# MBAC

# DO-331 解説書

目次

## １．本書の目的

本解説書は、Model Based Aviation Consortium (MBAC) の DO-331 ・ モデル検証 Working Group の成果として発行する。
本解説書は、航空機搭載ソフトウェア開発をモデルを使用して行う場合において、型式証明を取得するために準拠が求められる、DO-331 についての概要、また適合に対する方法、知見を共有するために作成する。

## ２．用語

DO-331:
タイトルは「Model-based Development and Verification Supplement to DO-178C and DO-278A 」
DO-178C の補足として、MBD に対する認証プロセスについて定めたガイドライン。

MBD:
Model Based Development
ソフトウェア（又はシステム）の仕様をモデルにより表現し、設計・製造・検証に活用すること。

Model:
分析、検証、シミュレーション、コード生成、またはそれらの組み合わせに使用される、システムの特定の側面のセットの抽象表現。モデルは、抽象化のレベルに関係なく、明確である必要がある。

Specification Model:
ソフトウェアコンポーネントの機能、パフォーマンス、インターフェース、または安全性の特性を抽象的に表現する、High Level Requirements を表すモデル。Specification Model では、内部データ構造、内部データフロー、内部制御フローなどのソフトウェア設計の詳細は定義されない。

Design Model:
Low Level Requirements、ソフトウェアアーキテクチャ、アルゴリズム、コンポーネントの内部データ構造、データフロー、制御フローなどのソフトウェア設計を定義するモデル。ソースコードの生成に使用されるモデルは、Design Model。

HLR:

High Level Requirements

LLR:

Low Level Requirements

MILS:

Model In the Loop Simulation

PILS:

Processor In the Loop Simulation

HILS:

Hardware In the Loop Simulation

ACG:

Auto Code Generation

TQL:

Tool Qualification Level

EOC:

Executable Object Code

## ３．DO-331 の概要

DO-331 は、RTCA により発行された、モデルを使用してソフトウェアを開発、検証するためのプロセスについて定めたガイドラインである。航空機のソフトウェア開発、認証には、DO-178C への準拠がデファクトスタンダードとして要求される。DO-331 は、モデルを使用してソフトウェア開発を行う際の、追加の要件を、サプリメントとしてまとめた文書である。

MBD を開発に導入する利点 ：

- 統一されたモデリング言語を使用することにより、技術者間の認識共有を容易にする。 また、ダイアグラムは、自然言語よりも多くの情報を視覚的に伝達することができ、開発対象の理解を深めることに役立つ。
- モデルは、シミュレーションや静的解析にかけることができ、仕様の検証の効率、精度が向上する。
- モデルからソースコードを自動生成することが可能であり、ソフトウェア開発効率が向上する。

MBD を開発に導入する利点については、 MBAC 刊行物「MBD/ACG 普及ガイドライン」にも記載している。

DO-331 で扱うモデルの定義：

(1) Specification model

HLR を表現するモデルであり、ソフトウェアコンポーネントの機能、性能、インターフェース、安全に関わる特徴を表現するものである。SysML や UML といったモデリング言語で表現される。Software Design Model より高い抽象度のモデルであるため、Simulation や Auto Code Generation に対しては情報が不足している。システムやソフトウェア設計の初期から記述することができ、システムやソフトウェアの構造や振る舞いを視覚的にわかりやすく表現することが可能である。

(2) Design model

ソフトウェアコンポーネント内の内部データ構造、data flow/control flow を表現するモデルである。厳密な、software architecture 及び LLR が含まれており、Simulation や Auto Code Generation に使用することができる。自動コード生成により、ソースコード生成の入力となるモデルは、design model として扱われる。

モデルの適用の方法　：

DO-331 では以下の適用方法の例が挙げられているが、これだけに限定されない。

表 3.1 Model Usage Examples

| Process | MB Example 1 | MB Example 2 | MB Example 3 | MB Example 4 | MB Example 5 |
|---|---|---|---|---|---|
| System Requirement/ System Design | System Requirements | Requirements from which the Model is developed | Requirements from which the Model is developed | Requirements from which the Model is developed | Requirements from which the Model is developed |
| | | | | | Design Model |
| Software Requirement | HLR (textual) | Specification Model | Specification Model | Design Model | |
| Software Design | Design Model | Design Model | LLR (textual) | | |

## ４．DO-331 と DO-178C の違い

MBD を開発に導入した場合、DO-331 の DO-178C からの差異は以下となる。

DO-178C は、MBD を使用しない場合のソフトウェア認証ガイドラインであり、一般的なソフトウェア開発の認証において適用する。要件をはじめとした Life Cycle Data をテキストベースで作成し、マニュアルでコーディングするプロセスを想定している。これに対して DO-331 は、MBD をソフトウェア開発に取り入れる場合の Supplement として、追加の要件、変更を記載している。

表 4.1 DO-331 の DO-178C からの主な変更

| DO-178C | DO-331 |
|---|---|
| 設計 Life Cycle Data は、テキストベースのドキュメントで表記される。 | 設計 Life Cycle Data (HLR, LLR) をモデル (Specification Model, Design Model) で表記することができる。 |
| 設計 Life Cycle Data は Review & Analysis で検証される。 | Simulation 可能なモデリングツールを採用している場合、検証に Simulation を併用することができる。 |
| Coding は、人手による Manual Coding | Auto Code Generation が可能なモデリングツールを採用している場合、Auto Code Generation を用いることができる。 |
| Code の検証は、Requirement Base Test で検証する。 | Code Generator が TQL-1 の場合、Design Model と Code の一致性検証を省略できる。<br>上記 TQL に満たない場合にも、Design Model と Code の一致性を検証するためのツールが提供される場合、これにより一致性を検証できる。 |

従来のソフトウェア開発と、DO-331 を適用したモデルベース ソフトウェア開発のプロセスの違いを図示する。図 3.1 の MB Example 1 の適用方法を例として示す。

Design Process と Coding Process が、Modeling と Auto Coding に置き換わり、それぞれの検証活動に、Simulation 及び Tool の支援を適用することができる。

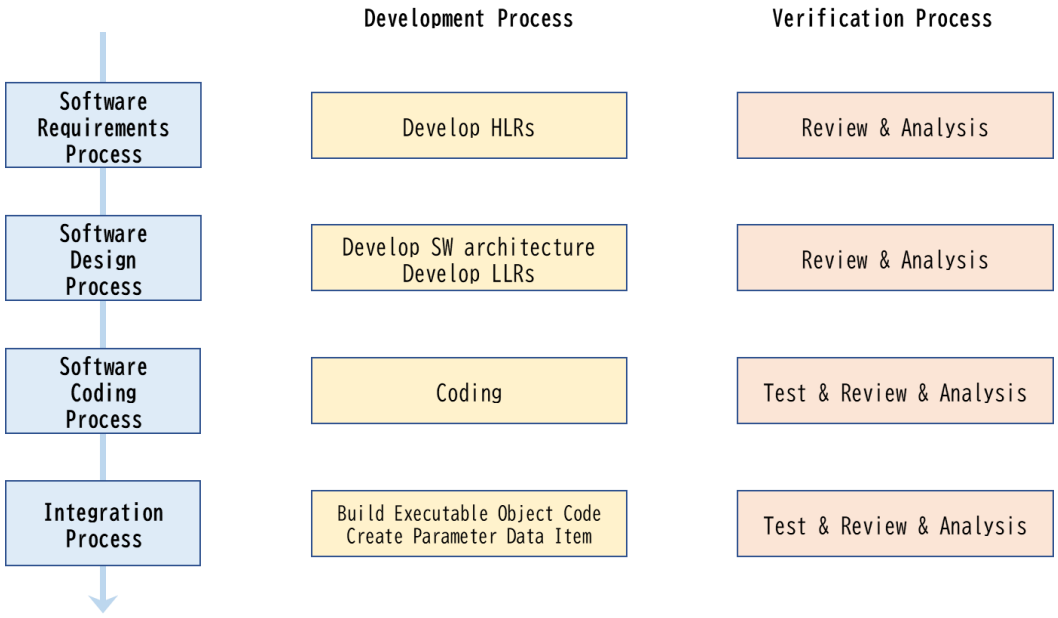



図 4.1 DO-178C ソフトウェア プロセス

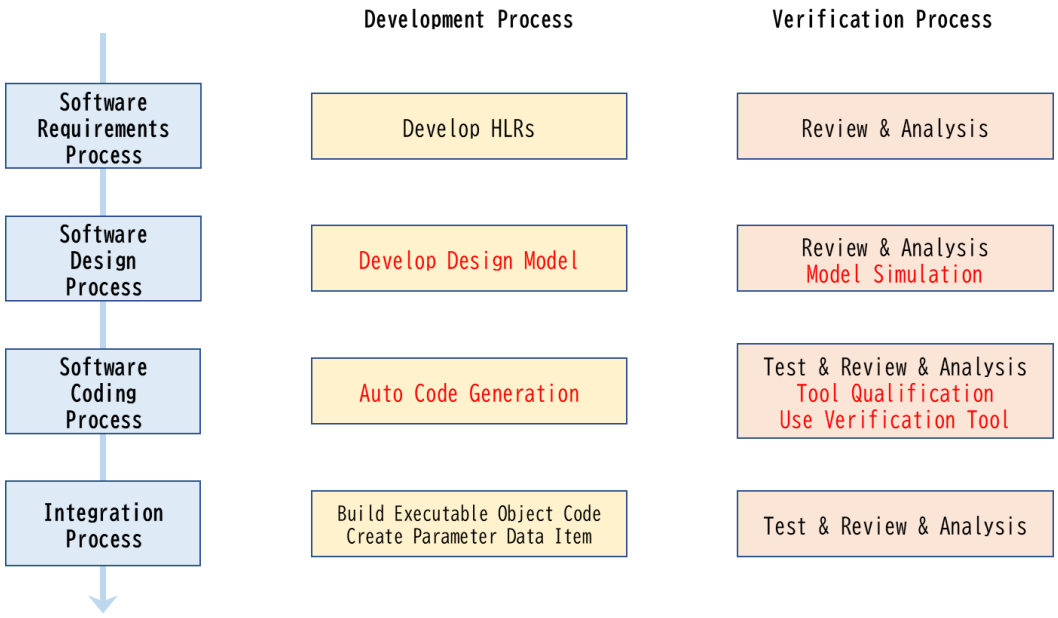



図 4.2 DO-331 ソフトウェア プロセス

MBD を利用する場合、DO-331 で追加される、ソフトウェア開発 Activity & Objective には以下の様なものがある。これらをプロセス及びツールにより達成する必要がある。

- 基本的な開発プロセスにおいては通常のソフトウェア開発と同様である。
- ソフトウェア開発を行う際に作成する HLR 及び LLR に加えてモデルを使用する場合、モデルの作成規約、管理要領、検証計画を制定する。
- ツールを選定する。必要に応じて、DO-330 のガイドラインに基づいて Tool Qualification を取得する。
- 開発するモデルは上位の要求にトレースできるか、派生した機能をモデルに組み込む場合は根拠を明確にする。
- モデルに対する静的解析を行う。また、モデル上でのシミュレーション、動的検証を行う。
- シミュレーションケースが正しいことを検証する。
- シミュレーション手順が正しいことを検証する。
- シミュレーション結果が正しいことを検証する。
- ツールが対応している場合、モデルからソースコードを自動生成する。
- ツールが対応している場合、モデルとソースコードが一対一対応していることを検証する。

これらの、モデルを用いて可能となる Objective を達成することにより、以下のメリットを得ることが MBD を導入する利点である。

- DO-178C で要求されるいくつかの Activity を省略可能となる場合がある。
- 従来 Software が完成した後でなければできなかった検証の一部を、設計の上流工程で前倒して行うことが可能となる。これにより、不具合を早期に発見し、戻り作業を削減することができる。

