モデルで表現したもの　出典：LDRA

ISO/SAE 21434 §8.5 脆弱性解析

脆弱性解析は、サイバーセキュリティクリティカルなシステムの開発ライフサイクルと、機能安全のみに関係するシステムとを区別する重要な要因です。この規格では関連する原則を一般化しているため、接続された組込みソフトウェアシステムにどのように適用するかを検討するのに役立ちます。

その 1 つが「信頼境界」という概念で、これはプログラム上に引かれた線と考えることができます。この線の片側ではデータは信頼されません。その反対側では、データは信頼できるものと見なされます。

ソフトウェア脆弱性解析（SVA）の最初のステップは、アプリケーションを分解し、データと制御の入口/出口点を分析することです。データが信頼境界を越える場合は適切な制御が定義され、「脅威モデル」として文書化されます。このモデルは、システム全体を通るさまざまなパス

を示す特別のデータフロー図であり、権限の境界を強調するものです。この分析とそれに関連するドキュメントは、設計チームがサイバーセキュリティの検証時に特に重視する箇所を決定するのに役立ちます。

ソフトウェア脆弱性解析の第 2 ステップでは、STRIDE や DREAD などの脅威分類を用いて、システムの詳細項目に基づいて脅威を特定します。

共通脆弱性評価システム CVSS（Common Vulnerability Scoring System)と CWSS（Common Weakness Scoring System）は、それぞれ共通脆弱性識別子 CVE（Common Vulnerabilities and Exposures）と共通脆弱性タイプ一覧 CWE（Common Weakness Enumeration）に関連付けられており、モジュールやアプリケーション、ソースコードのレベルでソフトウェアの脆弱性と弱点を分類するのに役立ちます。

これらの脅威の特定と分類は、ISO/SAE 21434 で説明されている脅威エージェントリスクアクセス TARA（Threat Agent Risk Assessment）の原則と自然に調和し、適切なセキュリティ対策が確実に適用されるように支援するものです。

ISO/SAE 21434 §9 コンセプト

コンセプトフェーズでは、車両レベルの機能を検討します。機能は、サイバーセキュリティの目標が定義された「アイテム」に実装されます。サイバーセキュリティの目標は、後続の設計と実装フェーズに反映されなければならない高レベル要件です。このトレーサビリティを従来の方法で実証することは、特にテストが失敗した場合や要件が変更されたた場合に、プロジェクト管理での頭痛の種となることがあります。

要件のトレーサビリティと ISO/SAE 21434

サイバーセキュリティ要件のトレーサビリティは、ISO 26262 における機能安全要件のトレーサビリティと同様、ISO/SAE 21434 の主要な目標です。ISO/SAE 21434 の要求事項 [RQ-09-08]、[RQ-09-09]、[RQ-09-10] は、サイバーセキュリティ仕様からサイバーセキュリティ要件を導き出すこと、およびトレーサビリティの原則が開発ライフサイクルを通して確立されていることについて説明しています。

プロジェクトの要件と規格の目的、両方のトレーサビリティを追跡することで、特に変更が発生した場合にプロジェクト管理での悪夢を回避できます。この問題は、ISO 26262、AUTOSAR、ASPICE といった追加の規格が同時に適用される場合、より深刻になります。自動化されたトレーサビリティにより、持続的に最新の要件トレーサビリティマトリクス（RTM）にアクセスできます。

