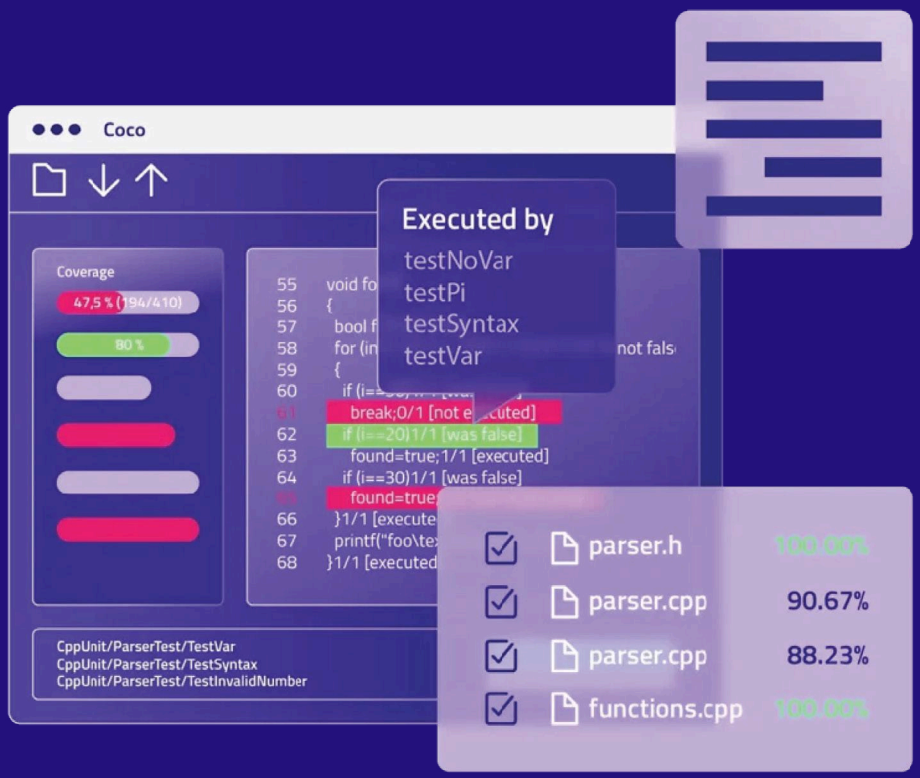
Qt Group



Coco
コードカバレッジ分析

テストの有効性を「推測」から「実測」へ

C、C++、C#、QML、Tcl、Python 対応
デスクトップからセーフティークリティカルな
組込みシステムまで

SGS TÜV SAAR

コードカバレッジの計測が可能にすること

テストスイートがすべてパスするのは安心材料です。しかし、「テストに通ること」と「十分にテストされていること」は同義ではありません。

テストは成功していても、ロジックの分岐全体が未実行のままだったり、エッジケースが検証されていなかったり、エラーハンドリングの経路が一度も通らないことがあります。カバレッジを計測しなければ、テスト中に実際にどの行・どの判断・どの条件が実行され、どれが見過ごされたのかを信頼して把握することはできません。

想定上のカバレッジと実測カバレッジのギャップこそが、本番不具合が潜む場所です。コードカバレッジはそれを可視化し、測定可能にし、対処可能にします。

Cocoでコードカバレッジを測定することで、チームは次のメリットを得られます：

- **実行の完全な可視化** — テストが通ったかどうかだけでなく、どのステートメント・分岐・条件がテスト中に実行されたかを正確に把握できます
- **本番品質への信頼性向上** — カバレッジの抜け漏れを顧客に届く前に検出し、現場での不具合発見に頼る必要がなくなります
- **的確な再テスト** — コード変更が影響するテストを特定し、広範囲ではなくピンポイントで再テストを実施できます
- **監査対応可能なデータ** — 判定（decision）、条件（condition）、およびMC/DC基準を含むカバレッジレポートを標準で提供し、レビューや認証に追加作業なしで対応できます
- **フルスタックカバレッジ** — C、C++、Pythonの各レイヤーを横断して測定し、部分的なツールでは埋めきれないギャップを解消します

コードカバレッジの計測は、テストの実行状況を客観的に把握する手段です。テストされていない挙動を可視化し、カバレッジのギャップをより早期に検出し、リリース判断への確信を高めます。その結果、ソフトウェアのリスクをより明確に把握でき、テストの有効性をより正確に理解できます。規制のある環境では、検証目標が達成されたことを証明するために、こうした計測に基づくエビデンスが求められます。