Tool Qualification については、DO-330 で規定されている。ツールのエラーはソフトウェアの品質に悪影響を及ぼす可能性があるため、ツールは十分に適切なプロセスに基づいて開発され、検証され、Tool の機能の完全性を保証する必要がある。
TQL は Level 1～5 に分類され、Level 1 が最高レベルである。
ソフトウェア開発に用いられるツールに求められる TQL は、開発対象の Software Level と Criteria により決定される。Software Level は安全性への影響解析の結果より決定される Software の重要度であり、Criteria は対象ツールの使用用途により決定される。

DO-178C (Figure 1-1) の各ソフトウェア・ライフサイクル・プロセスに対する、DO-331 の追加のアクティビティを下表にまとめる。

表 4.2 MBD 適用時の追加アクティビティ

| Software Lifecycle Process | DO-178C | DO-331 |
|---|---|---|
| Software Planning Process | Software Planning Process (Section 4) | モデルの管理、作成、検証に対する計画、標準の作成。 |
| | | Modeling Tool の選定。 |
| | | Tool Qualification の取得 |
| Software Development Process | Software Requirement/Design Process (Section 5) | モデリングの実施 |
| Integral Process | Software Verification Process (Model) (Section 6) | モデルとその源泉要求とのトレーサビリティ管理 |
| | | モデルの静的解析 |
| | | モデルに対するテスト（Simulation） |
| | | モデル上でのテストカバレッジ計測 |
| | | シミュレーションケースが正しいことの検証 |
| | | シミュレーション手順が正しいことの検証 |
| | | シミュレーション結果が正しいことの検証 |
| Software Development Process | Software Coding Process (Section 5) | モデルからソースコードの自動生成 |
| Integral Process | Software Verification Process (Code) (Section 6) | モデルとソースコードの一致性検証 |
| | | ソースコードとコード生成元モデルとのトレーサビリティ管理 |

## 5. DO-331 各フェーズの解説

DO-331 は、Modeling Tool を特定せず、一般化された Modeling Approach のガイドラインとなっている。本資料では、理解を促進するため、Mathworks Tool を使用し、具体例を交えて解説をする。Mathworks Tool 群が取得できる TQL について、表 5.1 に示す。

本書の例では、図 3.1 の MB Example 1 の適用方法の採用を想定したプロセスである。したがって、モデルが使用されるフェーズは、Software Design Process 以降となる。対象のプロセスに適用される Objective Table A-4, A-5, A-6, A-7 について詳述する。

以下は MBD を適用した際の、プロセスと Life Cycle Data の一例である。各プロセスの Life Cycle Data は、その他の Life Cycle Data との適合性、及び標準への適合性を検証することが求められる。各プロセスで求められる検証 Activity と準拠するべき Objective の関連を以下に示す。

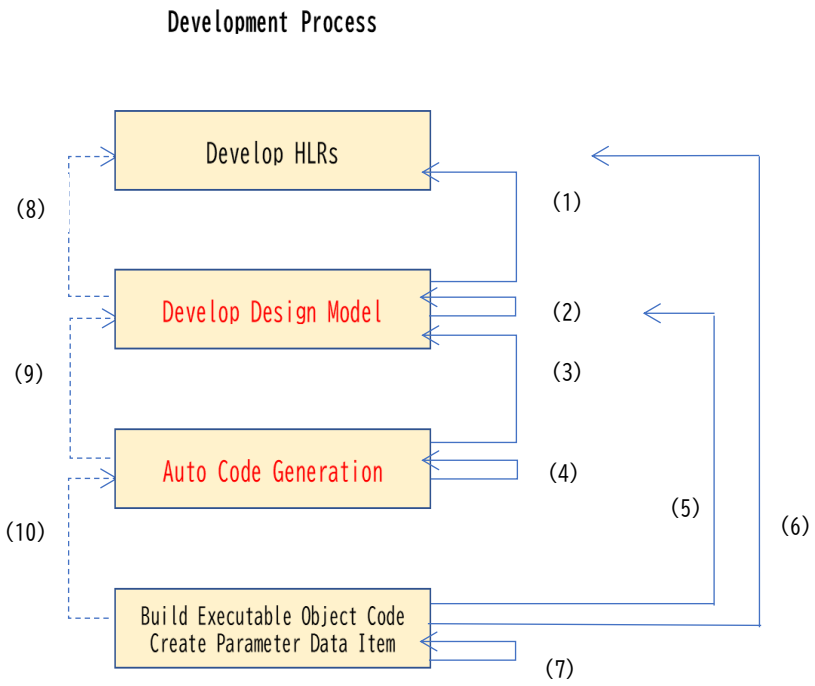



図 5.1 Software 開発プロセス（MB Example 1）と O-331 Appendix A Objective Table との関連

通常のソフトウェア開発と同様、以下の Activity が必要となり、以下の Object Table の各項に適合することが必要となる。

(1) LLR (Design Model) が HLR に適合していることを検証する。

【A-4】1, 8, 14, 15, 16

【A-7】1, 2, 3, 10, 11, 12

(2) LLR (Design Model) 自体が適切なものであるかを検証する。

【A-4】2, 3, 4, 5, 7, 9, 10, 11, 12, 13, 14, 15, 16

【A-7】1, 2, 4, 10, 11, 12

(3) Code が LLR (Design Model) に適合していることを検証する。

【A-5】1, 2

(4) Code 自体が適切であることを検証する。

【A-5】3, 4, 6

(5) Executable Object Code が LLR (Design Model) に適合していることを検証する。

【A-6】3, 4

【A-7】1, 2, 4, 5, 6, 7, 8

(6) Executable Object Code が HLR に適合していることを検証する。

【A-6】1, 2

【A-7】1, 2, 3, 5, 6, 7, 8

(7) Executable Object Code 自体が適切であることを検証する。

【A-5】7, 8, 9

【A-6】5

(8) LLR (Design Model) が HLR にトレースできることを検証する。

【A-4】6

(9) Code が LLR (Design Model) にトレースできることを検証する。

【A-5】5

(10) Executable Object Code が Code にトレースできることを検証する。

【A-7】9

DO-331 Appendix の Objective Table ごとの解説を次頁以降に示す。

表 5.1 Mathworks Toolbox で取得できる TQL

| Tool 名 | 使用するプロセス | Criteria | TQL |
|---|---|---|---|
| Requirement Toolbox | Software Requirement Verification | 3 | 5 |
| Simulink Report Generator | Software Requirement Verification | 3 | 5 |
| | Software Design Verification | 3 | 5 |
| Simulink Test | Software Design Verification | 3 | 5 |
| Simulink Check | Software Design Verification | 3 | 5 |
| Simulink Coverage | Software Design Verification | 3 | 5 |
| Simulink Design Verifier | Software Design Verification | 3 | 5 |
| Simulink Code Inspector | Software Code Verification | 3 | 5 |
| Polyspace Bug Finder | Software Code Verification | 3 | 5 |
| Polyspace Code Prover | Executable Object Code Verification | 2 | 4 |

**5.1 Table A-4 概要**

DO-331 Appendix Table A-4 は、Verification of Outputs of Software Design Process に適用する Objective の一覧表である。Software Design Process のアウトプットである LLR (Design Model)について検証する際の Objective が規定されている。 MB 6.3.2/6.3.3 に基づいて、Low Level Requirements 及び Software Architecture のレビュー、解析を行い、エラーを検出する。 DO-331 MB 6.3.2 は基本的に DO-178C 6.3.2 と同様であるが、LLR→Design Model と置き換え、Model の検証活動が必要である点が追加されている。MB 6.3.3 も DO-178C の同項と同様であるが、Software Architecture をモデルで表現した場合の追加の要件がある。
また Design Model の検証には、シミュレーションを活用することができるが、MB 6.8.3.2 が追加要件になっており、Simulation の Case、Procedure、Result の検証が要求されている。

Model を用いない従来型のソフトウェア開発では、設計結果 LLR の検証は、主にレビューにより行われていた。これに対し MBD を利用する場合、Simulation を検証に併用することができる。
Software Design Process では以下の Activity を実施する。この Activity を実施する際、Table A-4 の 以下に示す Objective を達成する必要がある。

◇ LLR (Design Model) を開発し、以下の検証を実施する

- Review, 解析を実施する （DO-178C と同様） A-4 ： 10*, 13*
- Simulation（テスト）を併用し、動的検証を実施する A-4 ： 1, 8, 14, 15, 16
- 静的解析を使用し、標準適合性等を検証する A-4 ： 2, 3, 4, 5, 7, 9, 11, 12
- HLR に対するトレースを行う （DO-178C と同様） A-4 ： 6

* : Target Computer への適合性及び Partitioning の適切さは、Model によらない設計。ハンドコードと同様にレビューにより、実装に際し、矛盾、もれなく設計検討されていることを検証する。

表 5.1.1 Table A-4 の Objective と使用するツール

| | DO-331 Table A-4 Objectives | 適用する Toolbox (*1) | (Ref) |
|---|---|---|---|
| 1 | Low-level requirements comply with high-level requirements.<br>LLR は HLR に適合すること<br>この Objective は、LLR が HLR を満たし、派生した要件と設計の基盤が正しく定義されていることを確認することです。設計モデル内の意図しない機能の存在を確実に検出できるようにすることでもあります。 | Simulink Coverage,<br>Simulink Design Verifier,<br>Simulink Test, | MB 6.3.2 a |
| 2 | Low-level requirements are accurate and consistent.<br>LLR は正確かつ一貫性があること<br>この Objective は、各 LLR が正確かつ明確であり、互いに矛盾しないことを保証することです。 | Simulink Check, Simulink Test, Simulink Design Verifier, | MB 6.3.2 b |
| 3 | Low-level requirements are compatible with target computer.<br>LLR はターゲットに適合すること<br>この Objective は、LLR とターゲットコンピュータのハードウェア/ソフトウェア機能、特にバス負荷、システム応答時間、入出力ハードウェアなどのリソースの使用との間に矛盾が存在しないようにすることです。 | Simulink Check, | MB 6.3.2 c |
| 4 | Low-level requirements are verifiable.<br>LLR は検証できること<br>この Objective は、各 LLR を確実に検証できるようにすることです。 | Simulink Check, Simulink Coverage, Simulink Design Verifier, Simulink Test, | MB 6.3.2 d |
| 5 | Low-level requirements conform to standards.<br>LLR は標準に準拠すること<br>この Objective は、ソフトウェア設計プロセス中にソフトウェア設計標準が遵守されていること、および標準からの逸脱が正当であることを確認することです。 | Simulink Check, Simulink Coverage, | MB 6.3.2 e |
| 6 | Low-level requirements are traceable to high-level requirements.<br>LLR は HLR にトレースできること<br>この Objective は、HLR と派生要件が LLR に確実に設計されるようにすることです。 | Simulink Check, | MB 6.3.2 f |
| 7 | Algorithms are accurate.<br>アルゴリズムが正しいこと<br>この Objective は、特に不連続性の領域において、提案されたアルゴリズムの正確性と動作を保証することです。 | Simulink Check, Simulink Design Verifier, Simulink Test, | MB 6.3.2 g |

| | DO-331 Table A-4 Objectives | 適用する Toolbox (*1) | (Ref) |
|---|---|---|---|
| 8 | Software architecture is compatible with high-level requirements.<br>ソフトウェアアーキテクチャは HLR へ適合すること<br>この Objective は、ソフトウェアアーキテクチャが HLR と矛盾しないようにすることです。 | Simulink Test, | MB 6.3.3 a |
| 9 | Software architecture is consistent.<br>ソフトウェアアーキテクチャは一貫性があること<br>この Objective は、ソフトウェアアーキテクチャのコンポーネント間に正しい関係が存在することを保証することです。 | Simulink Check, Simulink Design Verifier, Simulink Test, | MB 6.3.3 b |
| 10 | Software architecture is compatible with target computer.<br>ソフトウェアアーキテクチャはターゲットに適合すること<br>この Objective は、ソフトウェアアーキテクチャとターゲットコンピュータのハードウェア/ソフトウェア機能の間に競合、特に初期化、非同期操作、同期、割り込みが存在しないことを確認することです。 | N/A *2 | MB 6.3.3 c |
| 11 | Software architecture is verifiable.<br>ソフトウェアアーキテクチャは検証できること<br>この Objective は、ソフトウェアアーキテクチャが検証できること、たとえば、無制限の再帰アルゴリズムがないことを確認することです。 | Simulink Coverage, Simulink Test, Simulink Design Verifier, | MB 6.3.3 d |
| 12 | Software architecture conforms to standards.<br>ソフトウェアアーキテクチャは標準に準拠すること<br>この Objective は、ソフトウェア設計プロセス中にソフトウェア設計標準が遵守されていること、および標準への逸脱が正当であることを確認することです。 | Simulink Check, Simulink Coverage, | MB 6.3.3 e |
| 13 | Software partitioning integrity is confirmed.<br>ソフトウェアパーティショニングの確実性が確認されること<br>この Objective は、パーティション違反を確実に防止することです。 | N/A *2 | MB 6.3.3 f |
| MB14 | Simulation cases are correct.<br>シミュレーションケースが正しいこと | Simulink Test, | MB 6.8.3.2 a |
| MB15 | Simulation procedures are correct.<br>シミュレーション手順が正しいこと | Simulink Test, | MB 6.8.3.2 b |
| MB16 | Simulation results are correct and discrepancies explained.<br>シミュレーション結果が正しく、矛盾が説明されること | Simulink Test, | MB 6.8.3.2 c |