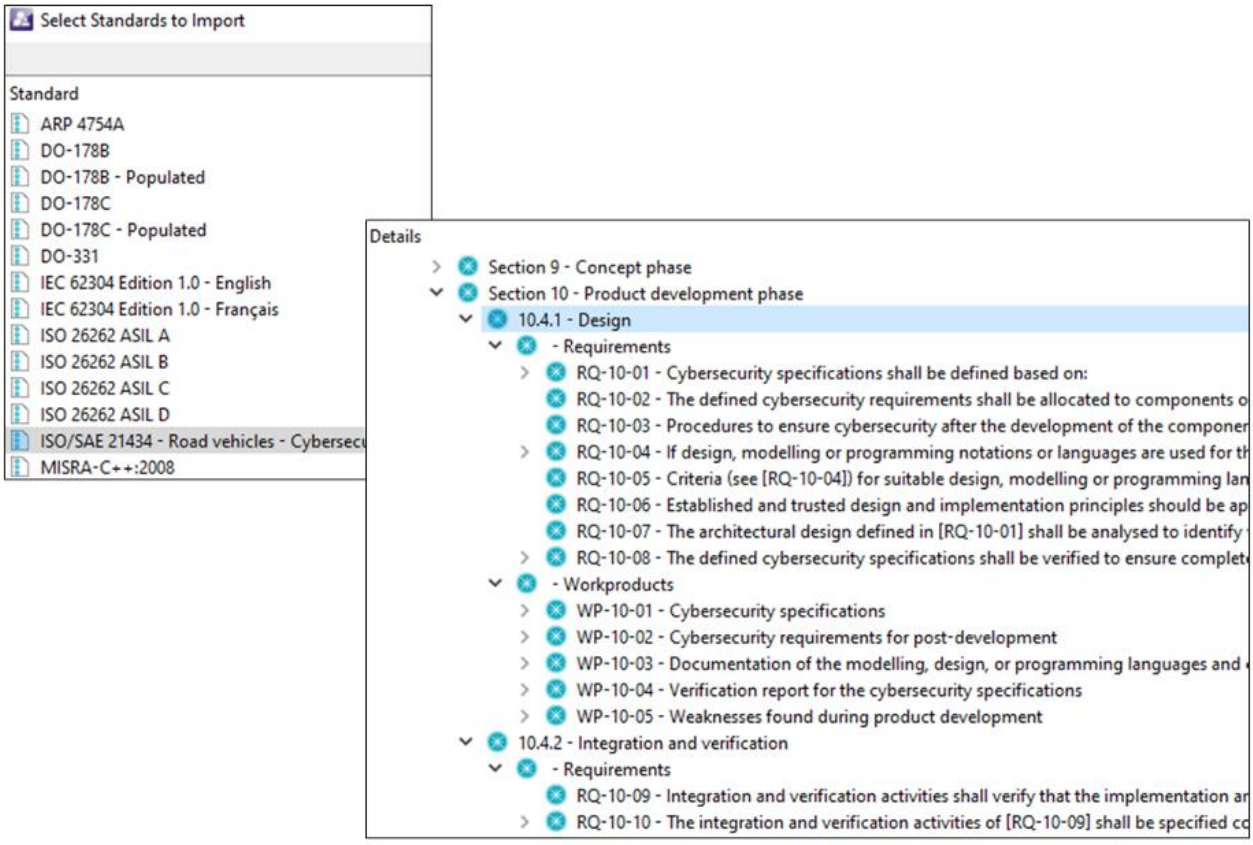

図 2 LDRA ツールスイートの TBmanager コンポーネントは、1 つ以上の規格から照合された目的だけでなく、プロジェクト要件へのトレーサビリティも自動化します

ISO/SAE 21434 §10.4.1 製品開発:設計

コーディング規約

ISO/SAE 21434 の要件 [RQ-10-05]では、適切な言語サブセット（コーディング規約としてよく知られます）を使用することを提案しています。C/C++、MISRA C/C++、CERTC/C++など、多くのコーディング規約が利用できます。IEC/ISO 21434 では、適切な言語サブセットの例として MISRA と SEI CERT を挙げています。ISO 26262 でも機能安全の観点から MISRA 言語サブセットを推奨しています。

これらガイドラインへの準拠を強制するための従来のアプローチは、ピアコードレビューを行うことです。これはチームメンバーの学習に役立ちますが、面倒なチェックを自動化する方がはるかに効率的で、エラーも発生しにくくなります（図 3）。

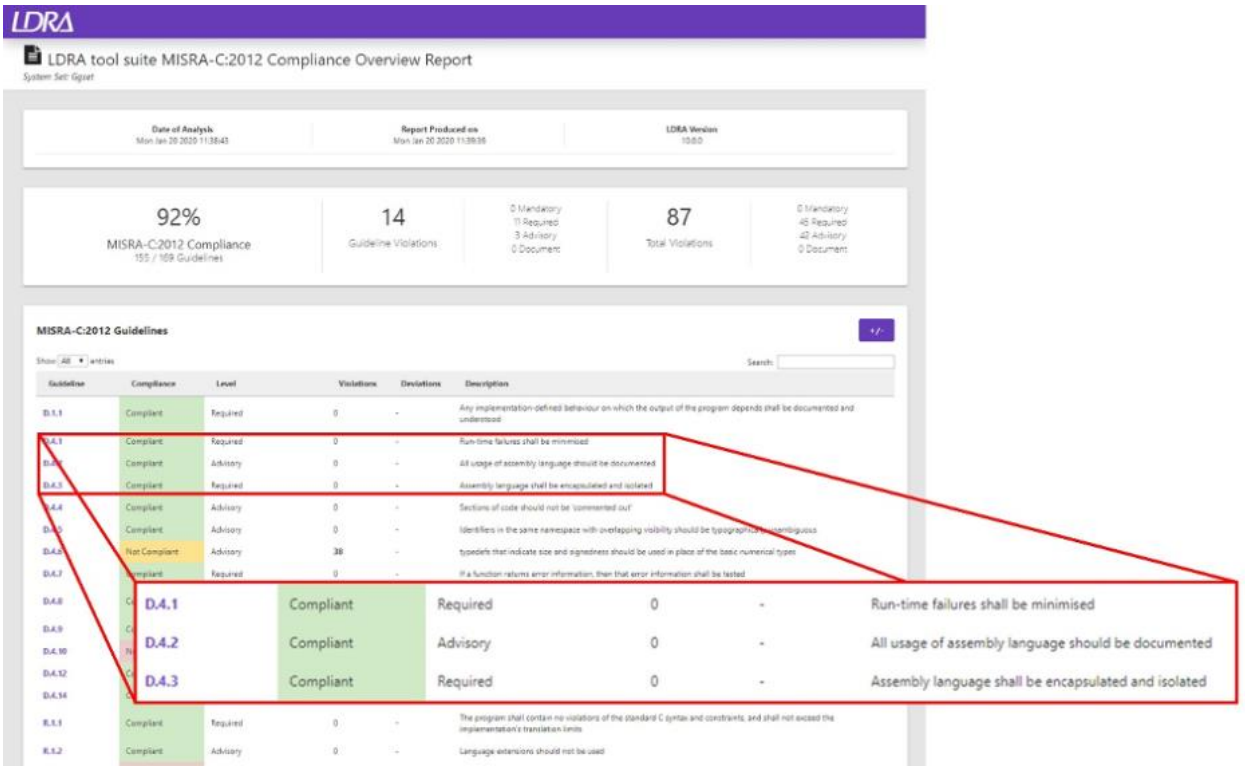

図 3　LDRA ツール スイートの TBvision コンポーネントを使用して、コーディング規約やスタイル ガイド、言語サブセットで指定されたコーディング規則への準拠を検証できます