一般的なアプローチが抱える限界

多くのチームは、カバレッジデータは存在するものの、意思決定・最適化・認証に使えるほどの信頼性がない、という状況に直面しています。

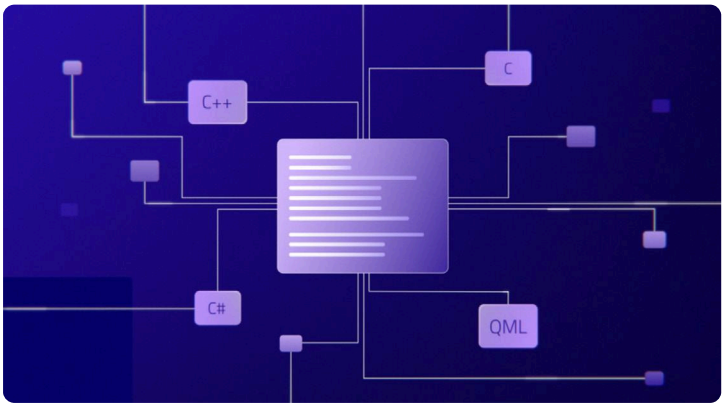
よく使われるカバレッジの手法の多くは、得られる情報が限定的・断片的です。

- コンパイラ生成のカバレッジは、基本的なメトリクスにとどまることが多く、組込みやクロスコンパイル環境向けには設計されていません
- ユニットテストのカバレッジだけでは、特に組込み環境において、システムレベルの挙動やハードウェア依存の動作を捉えきれないことがよくあります
- 手動レビューとトレーサビリティは有益なコンテキストを提供しますが、客観的なランタイム実行データの代替にはなりません

Cocoは、既存のワークフローを置き換えたりテストを書き換えたりすることなく、言語・プラットフォーム・実機ターゲットを横断して動作するコンパイラレベルのインストルメンテーションによって、これらのギャップを解消します。

Coco が提供するもの

Cocoは、デスクトップおよび組込みシステムの両方を開発するソフトウェアチーム向けに設計された、最新のコードカバレッジツールです。C、C++、C#、QML、Tcl、そしてPython（coverage.pyとの連携）をサポートし、コンパイラレベルのインストルメンテーションを用いて、ステートメント、分岐、条件、MC/DCを含む詳細な実行データを収集します。これにより、幅広いプラットフォームにわたって包括的な可視化を実現します。



このような課題があるとき	Coco でできること
未テストのロジックへの可視性が不足している	インストルメンテーションとカバレッジメトリクス により、未テストのロジック・コード・未実行のステートメント／条件を特定できます
動的テストの効率が低い	テスト中に実行されたステートメント・分岐・条件を確認し、ユニットテスト・機能テスト・手動テストの改善と最適化に活かせます。インストルメンテーションが存在する場合、手動実行のカバレッジも分析できます
対応コンパイラ・プラットフォームを横断した一貫したカバレッジインストルメンテーションが必要	C、C++、C#、QML、Tclのカバレッジをデスクトップ・組込み・実機ハードウェアターゲット向けに生成し、他の多くのツールが抱えるフラグメンテーションの限界を克服します
開発継続中に未テストコードが増え続けている	パッチ解析を使って変更されたコードに必要なテストを特定し、コード変更の影響を受けるテストだけを選択して実行することで、全テストの再実行を回避できます
リリースサイクルのリスクが高い	パッチベースのテスト推奨により、リリース準備をより確実にコントロール。特にセーフティクリティカルな環境での品質保証に役立ちます
システムの挙動が不透明	Cocoのインストルメンテーションで実際のランタイム実行パスを可視化し、内部の挙動を観測可能にします
監査でテストの完全性を証明することが難しい	ISO 26262、DO-178C、DO-330、IEC 61508、IEC 62304に対応した、監査対応・標準準拠のレポートを生成できます

Cocoがカバレッジ結果を生成する仕組み

成果：環境・時期を問わず、一貫したカバレッジ結果が得られます。

Cocoはビルドプロセス中にソフトウェアをインストルメント化し、実行ファイルの構造的な記述を生成します。テスト実行中は、どのステートメント・判断・条件が実行されたかをランタイムデータとして記録します。