*1 Report Generator と DO Qualification Kit を除く

*2 Architecture の Target Compatibility （10） 及び Software Partitioning （13） は Design Process ではレビューで検証する。

**5.2 Table A-5 概要**

DO-331 Appendix Table A-5 は Verification of Outputs of Software Coding & Integration Processes に適用する objective の一覧表である。

Software Coding & Integration Processes のアウトプットについて検証する際の Objective が規定されており、MB6.3.4 に基づいて、ソースコードと LLR のレビュー、分析を行い、Software のコーディングプロセスにおいて発生し得るエラーを検出することを目的とする。

DO-331 MB6.3.4 は基本的には DO-178C 6.3.4 と同様であるが、「LLR がモデルで記述されている場合、LLR を表していないモデル要素にソースコードがトレースされないこと」が追記されている。

Software Coding & Integration Processes では、以下の Activity を実施し、ソースコードの妥当性を示すために、以下を行う。

- LLR とソースコードのトレースを行う　A-5 ： 1, 5
- 静的解析により構造解析を行う　A-5 ： 2, 3, 6
- 標準適合性を検証する　A-5 ： 4

表 5.2.1 Table A-5 の Objective と使用するツール

| | DO-331 Table A-5 Objectives | 適用する Toolbox (*1) | (Ref) |
|---|---|---|---|
| 1 | Source Code complies with lowlevel requirements.<br>lowlevel requirements への準拠<br>この objective は、ソースコードが lowlevel requirements に関して正確かつ完全であり、文書化されていない機能を実装するソースコードがないことを保証する。 | Simulink Code Inspector | MB.6.3.4.a |
| 2 | Source Code complies with software architecture.<br>ソフトウェア・アーキテクチャへの準拠<br>この objective は、ソース・コードがソフトウェア・アーキテクチャで定義されたデータ・フローと制御フローに一致していることを確認することである。 | Polyspace Code Prover | MB.6.3.4.b |
| 3 | Source Code is verifiable.<br>検証可能性<br>この objective は、ソースコードに検証できないステートメントや構造が含まれていないこと、また、テストするためにコードを変更する必要がないことを保証することである。 | Polyspace Code Prover<br>Simulink Code Inspector | MB.6.3.4.c |
| 4 | Source Code conforms to standards.<br>標準への適合<br>複雑さの制限やコードの制約などの検証である。複雑さには、ソフトウェアコンポーネント間の結合度、制御構造の入れ子レベル、論理式や数値式の複雑さなどが含まれる。この分析により、標準からの逸脱が正当化されることも保証される。 | Polyspace Code Prover<br>Polyspace Bug Finder | MB.6.3.4.d |
| 5 | Source Code is traceable to lowlevel requirements.<br>トレーサビリティ<br>この objective は、lowlevel requirements がソースコードに展開されたことを確認することである。 | Simulink Code Inspector | MB.6.3.4.e |
| 6 | Source Code is accurate and consistent.<br>正確さと一貫性<br>スタック使用量、メモリ使用量、固定小数点演算のオーバーフロー、浮動小数点演算、リソース競合と制限、ワーストケース実行タイミング、例外処理、未初期化変数の使用、キャッシュ管理、未使用変数、タスクや割り込み競合によるデータ破損など、ソースコードの正確さと一貫性を判断することが目的です。コンパイラ(そのオプションも含む)、リンカ(そのオプションも含む)、およびいくつかのハードウェア機能は、ワーストケース実行タイミングに影響を与える可能性があり、この影響を評価する。 | Polyspace Code Prover<br>Polyspace Code Prover | MB.6.3.4.f |

| | DO-331 Table A-5 Objectives | 適用する Toolbox (*1) | (Ref) |
|---|---|---|---|
| 7 | Output of software integration process is complete and correct.<br>Software integration process の成果物が正しいこと。 | N/A | MB. 6.3.5 a |
| 8 | Parameter Data Item File is correct and complete.<br>Parameter Data Item File が正しいこと。 | N/A | MB. 6.6.a |
| 9 | Verification of Parameter Data Item File is achieved.<br>Parameter Data Item File 検証可能であること。 | N/A | MB. 6.6.b |
| ※ Output of software integration process is complete and correct(7), Parameter Data Item File is correct and complete(8), Verificaion of Parameter Data Item File is achived(9)は MBD の範囲外であるため従来の手法で実施する。 | | | |

*1 Report Generator と DO Qualification Kit を除く

**5.3 Table A-6 概要**

A-6 は、Testing of Outputs of Integration Process である。

Integration Process のアウトプットについてテストする際の Objective が規定されている。DO-178C の Sec.6.4 に基づいて、テストを行い EOC が High-level requirements および Low-level requirements への適合性、堅牢性を確認する。ただし、EOC の検証にシミュレーションを使用する場合には、MB.6.8.2a の活動が追加で必要となる。

モデルシミュレーションを使用して要件を満たす場合には、以下対応(MB.6.8.2a)が必要となる：

1．ソースコード及び EOC の製作に使用したものと同じデザインモデルを使用してモデルシミュレーションを実施する。
2．どのソフトウェアテスト、カバレッジの Objective 及び HLR がモデルシミュレーションによって達成されるのかを明確にする。ターゲットコンピュータ環境でのテストが必要となる項目もあることを考慮する。
3．シミュレーションによってソフトウェアテストとカバレッジの Objective が達成されること（上記 2 での決定内容）を証明する為、以下を検証する。
   i. モデルシミュレーション環境とターゲットコンピュータ環境の等価性・違い
   ii. シミュレーション EOC とターゲット EOC の等価性、異なる点の明確化

表 5.3.1 Table A-6 の Objective と使用するツール

| | DO-331 Table A-6 Objectives | 適用する Toolbox (*1) | (Ref) |
|---|---|---|---|
| 1 | Executable Object Code complies with the high-level requirements.<br>EOC は HLR に適合すること<br>HLR に対してテストケース(通常範囲)を作成し、Simulink Test でテストを行う。ソフトウェア要件、テストケース、テスト手順、テスト結果は関連性を示す。要求ベーステストは PIL にて実施されていること。 | Simulink Test,<br>Simulink Design Verifier,<br>Embedded Coder-PIL Mode | 6.4.a |
| 2 | Executable Object Code is robust with the high-level requirements.<br>EOC は HLR に対して堅牢性があること<br>HLR に対してテストケース(異常範囲)を作成し、テストを行う。ソフトウェア要件、テストケース、テスト手順、テスト結果は関連性を示す。要求ベーステストは PIL にて実施されていること。 | Simulink Test,<br>Simulink Design Verifier,<br>Embedded Coder-PIL Mode,<br>Polyspace Code Prover,<br>Polyspace Code Prover Server | 6.4.b<br>FM.6.7.b<br>FM.6.7.c |
| 3 | Executable Object Code complies with the low-level requirements.<br>EOC は LLC に適合すること<br>LLR に対してテストケース(通常範囲)を作成し、テストを行う。ソフトウェア要件、テストケース、テスト手順、テスト結果は関連性を示す。要求ベーステストは PIL にて実施されていること。 | Simulink Test, Simulink Design Verifier,<br>Embedded Coder-PIL Mode | 6.4 |
| 4 | Executable Object Code is robust with the low-level requirements.<br>EOC は LLR に対して堅牢性があること<br>LLR に対してテストケース(異常範囲)を作成し、テストを行う。ソフトウェア要件、テストケース、テスト手順、テスト結果は関連性を示す。要求ベーステストは PIL にて実施されていること。 | Simulink Test,<br>Simulink Design Verifier,<br>Embedded Coder-PIL Mode,<br>Polyspace Code Prover,<br>Polyspace Code Prover Server | 6.4.d<br>FM.6.7.b<br>FM6.7.c |
| 5 | Executable Object Code is compatible with the target computer.<br>EOC はターゲットに適合すること<br>ターゲット上でのみ発生するようなエラーに関しても検証する。(PIL が必要)。 | Embedded Coder-PIL Mode | 6.4.e |

*1 Report Generator と DO Qualification Kit を除く

**5.4 Table A-7 概要**

A-7 は、Verification of Verification Process である。

Verification Process の Output を検証する際の Objective が規定されている。テスト手順とテスト結果の検証は DO-178C と同様であり、6.4.5 項に基づきレビューと分析を実施する。

テストカバレッジ分析の手法についても基本的には DO-178C と同様であり、6.4.4 項に基づき要求ベースのテストがソフトウェア要件の実装を十分に検証したかを判断するために HLR および LLR に対するテストカバレッジを分析する。EOC 検証のために実施したシミュレーションでテストカバレッジ分析の要求を達成する場合には、HLR は MB6.8.2.a、LLR では MB6.7 のガイダンスに従いモデルカバレッジ分析を実施する。

ソフトウェア構造のテストカバレッジ解析において、EOC 検証で実施したシミュレーションのモデルカバレッジを使用する場合には、MB.6.8.2. b のガイダンスに従い、モデルシミュレーションに使用するソースコードが、ターゲットで動作する EOC を作成するソースコードと等価であることを示さなければならない。

シミュレーションケース、手順および結果は、MB6.8.3.2 のガイダンスに基づきレビューと分析を実施する。

表 5.4.1 Table A-7 の Objective と使用するツール

| | DO-331 Table A-7 Objectives | 適用する Toolbox（*1） | （Ref） |
|---|---|---|---|
| 1 | Test procedures are correct.<br>テスト手順が正しいこと<br>期待される結果を含むテストケースがテスト手順に正しく展開されていることを検証する。 | Simulink Test,<br>Simulink Design Verifier,<br>Embedded Coder-PIL Mode | 6.4.5.b |
| 2 | Test results are correct and discrepancies explained.<br>テスト結果が正しく、矛盾点は説明されていること<br>テスト結果が正しいことを確認すること。実際の結果と期待される結果が不一致の場合は説明すること。 | Simulink Test | 6.4.5.c |
| 3 | Test coverage of high-level requirements is achieved.<br>HLR のテストカバレッジが達成されていること<br>HLR に対してテストカバレッジを対応すること。また、テストは PIL で行うこと。HLR とテストケースが漏れなくトレースできていることを確認する。各テストケースは正常範囲と異常範囲ともに行う。テストカバレッジに不足があった場合は追加または補強する。 | Simulink Test,<br>RequirementsToolbox,<br>Embedded Coder-PIL Mode | 6.4.4.a |
| 4 | Test coverage of low-level requirements is achieved.<br>LLR のテストカバレッジが達成されていること<br>LLR に対してテストカバレッジを対応すること。また、テストは PIL で行うこと。LLR とテストケースが漏れなくトレースできていることを確認する。各テストケースは正常範囲と異常範囲ともに行う。テストカバレッジに不足があった場合は追加または補強する。 | Simulink Test,<br>RequirementsToolbox,<br>Embedded Coder-PIL Mode | 6.4.4b |
| 5 | Test coverage of software structure is achieved. (modified condition/decision)<br>ソフトウェアの構造カバレッジ(MCDC)が達成されていること<br>MCDC カバレッジを満たしていることを確認する。 | Simulink Test，Embedded Coder-PIL Mode | 6.4.4.c |
| 6 | Test coverage of software structure is achieved. (decision coverage)<br>ソフトウェア構造カバレッジ(decision coverage)が達成されていること<br>decision カバレッジを満たしていることを確認する。 | Simulink Test，Embedded Coder-PIL Mode | 6.4.4.c |