ISO/SAE 21434 は、特定のコーディング規約の使用を強制するものではなく、開発者はプロジェクトに特化したコーディング規約を使用することも、確立した規約を選択して特定のアプリケーションに適合させることも可能です。このような状況でツールが有用であるためには、こういった調整に対応できなければなりません（図 4）

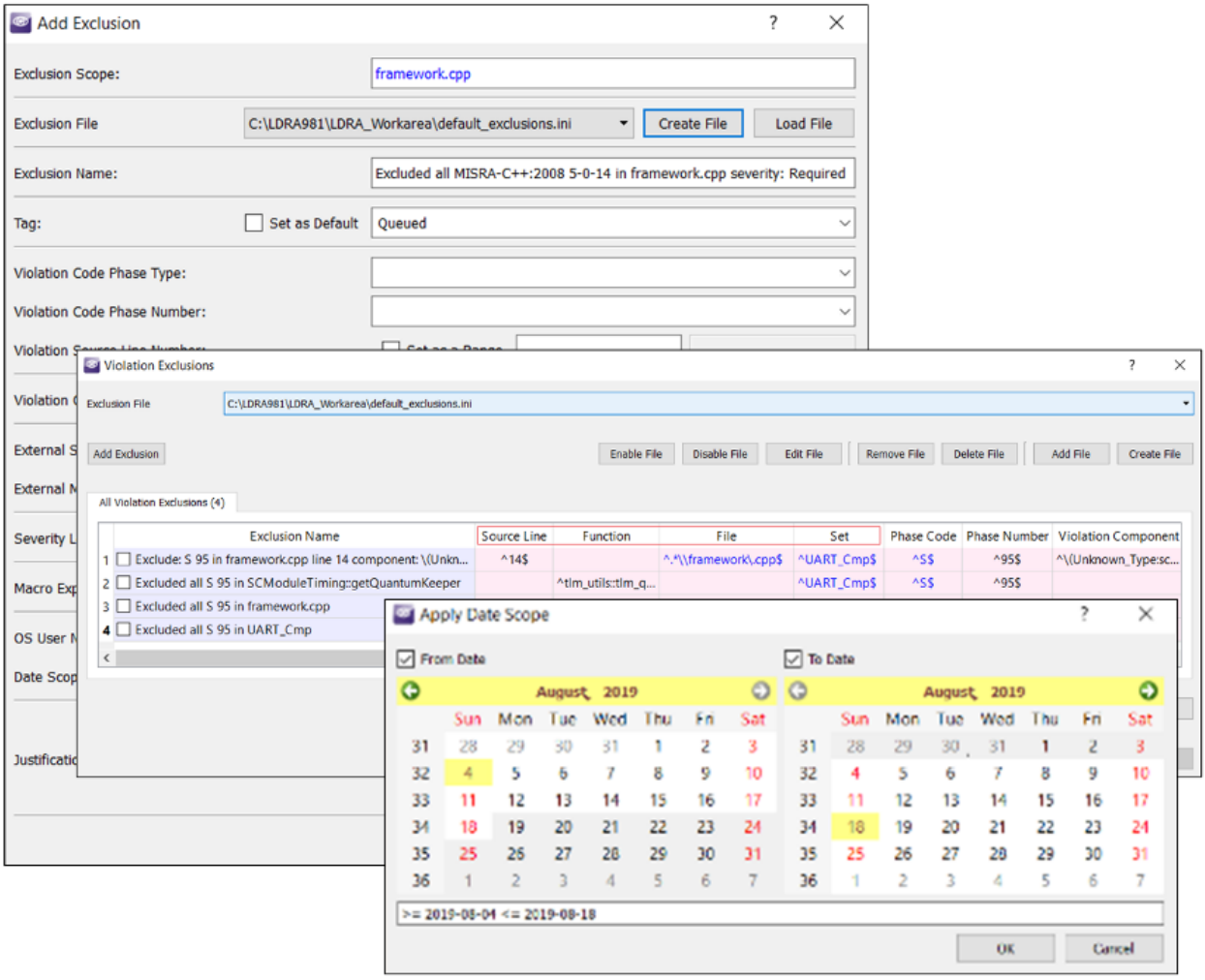

図 4　TBexclude モジュールには多階層の違反除外機能があります

ISO/SAE 21434 §10.4.2 製品開発:統合と検証

ISO/SAE 21434 [RQ-10-10] では以下の検証方法を強調していますが、どのようにアプローチするべきかの詳細は示していません。しかし、これらの検証技法は、重要なアプリケーションで使用される中で証明されているので、ここでのベストプラクティスを示す適切なケースがあります。

要件ベースのテスト

要件ベースのテストでは、コードの全体または一部分を動的にテスト（実行）し、そのコードがソフトウェア要件（およびそれらの要件から派生した中間設計）を満たしており、指定されていない機能や冗長なコードが含まれていないことを実証します。構造カバレッジ解析は、これらの要件ベースのテスト手順の実行中に、どのコード構造やコンポーネントインターフェースが実行されなかったかを判定します。コードの実行されていない部分には、さらに解析と適切な改善が必要です。この処理を完了するためにツールを使用することを、この規格は要求していませんが、トリビアルなものを除いて、すべてのアプリケーションでそうする方が効率的です（図 5）。