- 複数の実行・構成・ターゲットから得られたカバレッジデータは、まとめてマージし、統合的に分析できます。

見えないものは、管理できない

Cocoは、テスト中に実際に実行されたコードと、実行されなかったコードを可視化します。

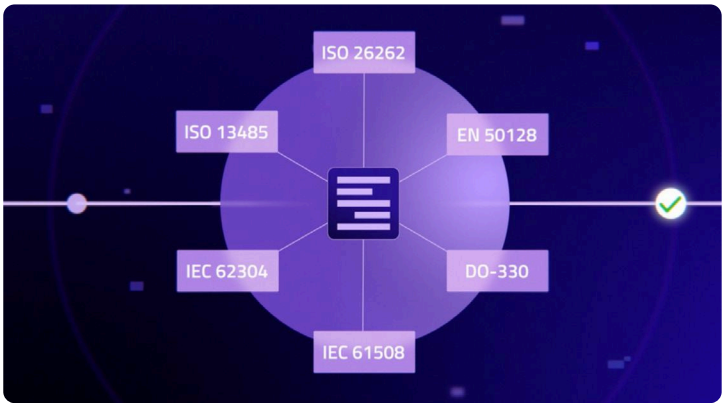
ご興味をお持ちの方は、お気軽にお問い合わせください。
無料トライアルで実プロジェクトをお試しいただけます。

対応する認証・規制要件

成果：チームは、認証要件に対応したトレーサブルなカバレッジエビデンスを生成し、監査・アセスメント・長期ライフサイクルのコンプライアンス活動に備えられます。

Cocoは、ステートメント・判断・条件・Modified Condition/Decision Coverage（MC/DC）など、規制標準で参照・要求される構造的カバレッジ基準に対応しています。カバレッジ結果は実際のランタイム実行から導出され、複数の実行・構成・ターゲットにわたって統合できます。

信頼性とトレーサビリティが求められる環境向けに設計されたCocoは、ISO 26262、DO-330、DO-178C、IEC 61508、IEC 62304といった主要な国際的安全・コンプライアンス標準に準拠した、監査対応のカバレッジレポートを生成します。自動車・航空宇宙・産業・医療分野のソフトウェア開発に適しています。



Cocoは TÜV (Technischer Überwachungsverein) による審査を受けています。TÜVは、認められた安全・品質標準に基づいてツールを評価する、独立した認証機関です。この資格認定により、チームのツール資格認定にかかる負担を直接軽減できます。CocoはISO 26262のASIL-D、およびDO-178CとIEC 62304の同等レベルを対象とするプロジェクトへの使用について、事前検証済みです。資格認定エビデンスをゼロから構築する必要はありません。

規制産業向けツール資格認定

Coco の資格認定キットは、テストスイートの構築や正当性の文書化をゼロから行うことなく、セーフティクリティカルな開発においてCocoが安全かつ信頼性の高いツールであることを証明するために必要なものをすべて提供します。

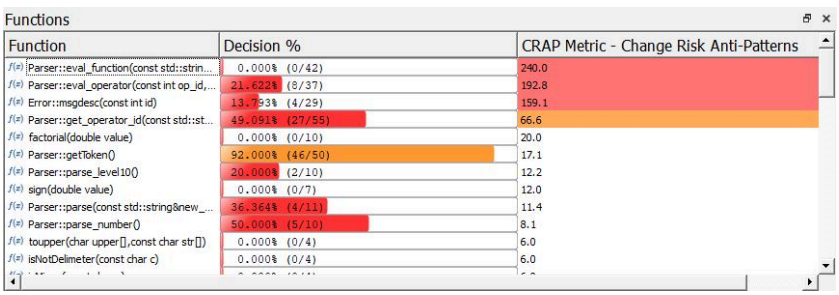
資格認定キットは安全規格ごとに個別に提供されており、ISO 26262、DO-178C、IEC 62304、EN 50128に対応しています。特定の安全規格のもとで開発するチームが、各認証コンテキストに関連したターゲットを絞ったドキュメントと事前構築済みテストケースを受け取れるようになっています。