| | DO-331 Table A-7 Objectives | 適用する Toolbox (*1) | (Ref) |
|---|---|---|---|
| 7 | Test coverage of software structure is achieved. (statement coverage)<br>ソフトウェア構造カバレッジ(statement coverage)が達成されていること<br>statement カバレッジを満たしていることを確認する。 | Simulink Test, Embedded Coder-PIL Mode | 6.4.4.c |
| 8 | Test coverage of software structure is achieved. (data coupling and control coupling)<br>ソフトウェア構造カバレッジ(data coupling、control coupling)が達成されていること<br>データカップリング(データ受け渡し)とコントロールカップリング(呼び出し関係)を満たしていることを確認する。達成方法として、コードカバレッジを計測する。Statement カバレッジによるコンポーネント間の繋がりがわかるためデータカップリングを確認できる。また、コントロールカップリングについては Function Call が分かれば確認できる。 | Simulink Test | 6.4.4.d |
| 9 | Verification of additional code, that cannot be traced to Source Code, is achieved.<br>ソースコードにトレースできない追加のコードの検証が達成されていること<br>適切なカバレッジ基準に対するソフトウェア構造のテストカバレッジが達成されている。オブジェクトコードのレビューで達成する。代表的なモデルからのソースコードとオブジェクトコードをレビューすることで達成できる。<br>達成方法として、下記の 3 つがある。<br>1. オブジェクトコードで要求ベーステストのカバレッジをみる。<br>2. ソースコードとオブジェクトコードの比較レビューにより確認する。代表的なケースを実施する。<br>3. Additional コードを作らないコンパイラを使用する。一般的に、C言語のコンパイラは達成できている。 | Simulink Test,<br>Embedded Coder-PIL Mode | 6.4.4. c |
| MB10 | Simulation cases are correct. (See Item 2)<br>シミュレーションケースが正しいこと(Item2 参照)<br>HLR と LLR に対してシミュレーションケースが適切であること。 | Simulink Test | MB.6.8.3.2.a |
| MB11 | Simulation procedures are correct. (See Item 2)<br>シミュレーション手順が正しいこと(Item2参照)<br>期待値とシミュレーション手順を準備すること。 | Simulink Test | MB.6.8.3.2.b |

| | DO-331 Table A-7 Objectives | 適用する Toolbox (*1) | (Ref) |
|---|---|---|---|
| MB12 | Simulation results are correct and discrepancies explained. (See Item 2)<br>シミュレーション結果が正しく、矛盾点は説明されていること (Item2 参照)<br>期待値とシミュレーション結果の差異があるときは説明すること。 | Simulink Test | MB.6.8.3.2 |

*1 Report Generator と DO Qualification Kit を除く

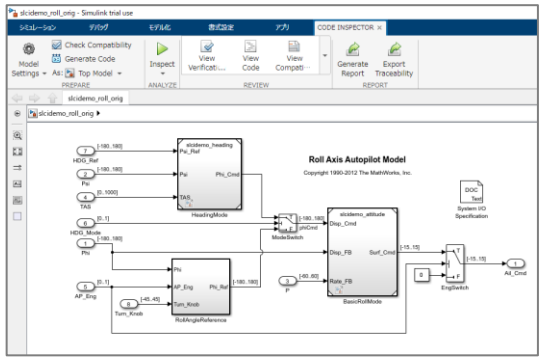
**Appendix A Verification of Outputs of Software Coding & Integration Process (Table A-5)検証手順**

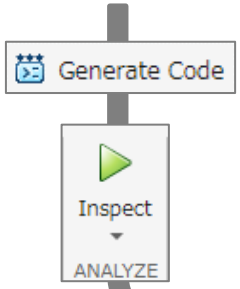
表 5.1 にある Mathworks Toolbox を使用し、DO-331 Appendix Table A-5 を検証する手順を以下に示す。

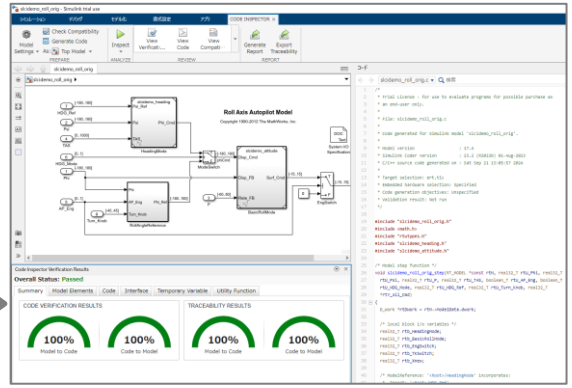
# 1．手順 Table-A5 No.1

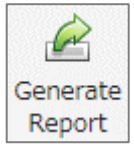
| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 1 | Source Code complies with lowlevel requirements. | MB.6.3.4.a | 低レベル要件への準拠：<br>このobjectiveは、ソースコードが低レベルの要求事項に関して正確かつ完全であり、文書化されていない機能を実装するソースコードがないことを保証することです。 | Simulink Code Inspector<br>**- Simulink Code Inspector Report**<br>**⇒ モデルとコードの相互のトレースデータにより、ソフトウエア要求(モデル)がコードにもれなく対応しているかを検証する。**<br>Simulink Code Inspector<br>- trace report<br>⇒ モデルとコードの相互のトレースデータにより、ソフトウエア要求(モデル)がコードにもれなく対応しているかを検証する。 |

## 画面操作

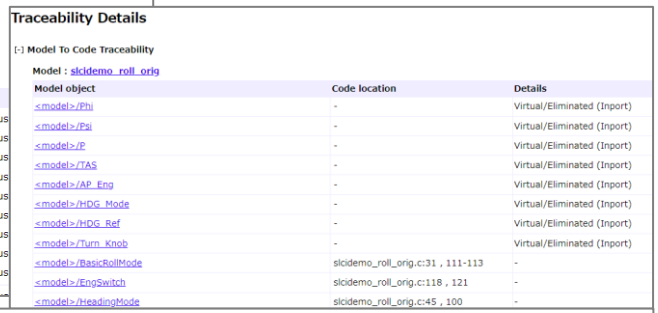


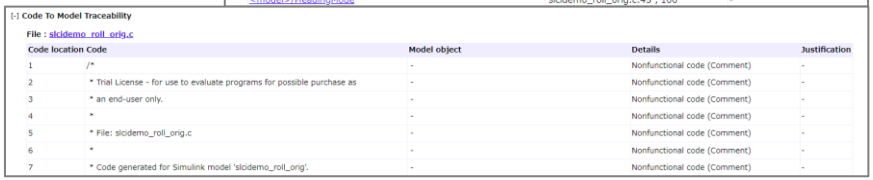




## 出力レポート：Simulink Code Inspector Report

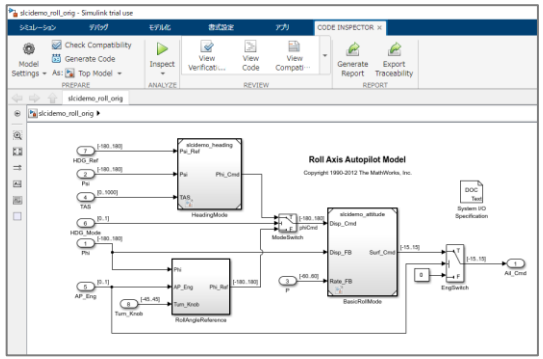



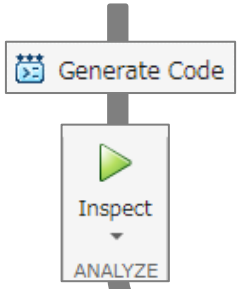



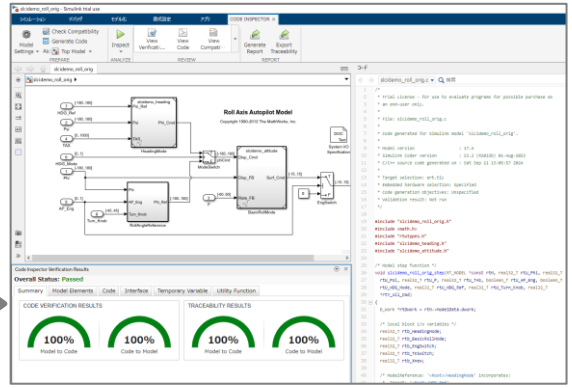
# 1．手順 Table-A5 No.1

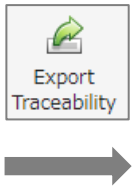
| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 1 | Source Code complies with lowlevel requirements. | MB.6.3.4.a | 低レベル要件への準拠：<br>このobjectiveは、ソースコードが低レベルの要求事項に関して正確かつ完全であり、文書化されていない機能を実装するソースコードがないことを保証することです。 | Simulink Code Inspector<br>- Simulink Code Inspector Report<br>⇒ モデルとコードの相互のトレースデータにより、ソフトウエア要求(モデル)がコードにもれなく対応しているかを検証する。<br>Simulink Code Inspector<br>**- trace report**<br>**⇒ モデルとコードの相互のトレースデータにより、ソフトウエア要求(モデル)がコードにもれなく対応しているかを検証する。** |

## 画面操作




## 出力レポート：-trace report

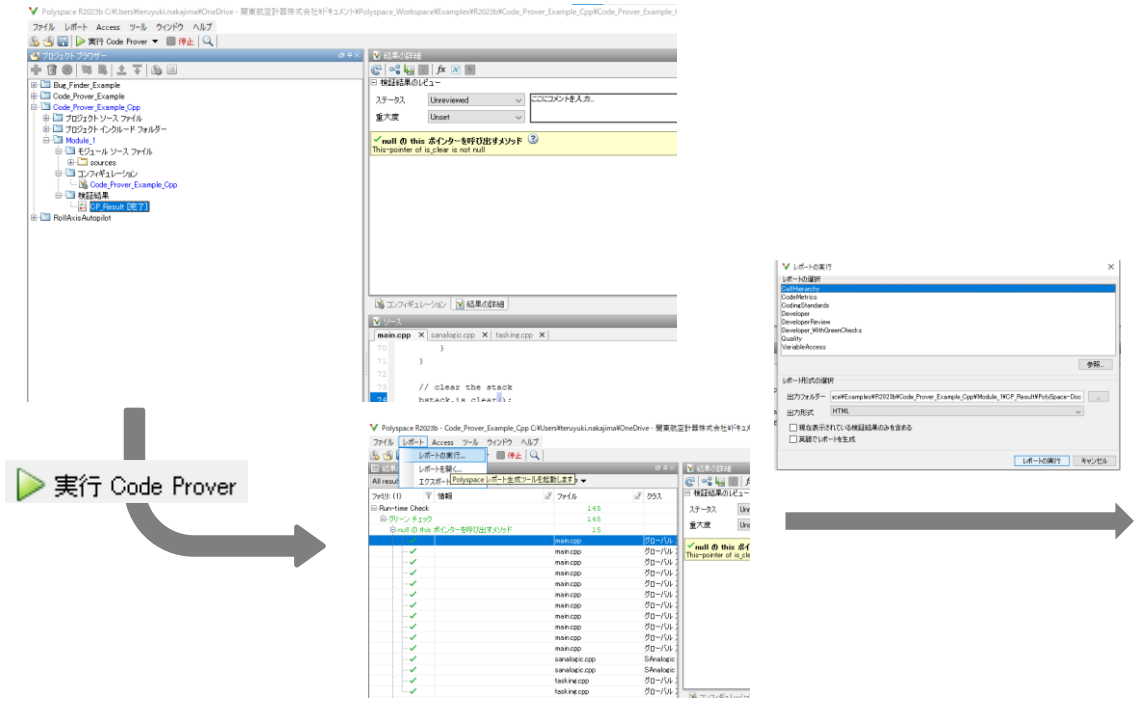
| | A | B | C | D | E | |
|---|---|---|---|---|---|---|
| 1 | モデル オブジェクト | 最適化されたモデル オブジェ | モデル オブジェクト パス | モデル オブジェクト サブシス | コード ファイルの場所 | コ |
| 2 | Phi | Yes | slcidemo_roll_orig | slcidemo_roll_orig | | |
| 3 | Psi | Yes | slcidemo_roll_orig | slcidemo_roll_orig | | |
| 4 | P | Yes | slcidemo_roll_orig | slcidemo_roll_orig | | |
| 5 | TAS | Yes | slcidemo_roll_orig | slcidemo_roll_orig | | |
| 6 | AP_Eng | Yes | slcidemo_roll_orig | slcidemo_roll_orig | | |
| 7 | HDG_Mode | Yes | slcidemo_roll_orig | slcidemo_roll_orig | | |
| 8 | HDG_Ref | Yes | slcidemo_roll_orig | slcidemo_roll_orig | | |
| 9 | Turn_Knob | Yes | slcidemo_roll_orig | slcidemo_roll_orig | | |
| 10 | BasicRollMode | | slcidemo_roll_orig | slcidemo_roll_orig | C:\user\MATLAB Examples\slcidemo_roll_orig_ert_rtw | slc |
| 11 | BasicRollMode | | slcidemo_roll_orig | slcidemo_roll_orig | C:\user\MATLAB Examples\slcidemo_roll_orig_ert_rtw | slc |
| 12 | BasicRollMode | | slcidemo_roll_orig | slcidemo_roll_orig | C:\user\MATLAB Examples\slcidemo_roll_orig_ert_rtw | slc |
| 13 | BasicRollMode | | slcidemo_roll_orig | slcidemo_roll_orig | C:\user\MATLAB Examples\slcidemo_roll_orig_ert_rtw | slc |