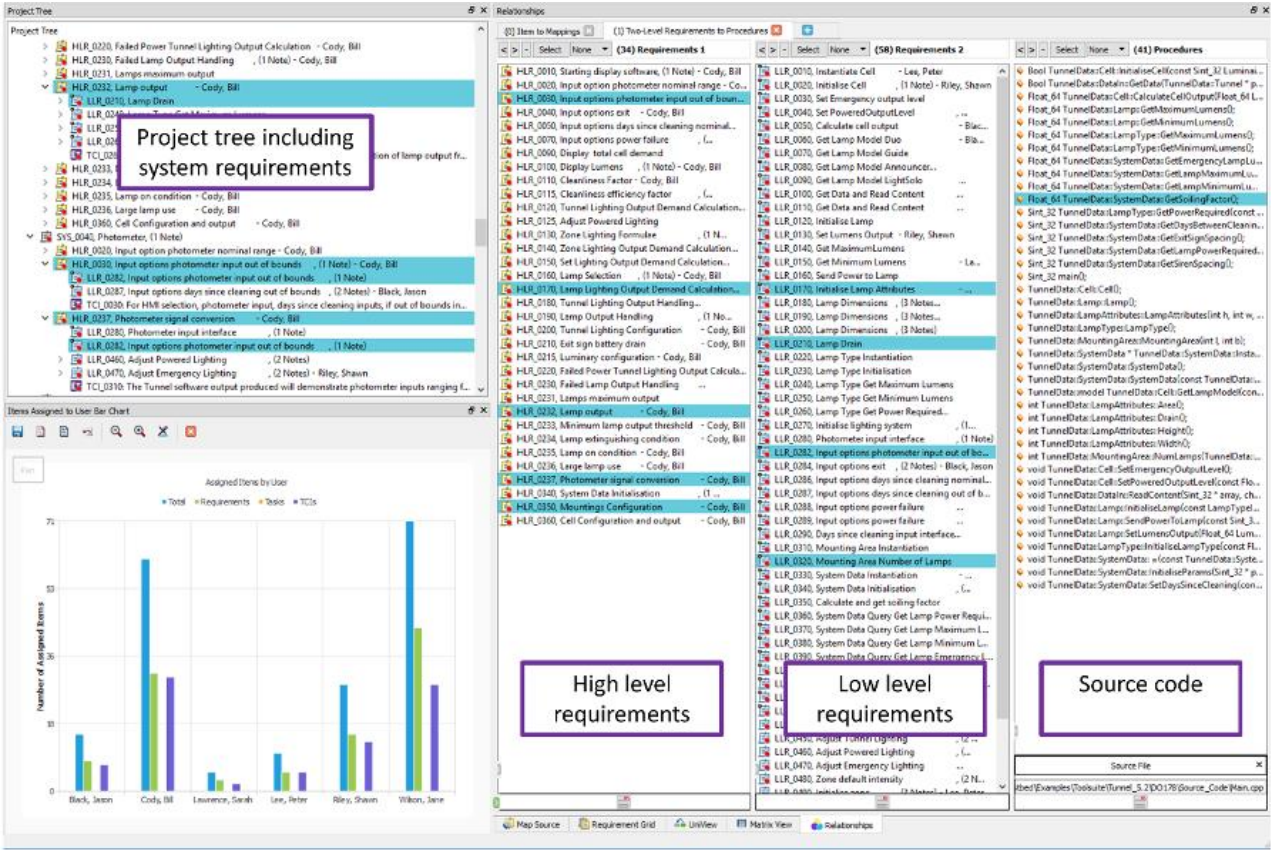

図 5　自動化された双方向の要件トレーサビリティにより、要件ベースのテストが容易になります
出典：LDRA

インターフェーステスト

インターフェーステストでは、ソフトウェアシステムやサブシステム、コンポーネント間の通信が正常に動作するかどうかを検証します（図 6）。

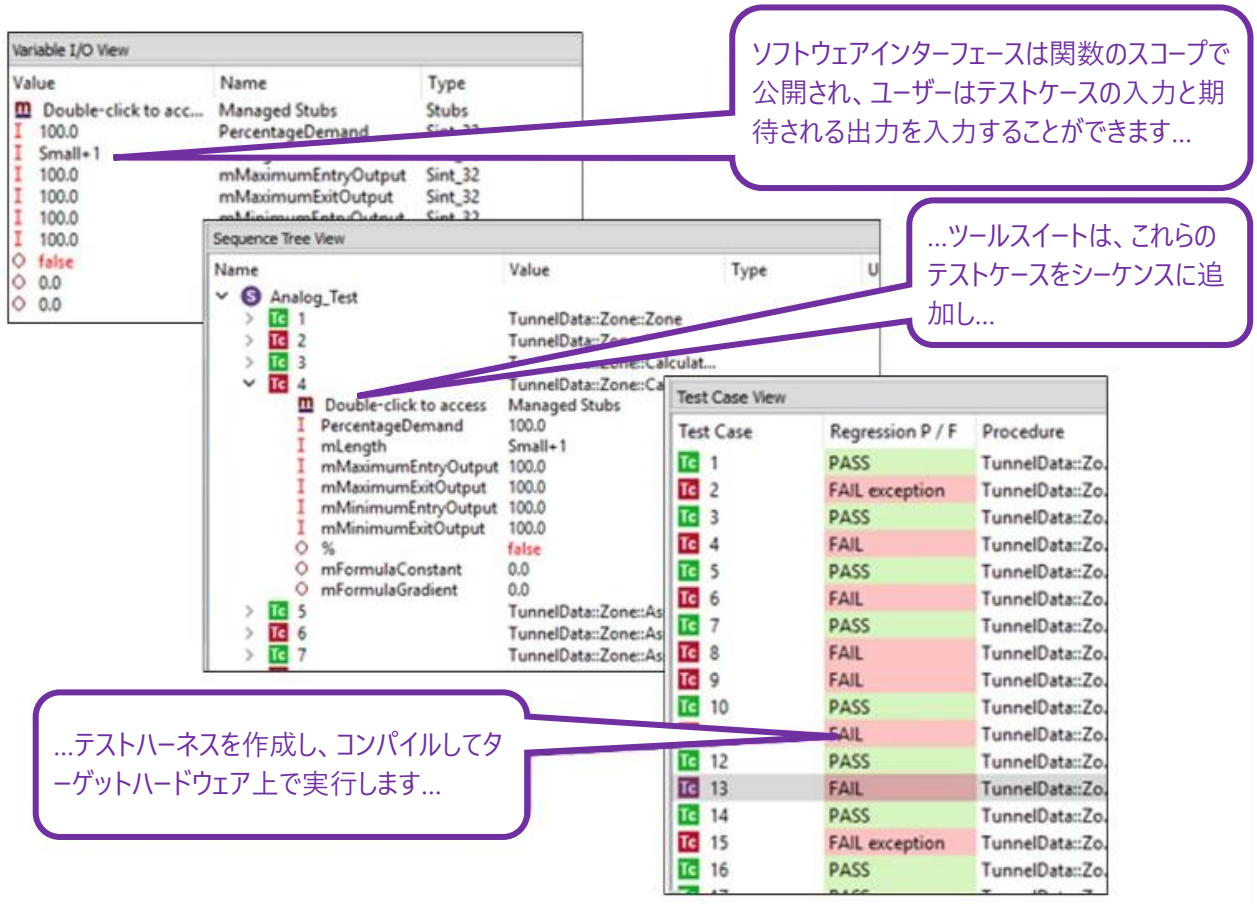

図 6　LDRA ツールスイートの TBrun コンポーネントは、コードカバレッジ解析と組み合わせて、インターフェースの正しい機能を実証するプラットフォームを提供します

リソース使用状況の評価

メモリ、タイミング、ファイルシステムなどのリソースを十分に確保し、競合の問題を排除することは、特に悪人から攻撃を受けている場合、接続されたシステムにとって重要な考慮事項です。マルチコアプロセッサの出現により、十分なリソースを確保することの重要性が高まっていますが、それらのリソースを使用可能として確保することはより困難になっています。

たとえば、最悪実行時間（WCET）の計算を考えてみます。Reinhard Wilhelm らによると、数学的解析で WCET の決定的な値を算出することは、一般の場合には解決不能です。したがって、純粋な静的解析のアプローチでは近似が必要になります。

その結果、ツールは「大事を取って」誤ってしまうのです。これは何もしないよりはましですが、精度がすべてである環境では理想的ではありません。しかし、選択したプラットフォームでの実行に依存しない、ソフトウェアコードの特性を測定するための実証済みのメカニズムがあります。たとえば、Halstead のメトリクスは、ソフトウェアモジュールのサイズ、ソフトウェアの複雑さ、データフロー情報を評価するため、さまざまな言語でのアルゴリズム実装を反映しています。そして、それらはソースコードの静的解析から正確に算出することができます（図 7）。この方法では、コードのどの部分が最も処理時間を要しているか特定することはできますが、最大経過時間の絶対値を提供することはできません。

Halsteads (*Cashregister.c*)

File	Total Operators	Total Operands	Unique Operators	Unique Operands	Vocabulary	Length	Volume
Total for Cashregister.c	149	230	16	56	72	379	2338

図 7　Halstead のメトリックが LDRA ツールスイートで計算されています

同じ静的解析により、このコードベースに関連するコールグラフも生成され、コードベース全体のコンテキストで最も負荷の高い関数が実行される場所が可視化されます(図 8)。