| | F | G | H | I |
|---|---|---|---|---|
| 1 | コード ファイル名 | コード関数 | コード行番号 | モデル オブジェクト タ |
| 2 | | | | Inport |
| 3 | | | | Inport |
| 4 | | | | Inport |
| 5 | | | | Inport |
| 6 | | | | Inport |
| 7 | | | | Inport |
| 8 | | | | Inport |
| 9 | | | | Inport |
| 10 | slcidemo_roll_orig.c | slcidemo_roll_orig_0_output | 111 | ModelReference |
| 11 | slcidemo_roll_orig.c | slcidemo_roll_orig_0_output | 112 | ModelReference |
| 12 | slcidemo_roll_orig.c | slcidemo_roll_orig_0_output | 113 | ModelReference |
| 13 | slcidemo_roll_orig.c | slcidemo_roll_orig_0_output | 31 | ModelReference |

# 1．手順 Table-A5 No.2

| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 2 | Source Code complies with software architecture. | MB.6.3.4.b | ソフトウェア・アーキテクチャへの準拠：<br>その目的は、ソース・コードがソフトウェア・アーキテクチャで定義されたデータ・フローと制御フローに一致していることを確認することである。 | Polyspace Code Prover<br>- 呼出し階層レポート<br>⇒ 関数の呼び出し関係ツリーによりモデルの構造とコードの構造が一致することを検証する。 |

## 画面操作




## 出力レポート：呼び出し階層レポート

**第1章 呼び出し階層**

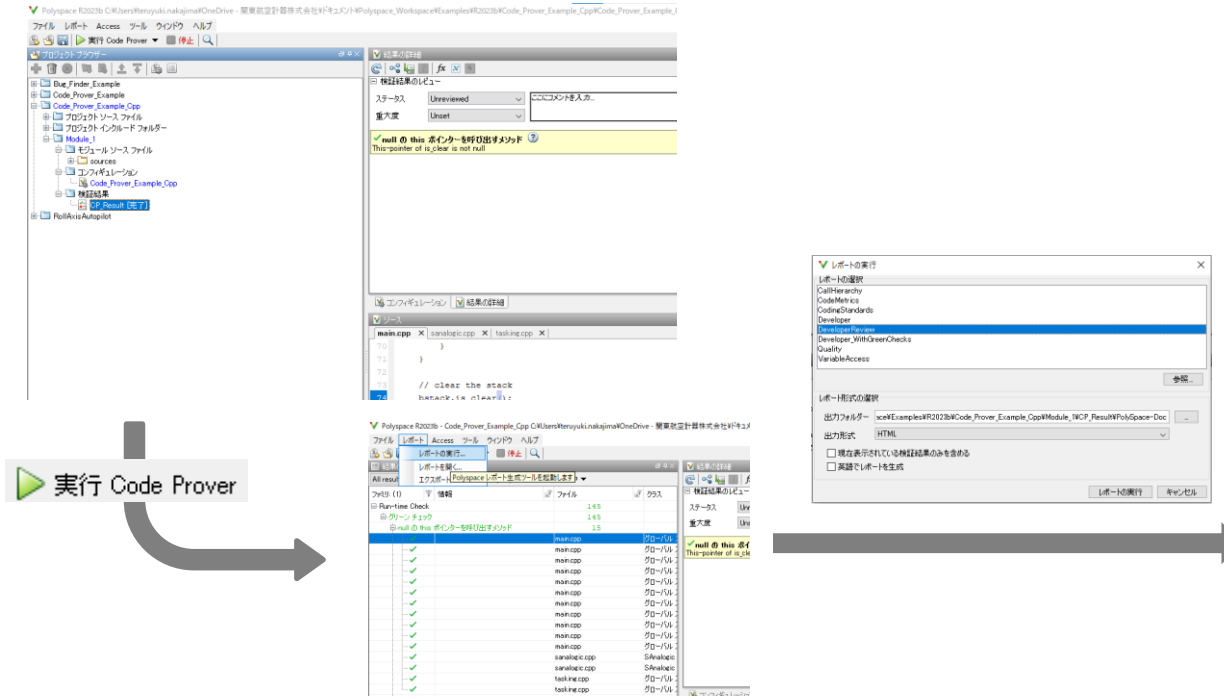
**プロジェクトの呼び出し階層レポート: Code_Prover_Example_Cpp - CP_Result**

| Call tree | File | Line | Col | Tasks | Definition file | Line | Col |
|---|---|---|---|---|---|---|---|
| __pstf1.main | | | | | | | |
| \| > tasking.proc1() | main.cpp | 82 | 4 | | tasking.cpp | 30 | 5 |
| \| > tasking.proc2() | main.cpp | 82 | 4 | | tasking.cpp | 34 | 5 |
| \| - > tasking.server1() | main.cpp | 82 | 4 | | tasking.cpp | 17 | 5 |
| \| \| > tasks.Task::Task() | tasking.cpp | 18 | 9 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 31 | 6 |
| \| \| - > tasks.Task::Tserver() | tasking.cpp | 19 | 6 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 68 | 11 |
| \| \| \| - > tasks.Task::Orderregulate() | tasks.cpp | 71 | 4 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 58 | 11 |
| \| \| \| \| > tasks.Task::Increase_PowerLevel() | tasks.cpp | 60 | 4 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 36 | 11 |
| \| \| \| > tasking.Begin_CS() | tasks.cpp | 74 | 8 | [__pstf3.server2(),__pstf3.server1()] | tasking.cpp | 46 | 5 |
| \| \| \| > tasking.End_CS() | tasks.cpp | 76 | 8 | [__pstf3.server2(),__pstf3.server1()] | tasking.cpp | 48 | 5 |
| \| \| \| - > tasks.Task::Exec_One_Cycle(int) | tasks.cpp | 77 | 8 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 52 | 11 |
| \| \| \| \| - > tasks.Task::Drive_Balance(int) | tasks.cpp | 53 | 4 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 99 | 11 |
| \| \| \| \| \| - > tasks.Task::Command_Ordering(int) | tasks.cpp | 100 | 4 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 93 | 11 |
| \| \| \| \| \| \| - > tasks.Task::Orderregulate() | tasks.cpp | 96 | 4 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 58 | 11 |
| Already displayed above | | | | | | | |
| \| \| \| \| - > tasks.Task::Scheduler(int) | tasks.cpp | 54 | 4 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 103 | 11 |
| \| \| \| \| - > tasks.Task::Command_Ordering(int) | tasks.cpp | 104 | 4 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 93 | 11 |
| Already displayed above | | | | | | | |
| \| \| > tasks.Task::~Task() | tasking.cpp | 20 | 0 | [__pstf3.server2(),__pstf3.server1()] | tasks.h | 17 | 4 |
| \| - > tasking.server2() | main.cpp | 82 | 4 | | tasking.cpp | 22 | 5 |
| \| \| > pst_stubs.operator new(unsigned long) | tasking.cpp | 23 | 14 | [__pstf3.server2()] | pst_stubs.cpp | 20 | 0 |
| \| \| > tasks.Task::Task() | tasking.cpp | 23 | 22 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 31 | 6 |
| \| \| - > tasks.Task::Tserver() | tasking.cpp | 25 | 11 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 68 | 11 |
| Already displayed above | | | | | | | |
| \| \| > tasks.Task::~Task() | tasking.cpp | 26 | 8 | [__pstf3.server2(),__pstf3.server1()] | tasks.h | 17 | 4 |
| \| \| > pst_stubs.operator delete(void*, unsigned long) | tasking.cpp | 26 | 8 | [__pstf3.server2()] | pst_stubs.cpp | 27 | 0 |
| \| - > tasks._dynamic_init_globals | main.cpp | 82 | 11 | | tasks.cpp | 0 | 0 |
| \| \| > tasks.Task::Task() | tasks.cpp | 20 | 5 | [__pstf3.server2(),__pstf3.server1()] | tasks.cpp | 31 | 6 |
| \| > pst_stubs.already_init() | main.cpp | 85 | 8 | | pst_stubs.cpp | 32 | 0 |
| \| - > main.table_loop() | main.cpp | 66 | 17 | | main.cpp | 21 | 12 |
| \| \| > pst_stubs.operator new(unsigned long) | main.cpp | 25 | 21 | [__pstf3.server2()] | pst_stubs.cpp | 20 | 0 |
| \| \| - > sanalogic.SAnalogic::SAnalogic() | main.cpp | 25 | 25 | | sanalogic.h | 14 | 4 |
| \| \| \| - > sensor.Sensor::Sensor() | sanalogic.h | 14 | 4 | | sensor.h | 16 | 4 |
| \| \| \| \| > zz_base.Base::Base() | sensor.h | 16 | 4 | | zz_base.h | 12 | 6 |
| \| \| > pst_stubs.operator new(unsigned long) | main.cpp | 25 | 36 | [__pstf3.server2()] | pst_stubs.cpp | 20 | 0 |
| \| \| - > sensor.Sensor::Sensor() | main.cpp | 25 | 40 | | sensor.h | 16 | 4 |
| Already displayed above | | | | | | | |
| \| \| > pst_stubs.operator new(unsigned long) | main.cpp | 25 | 48 | [__pstf3.server2()] | pst_stubs.cpp | 20 | 0 |
| \| \| - > sensor.Sensor::Sensor() | main.cpp | 25 | 52 | | sensor.h | 16 | 4 |

# 1．手順 Table-A5 No.3

| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 3 | Source Code is verifiable. | MB.6.3.4.c | 検証可能性：<br>このobjectiveは、ソースコードに検証できないステートメントや構造が含まれていないこと、また、テストするためにコードを変更する必要がないことを保証することである。 | **Polyspace Code Prover**<br>**- 開発者レポート**<br>**⇒ 到達不能なコードがないこと検証を行う。**<br>Simulink Code Inspector<br>- Check Compatibility<br>⇒ Simulink Code Inspectorで対応しているブロックセットのみの使用を確認し、Simulink Code Inspectorでの検証可能性を確認する。 |

## 画面操作




## 出力レポート：開発者レポート

**Polyspace コード検証**

**プロジェクトの開発者レポート: Code_Prover_Example_Cpp**

目次

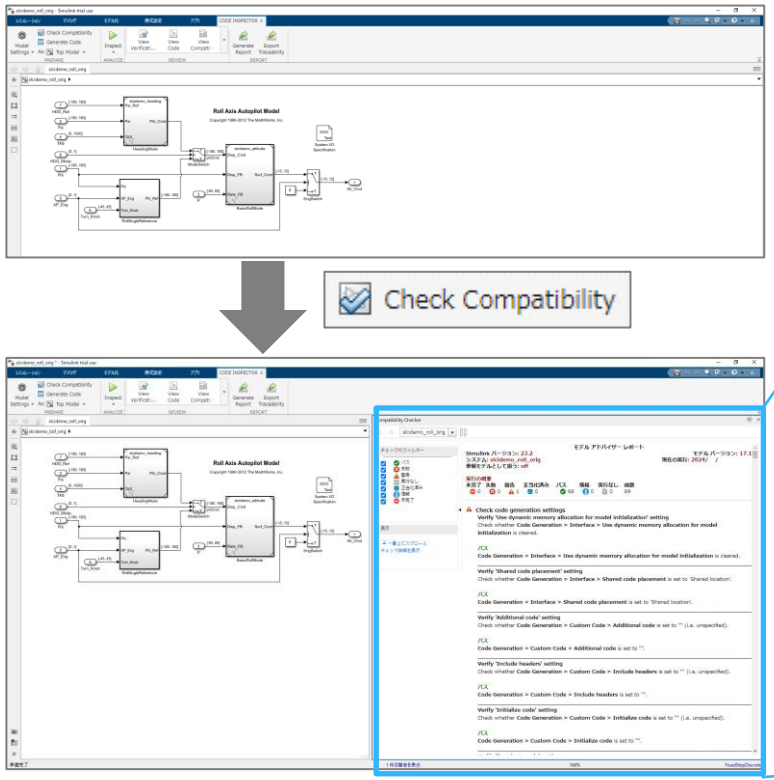


| ID | チェック | 関数 | 詳細 | 正当化済み | コメント |
|---|---|---|---|---|---|
| 242 | 到達不能コード | TStack<Base*>::pop() | The section of code is unreachable or the condition is redundant.<br>If-condition always evaluates to false at line 37 (column 12).<br>Block ends at line 39 (column 8) | No | |
| 247 | 不適切にデリファレンスされたポインター | TStack<Base*>::pop() | Warning: The pointed memory location might not be suitable for dereference | No | |
| 130 | 未初期化ポインター | TStack<Base*>::pop() | Warning: pointer may be non-initialized | No | |
| 133 | 未初期化ポインター | TStack<Base*>::pop() | Warning: pointer may be non-initialized | No | |
| 87 | 未初期化ポインター | TStackIterator<Base*>::operator++() | Warning: pointer may be non-initialized | No | |
| 237 | 到達不能コード | TStackIterator<Base*>::current() const | The section of code is unreachable or the condition is redundant.<br>If-condition always evaluates to false at line 149 (column 12).<br>Block ends at line 151 (column 8) | No | |
| | 不適切にデリファレンス | TStackIterator<Base*>::current() | Warning: The pointed | | |

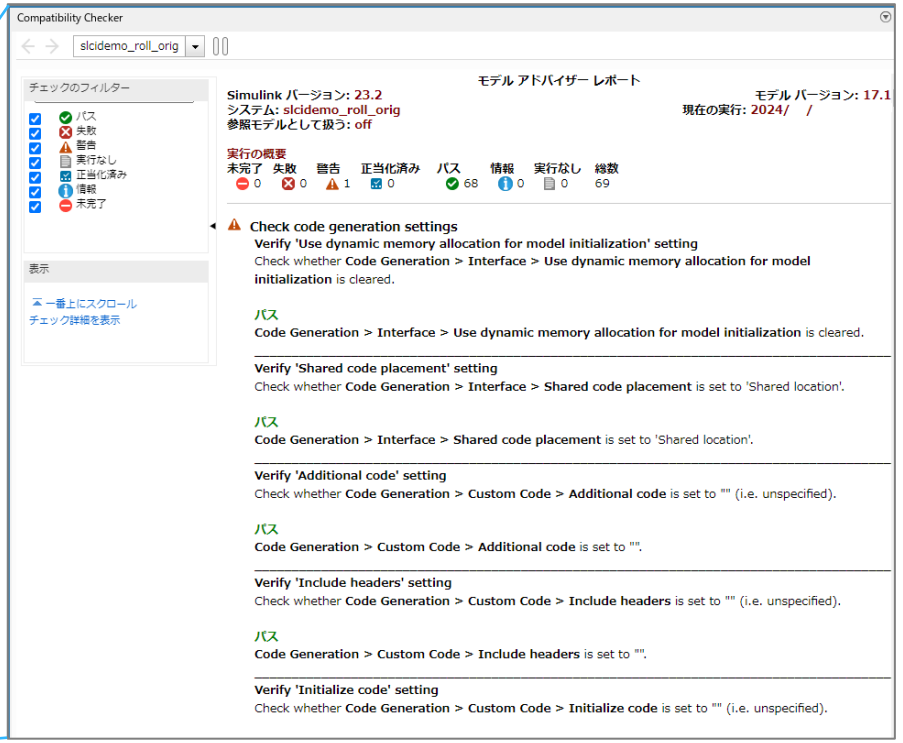
# 1．手順 Table-A5 No.3

| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 3 | Source Code is verifiable. | MB.6.3.4.c | 検証可能性：<br>このobjectiveは、ソースコードに検証できないステートメントや構造が含まれていないこと、また、テストするためにコードを変更する必要がないことを保証することである。 | Polyspace Code Prover<br>- 開発者レポート<br>⇒ 到達不能なコードがないこと検証を行う。<br>**Simulink Code Inspector**<br>**- Check Compatibility**<br>**⇒ Simulink Code Inspectorで対応しているブロックセットのみの使用を確認し、Simulink Code Inspectorでの検証可能性を確認する。** |

画面操作



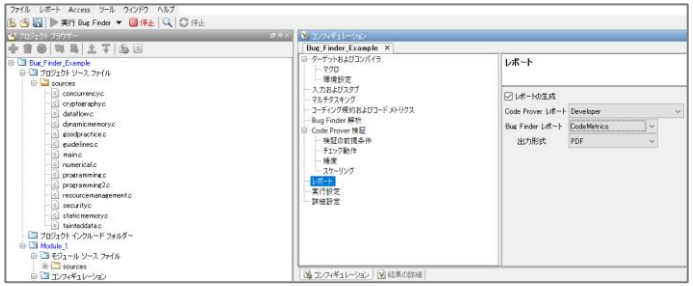

出力レポート：モデルアドバイザーレポート

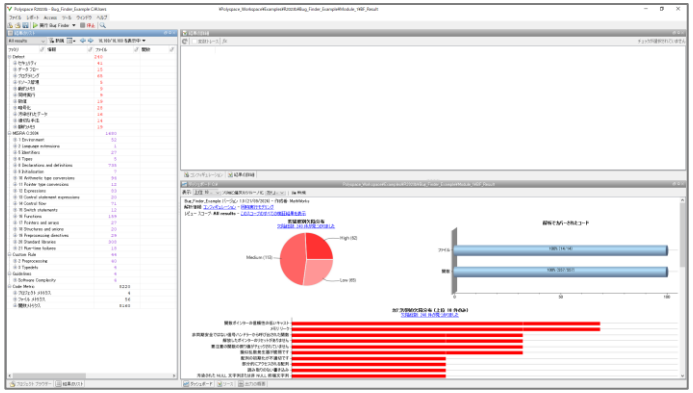


# 1．手順 Table-A5 No.4

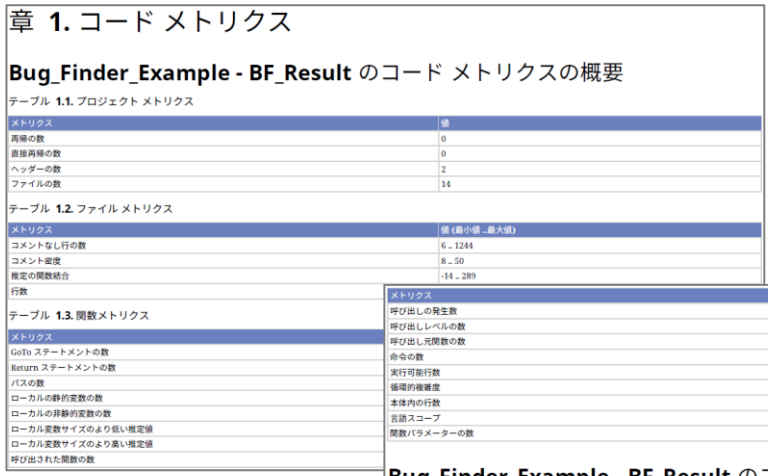
| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 4 | Source Code conforms to standards. | MB.6.3.4.d | 標準への適合：<br>複雑さの制限やコードの制約などの検証である。複雑さには、ソフトウェアコンポーネント間の結合度、制御構造の入れ子レベル、論理式や数値式の複雑さなどが含まれる。この分析により、標準からの逸脱が正当化されることも保証される。 | Polyspace Bug Finder<br>**-コードメトリクスレポート**<br>**⇒ コードの複雑度の検証を行う。**<br>-コーディング規約レポート<br>⇒ コーディング規約への準拠の検証を行う。 |

## 画面操作



## 出力レポート：コードメトリクスレポート



章 1. コード メトリクス

**Bug_Finder_Example - BF_Result** のコード メトリクスの概要

テーブル 1.1. プロジェクト メトリクス

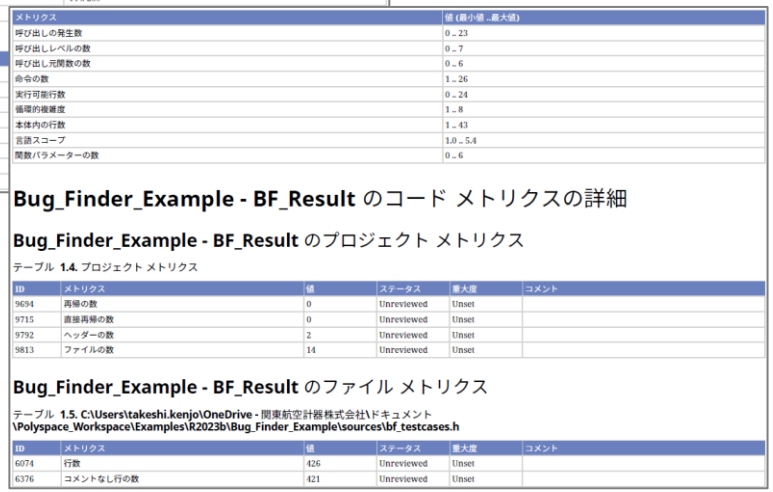
| メトリクス | 値 |
|---|---|
| 再帰の数 | 0 |
| 直接再帰の数 | 0 |
| ヘッダーの数 | 2 |
| ファイルの数 | 14 |

テーブル 1.2. ファイル メトリクス

| メトリクス | 値 (最小値 .. 最大値) |
|---|---|
| コメントなし行の数 | 6 .. 1244 |
| コメント密度 | 8 .. 50 |
| 推定の関数結合 | -14 .. 289 |
| 行数 | |

テーブル 1.3. 関数メトリクス

| メトリクス |
|---|
| GoTo ステートメントの数 |
| Return ステートメントの数 |
| パスの数 |
| ローカルの静的変数の数 |
| ローカルの非静的変数の数 |
| ローカル変数サイズのより低い推定値 |
| ローカル変数サイズのより高い推定値 |
| 呼び出された関数の数 |



| メトリクス | 値 (最小値 .. 最大値) |
|---|---|
| 呼び出しの発生数 | 0 .. 23 |
| 呼び出しレベルの数 | 0 .. 7 |
| 呼び出し元関数の数 | 0 .. 6 |
| 命令の数 | 1 .. 26 |
| 実行可能行数 | 0 .. 24 |
| 循環的複雑度 | 1 .. 8 |
| 本体内の行数 | 1 .. 43 |
| 言語スコープ | 1.0 .. 5.4 |
| 関数パラメーターの数 | 0 .. 6 |