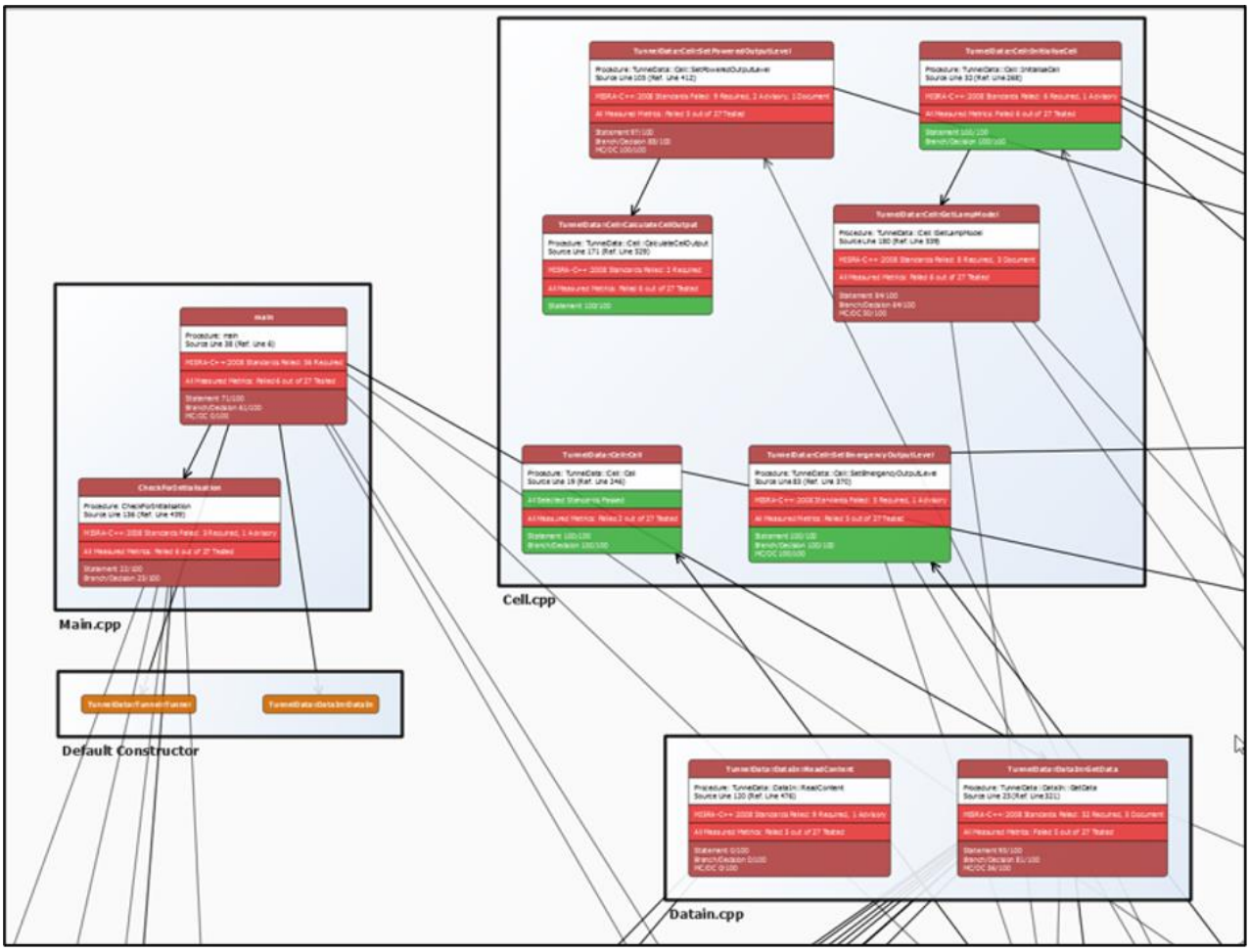

図 8　LDRA ツールスイートのコールグラフは、最も負荷の高い関数が呼び出される場所を可視化します

開発ツールを使用すると、静的解析に基づいた概算に頼らず、動的にこれらのクリティカルな実行時間を測定できます（図 9)。

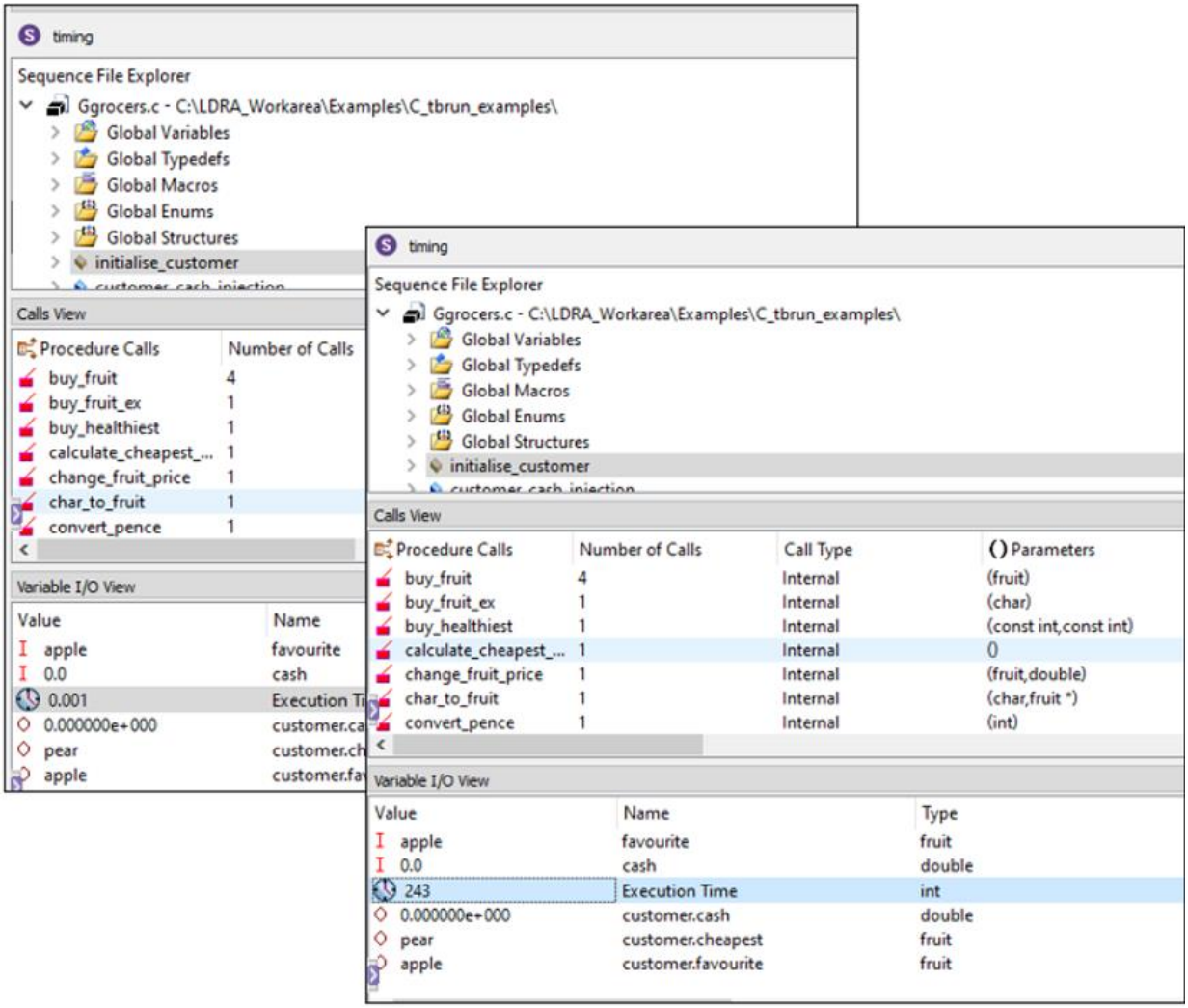

図 9　LDRA ツールスイートの TBrun コンポーネントを使用してタイミング解析が行われます

制御フローとデータフローの検証

データフローや制御フローに欠陥があると、コードの脆弱性につながる可能性があります。制御フロー/データフロー解析で、両者がシステム設計に従っていることを確認します。