**Bug_Finder_Example - BF_Result** のコード メトリクスの詳細

**Bug_Finder_Example - BF_Result** のプロジェクト メトリクス

テーブル 1.4. プロジェクト メトリクス

| ID | メトリクス | 値 | ステータス | 重大度 | コメント |
|---|---|---|---|---|---|
| 9694 | 再帰の数 | 0 | Unreviewed | Unset | |
| 9715 | 直接再帰の数 | 0 | Unreviewed | Unset | |
| 9792 | ヘッダーの数 | 2 | Unreviewed | Unset | |
| 9813 | ファイルの数 | 14 | Unreviewed | Unset | |

**Bug_Finder_Example - BF_Result** のファイル メトリクス

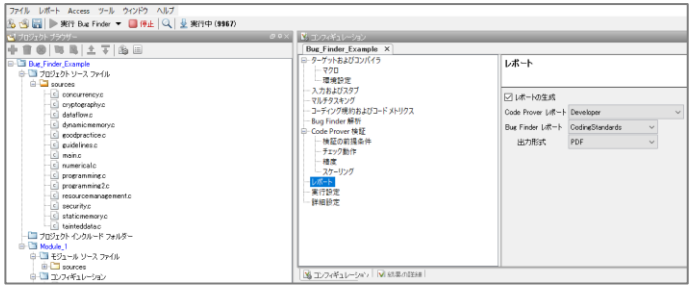
テーブル 1.5. C:\Users\takeshi.kenjo\OneDrive - 関東航空計器株式会社\ドキュメント\Polyspace_Workspace\Examples\R2023b\Bug_Finder_Example\sources\bf_testcases.h

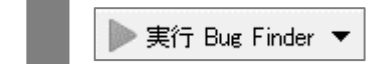
| ID | メトリクス | 値 | ステータス | 重大度 | コメント |
|---|---|---|---|---|---|
| 6074 | 行数 | 426 | Unreviewed | Unset | |
| 6376 | コメントなし行の数 | 421 | Unreviewed | Unset | |

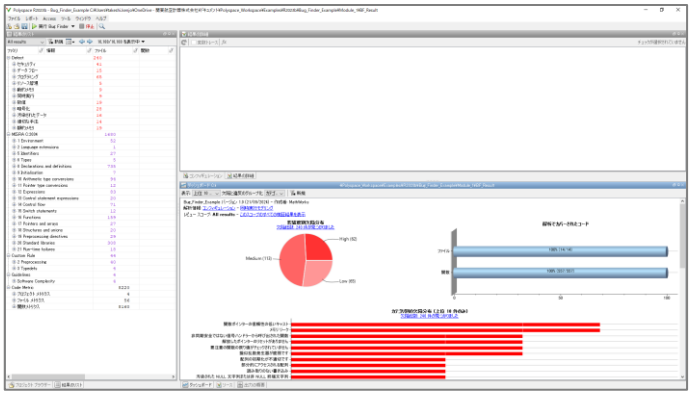
# 1．手順 Table-A5 No.4

| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 4 | Source Code conforms to standards. | MB.6.3.4.d | 標準への適合：<br>複雑さの制限やコードの制約などの検証である。複雑さには、ソフトウェアコンポーネント間の結合度、制御構造の入れ子レベル、論理式や数値式の複雑さなどが含まれる。この分析により、標準からの逸脱が正当化されることも保証される。 | Polyspace Bug Finder<br>-コードメトリクスレポート<br>⇒ コードの複雑度の検証を行う。<br>**-コーディング規約レポート**<br>**⇒ コーディング規約への準拠の検証を行う。** |

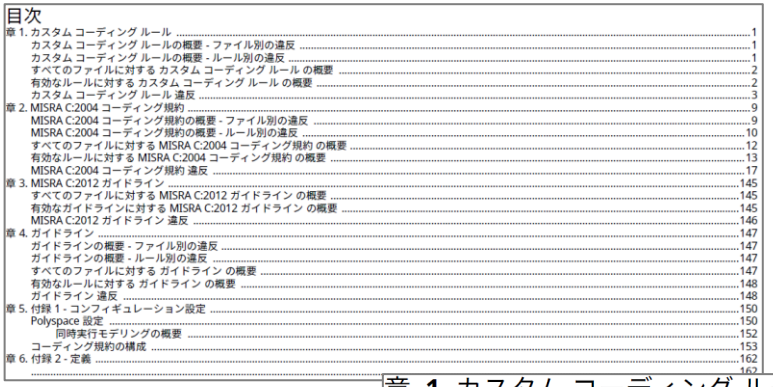
## 画面操作



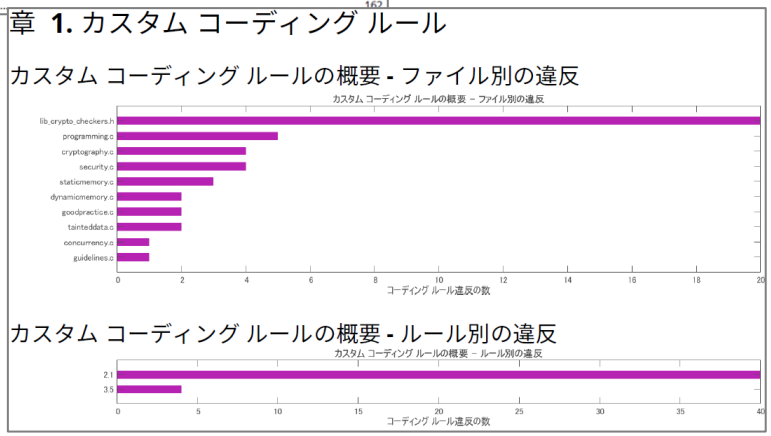


## 出力レポート：コーディング規約レポート


目次





# 1．手順 Table-A5 No.5

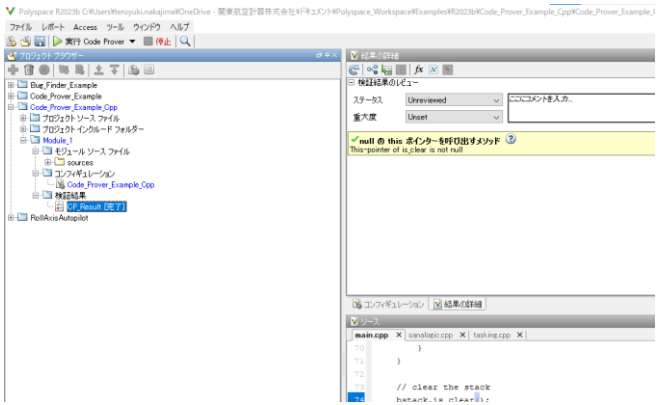
| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 5 | Source Code is traceable to lowlevel requirements. | MB.6.3.4.e | トレーサビリティ： このobjectiveは、低レベルの要求がソースコードに開発されたことを確認することである。 | Simulink Code Inspector<br>- Simulink Code Inspector Report<br>- trace report<br>⇒ ソフトウエア要求(モデル)がコードにもれなく対応しているかを検証する。 |

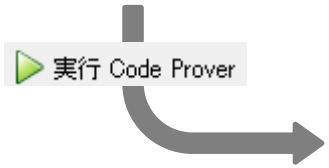
手順 No.1-1/-2と同じ

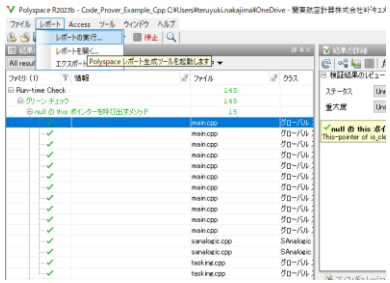
# 3．手順 Table-A5 No.6

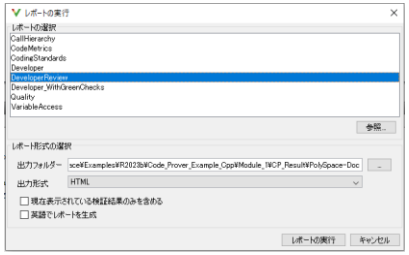
| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 6 | Source Code is accurate and consistent. | MB.6.3.4.f | 正確さと一貫性：スタック使用量、メモリ使用量、固定小数点演算のオーバーフローと解決、浮動小数点演算、リソース競合と制限、ワーストケース実行タイミング、例外処理、未初期化変数の使用、キャッシュ管理、未使用変数、タスクや割り込み競合によるデータ破損など、ソースコードの正確さと一貫性を判断することが目的です。コンパイラ（そのオプションも含む）、リンカ（そのオプションも含む）、およびいくつかのハードウェア機能は、ワーストケース実行タイミングに影響を与える可能性があり、この影響を評価する必要があります。 | **Polyspace Code Prover**<br>**- 開発者レポート**<br>**⇒ ランタイムエラー数、スタック使用量の確認を行う。**<br>**⇒ 到達不能なコードがないこと検証を行う。**<br><br>Polyspace Code Prover<br>- 品質レポート<br>⇒ ランタイムエラーがないことの検証を行う。 |

## 画面操作





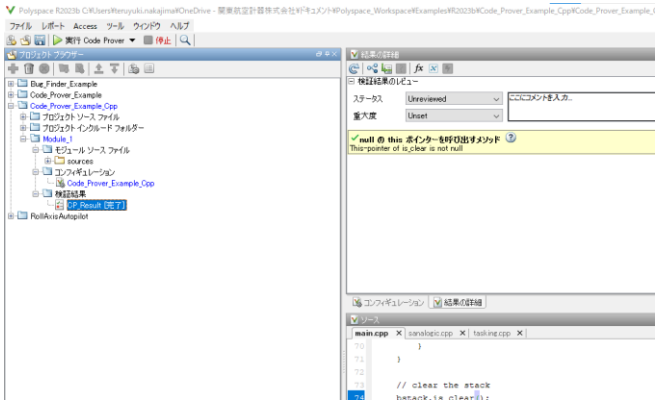


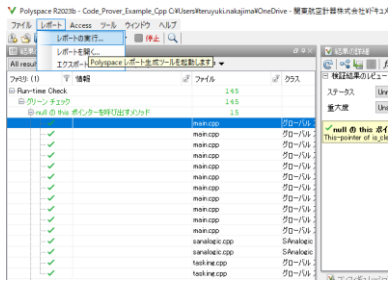
## 出力レポート：開発者レポート

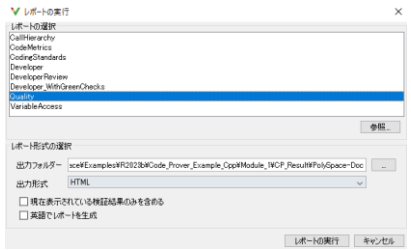
# 3．手順 Table-A5 No.6

| No. | Objective | | | 対応ツール |
|---|---|---|---|---|
| 6 | Source Code is accurate and consistent. | MB.6.3.4.f | 正確さと一貫性：スタック使用量、メモリ使用量、固定小数点演算のオーバーフローと解決、浮動小数点演算、リソース競合と制限、ワーストケース実行タイミング、例外処理、未初期化変数の使用、キャッシュ管理、未使用変数、タスクや割り込み競合によるデータ破損など、ソースコードの正確さと一貫性を判断することが目的です。コンパイラ（そのオプションも含む）、リンカ（そのオプションも含む）、およびいくつかのハードウェア機能は、ワーストケース実行タイミングに影響を与える可能性があり、この影響を評価する必要があります。 | Polyspace Code Prover<br>- 開発者レポート<br>⇒ ランタイムエラー数、スタック使用量の確認を行う。<br>⇒ 到達不能なコードがないこと検証を行う。<br><br>**Polyspace Code Prover**<br>**- 品質レポート**<br>**⇒ ランタイムエラーがないことの検証を行う。** |