ホストと組込み両方のソフトウェアの解析にグラフィカルな静的・動的解析機能を提供するツールは、制御フローを公開し、フローグラフの形で図示します。これらは、実行パスと重ね合わせることができます（図 10）。

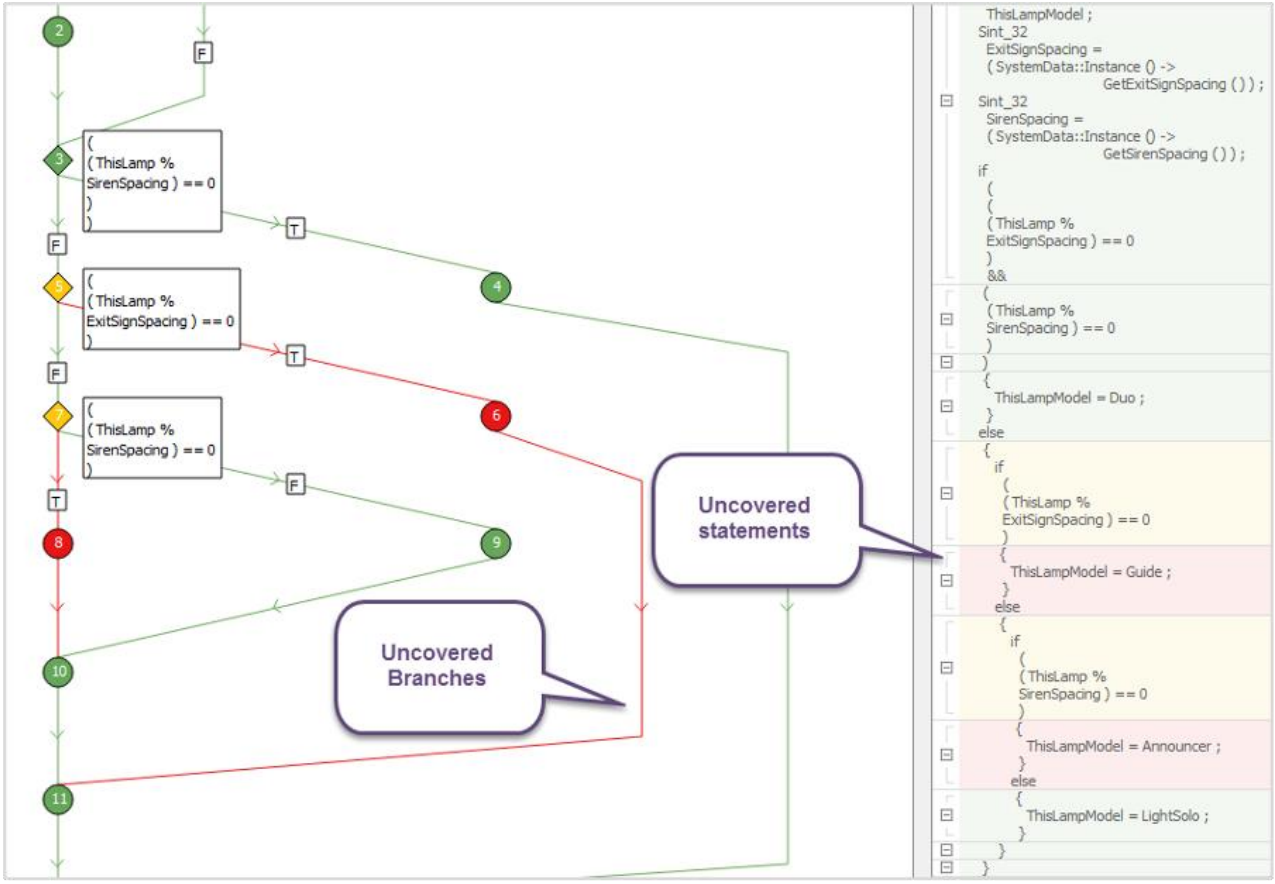

図 10　コードカバレッジをフローグラフで可視化したもの　出典：LDRA

動的解析

動的解析は、実行時にソフトウェアの挙動を解析することで、ソフトウェアの確認や検証を行うものです。動的解析ツールには、コード（構造）カバレッジ解析と単体テストに対応するものがあります。開発および統合時のコードのコンテキストにおける動的解析や DAST は、通常、オブジェクトコードの実行に対してソースコードが利用可能で、マッピングされた「ホワイトボックス」解析を意味します。