画面操作





出力レポート：品質レポート

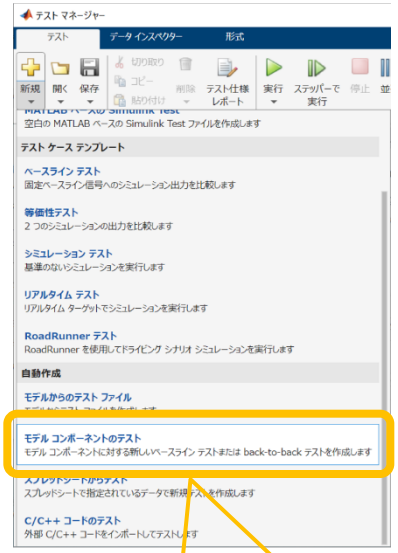
**Appendix B PIL 評価手順**

DO-331 Appendix Table A-6 および A-7 を達成する上で有効な手段となる PIL の評価手順を以下に示す。

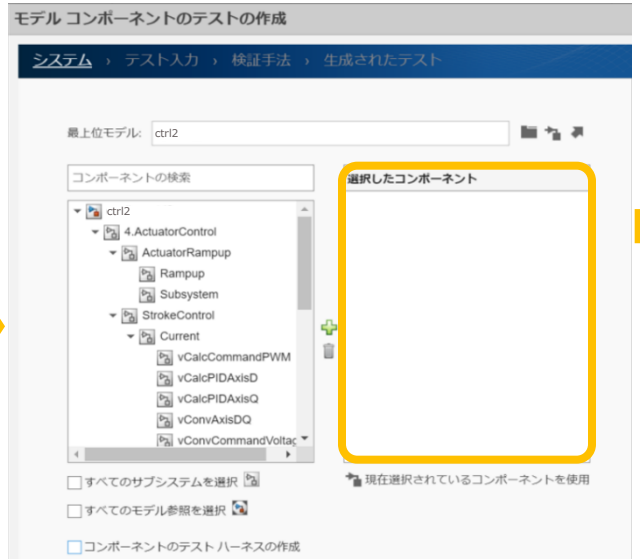
対応ツール： Simulink Test, Embedded Coder

# PIL実施手順1: 等価性確認テストケースの作成

・テストマネージャーからback-to-back（等価性確認）テスト作成メニューを選択

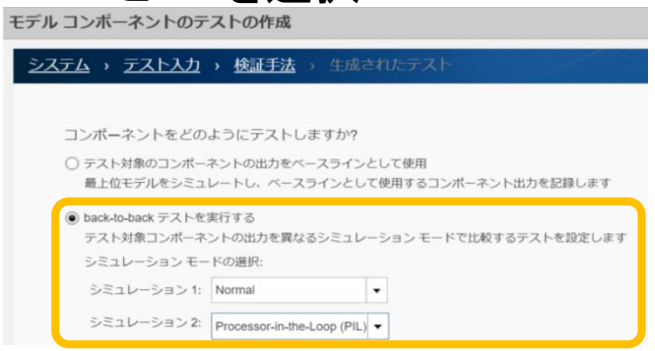


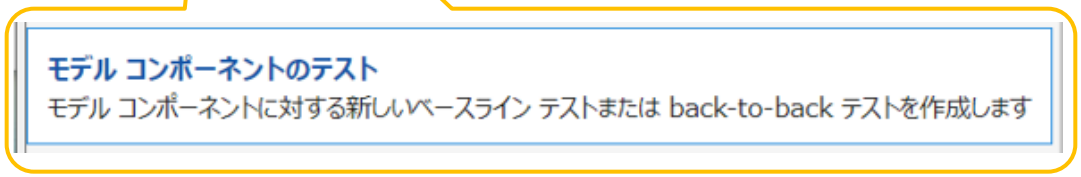

・テスト対象のコンポーネントを選択（最上位モデルは選択なし）




・テストの入力方法を選択

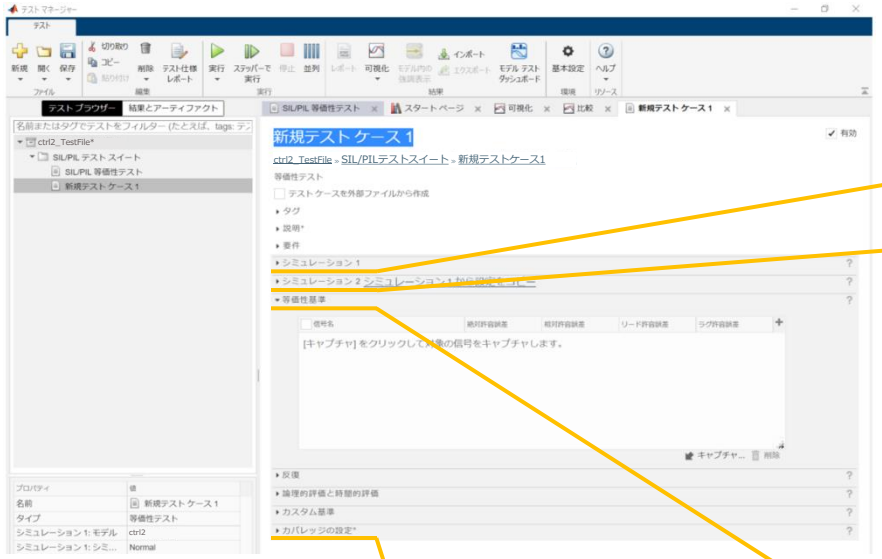

・シミュレーションモードに
MILとPILを選択

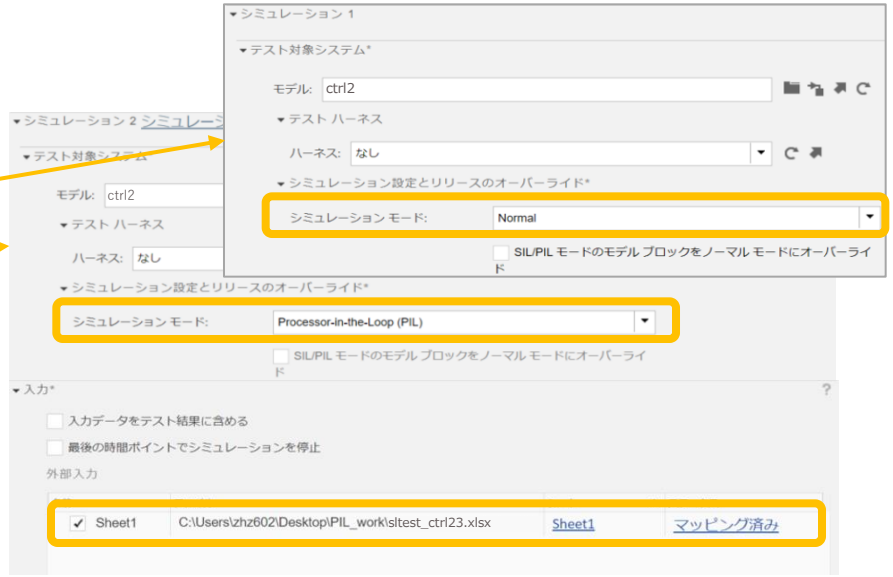


# PIL実施手順2: 等価性確認テストケースの設定

・自動作成されたテスト内容の確認（シミュレーションモード、入力など）



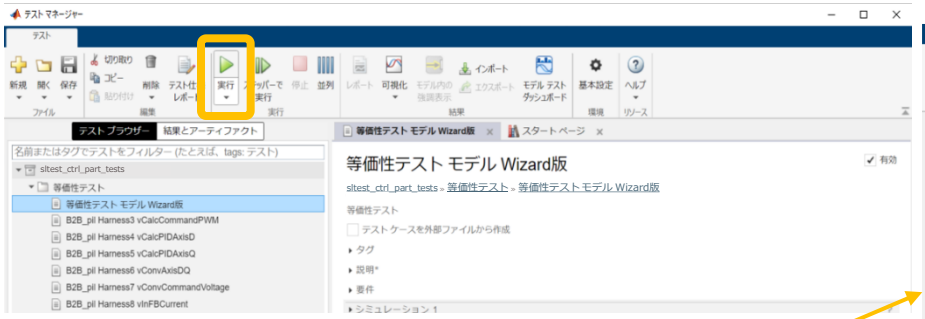




・カバレッジ取得設定

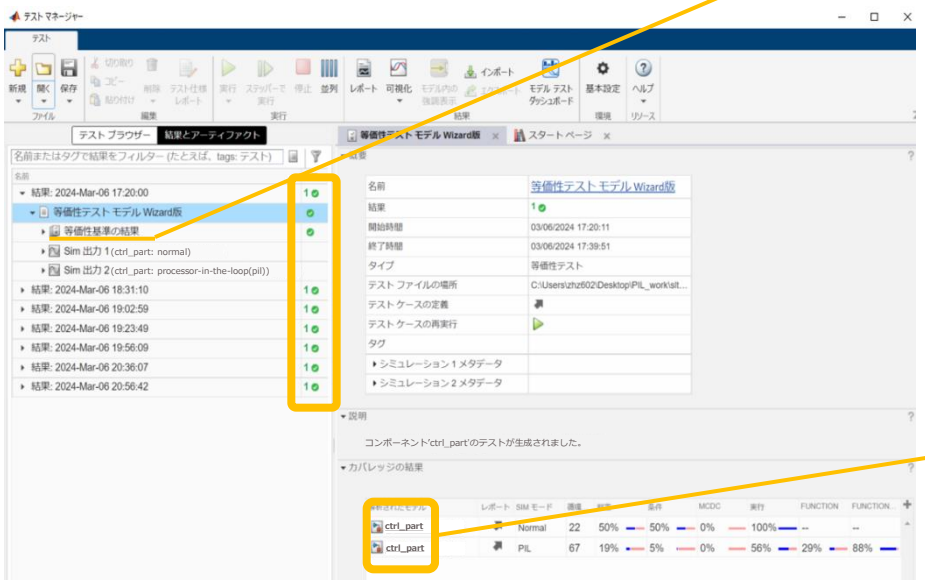

・等価性基準の設定（キャプチャ後に入力）

# PIL実施手順3: 等価性確認テストケース実施と結果確認

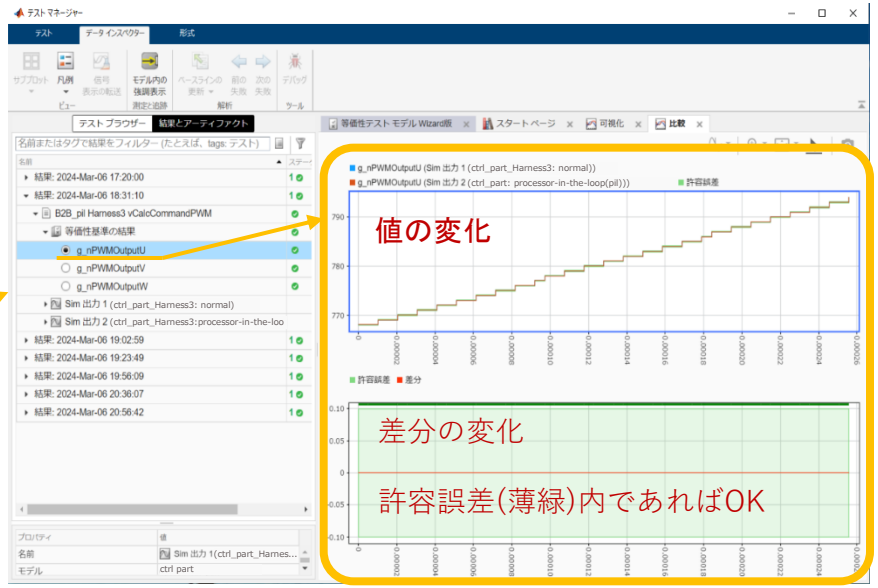
・「実行」ボタンでテスト実施




・等価性基準判定結果（緑のチェックは評価OK）




・各項目の等価性確認内容




・カバレッジ詳細内容