ファジングやペネトレーションテストのような、より伝統的な「ブラックボックス」セキュリティ技法は、ISO/SAE 21434 の開発ライフサイクルで依然として一定の役割がありますが、それらはシステムの完成後に、開発中に取られた予防措置の成功を確認するために用いることが最も適しています。

静的解析

静的解析は、ソースコードを解析することでソフトウェアを確認・検証します（図 11）。静的解析ツールそれぞれでコーディング規約違反の微妙なニュアンスを特定する能力が異なりますが、高度な実装では追加の処理が必要になるので、遅く感じられることがあります。開発初期は「軽量」モードで実行し、開発が進むにつれてより完全な解析を適用するオプションを備えたツールを選択するのが賢明な方法です。

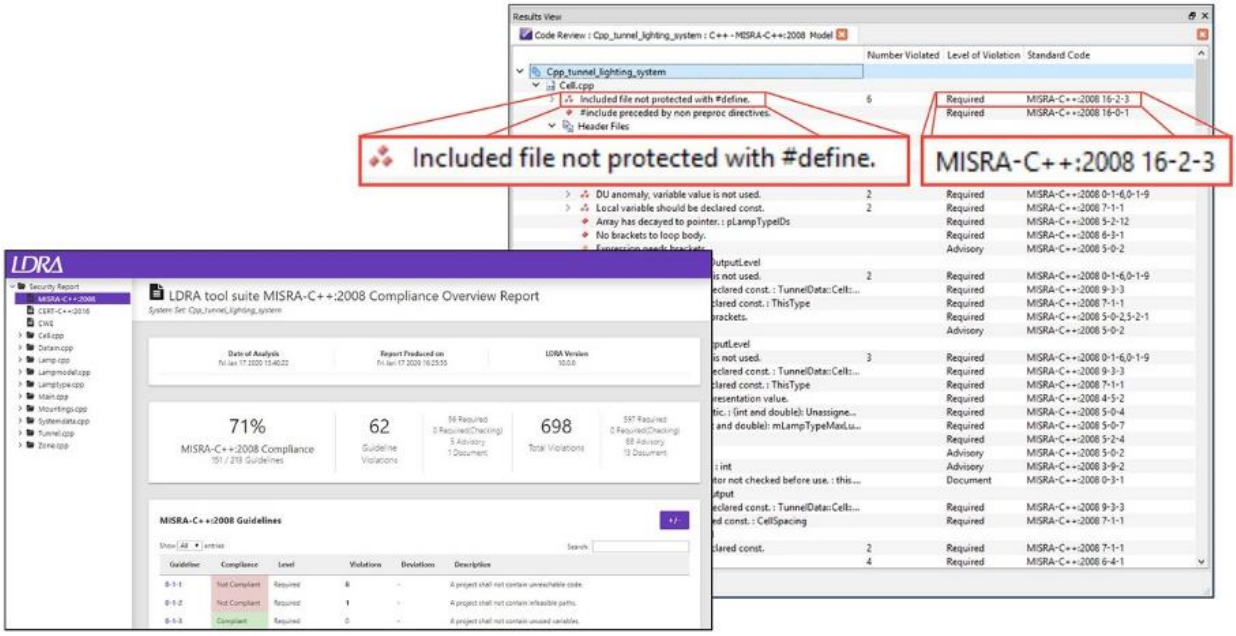

図 11　LDRA ツールスイートは、コードの複雑さに関連するものを含む品質メトリックを生成し、ISO/SAE 21434 に従って指定されたコーディング規約への準拠をチェックします

開発ツールスイート

ソフトウェアの変更時にコーディング規約を実施する、単体テストを再実行（リグレッションテスト）する、セキュリティ侵害時の影響分析を適用するといった機能は、開発中の変更や導入後の侵害に効果的かつ効率的に対応するための非常に貴重なツールです。統合開発ツールスイートは、これらの機能すべてを単一の環境で提供でき、開発プロセスと発売後の製品サポートとに等しく役立ちます。

ISO/SAE 21434 は、その目的を達成する方法の詳細なガイダンスがないにもかかわらず、ソフトウェア開発者が達成すべき価値ある一連の目標を提示しています。実績のあるツールは、適切に適用すると、これらの目標を達成するための実用的なアプローチを提供します。

Mark Pitchford は LDRA Software Technology のテクニカルスペシャリストで、DO-178、IEC 61508、ISO 26262、IIRA、RAMI 4.0 などの規格に取り組みながら、安全性とセキュリティクリティカル環境においてコンプライアンスに則ったソフトウェア開発を目指す開発チームと協力してきました。

FUJI SETSUBI

富士設備工業株式会社 電子機器事業部　www.fuji-setsu.co.jp