これらのキットは、適用される安全規格に従い、定義された開発・検証コンテキスト内でCocoを使用するためのドキュメント・資格認定資料・ガイダンスを提供します。

CRAPメトリクス

カバレッジのギャップをリスクランキングに変える

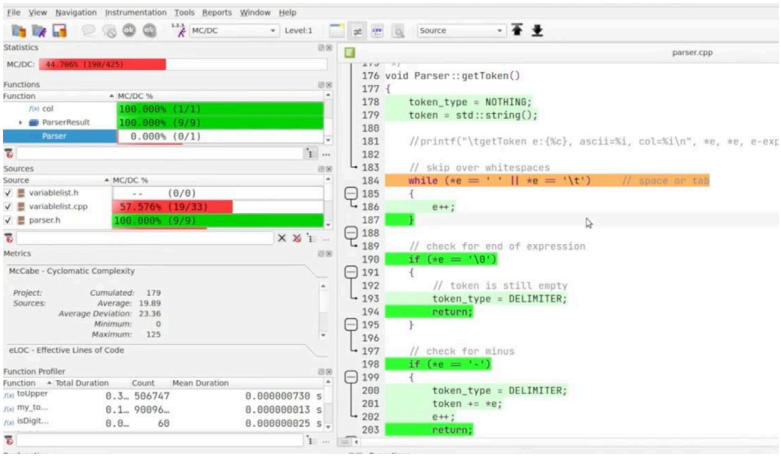
カバレッジが80%あるとわかっていても、未テストの20%が些細なものなのか、重大なものなのかはわかりません。CocoのCRAP（Change Risk Anti-Patterns）メトリクスはそれを明らかにします。複雑度の分析とカバレッジデータを組み合わせて、すべての関数にリスクスコアを算出します。どこに注目すべきか推測するのではなく、変更時に問題を引き起こす可能性が最も高い関数のランキングリストをもとに作業できます。複雑度が高く、カバレッジが低いほど、スコアは高くなります。スコアが30を超える関数はCoverageBrowserで高リスクとしてフラグが立てられ、リストの上位に表示されます。エンジニアと品質担当者は、コードベース全体を見直すことなく、新しいテストがどこで最もリスクを軽減できるかを正確に把握できます。結果はレポートおよびレビュー用にエクスポートできます。



Cocoがもたらす成果

実際の実行挙動を計測し、意味のあるカバレッジ基準を適用することで、Cocoはチームのビルドやテストの方法を変えることなく、品質向上・リスク低減・コンプライアンス対応の信頼できる基盤を提供します。

- 未テストおよび到達不能なコードの早期特定
- CRAPスコアによる関数の客観的なリスクランキング — 新しいテストをどこに書くべきかという勘頼りの判断を、複雑度が高くテストが不足している関数の明確な把握に置き換える
- コード変更後のより的を絞ったテスト
- カバレッジへの確信を損なわずにテスト実行を削減
- 組込みおよびリアルタイムの挙動を反映したカバレッジデータ
- 機能安全標準に沿ったカバレッジエビデンス
- 未実行コードのレビュー
- テストエビデンスの長期アーカイブ用HTMLエクスポート



技術概要

```
75 /*
76  * Get value of variable with given name
77  */
78 bool Variablelist::get_value(const char* name,
79                              double* value)
80 {
81     int id = get_id(name);
82     if (id != -1)
83     {
84         *value = var[id].value;
85         return true;
86     }
87     return false;
88 }
```

可視化： ソースコードは色分けされており、各行・分岐・条件がカバーされているかどうかをひと目で確認できます。

カバレッジレベル *

- **Function Coverage** — 呼び出された関数の数と頻度を計測します。C++などのオブジェクト指向言語のメンバー関数も対象です。
- **Line Coverage** — 実行された行数を総行数で割った値です。実行可能なステートメントを含む行のみが対象となります。
- **Statement Coverage** — 実行されたプログラムステートメントを、実行済みステートメントと総ステートメントの比率として追跡します。
- **Decision Coverage (Branch Coverage)** — すべてのステートメントが実行され、すべての判断がすべての可能な結果を生成することを検証します。各判断はtrueとfalseの2回カウントされます。
- **Condition Coverage** — Decision Coverageに似ていますが、判断をAND／OR演算子で結ばれた基本的な部分式に分解します。各条件は2回カウントされます。
- **MC/DC** – Modified Condition/Decision Coverage — 完全なカバレッジを達成するには、判断内のすべての条件を独立してテストし、それぞれの条件が結果に独立して影響することを示す必要があります。
- **MCC** – Multiple Condition Coverage — 完全なカバレッジを達成するには、各判断におけるすべての真理値の組み合わせが少なくとも1回発生する必要があります。

* Call CoverageメトリクスはCondition Coverage、MC/DC、MCCによって暗黙的にカバーされます。

技術概要

対応言語

- C, C++, C#, QML, Tcl
- SystemC (C++ コードとしてコンパイルされた場合)
- Python (Pythonモジュールにはcoverage.py経由、ネイティブのC/C++/C#/QML/TclレイヤーのインストルメンテーションはcocoがHandler)

対応環境

デスクトップ & サーバー： Windows、Linux、macOS、Unix variants (Solarisを含む)

組込み & リアルタイム：Embedded Linux、Embedded Windows、QNX、VxWorks、FreeRTOS、クロスコンパイルターゲット経由のベアメタル

ハードウェアターゲット：MCUs、MPUs、SoCs

Qt for MCUs (Qt Quick Ultralite / QUL)：CocoはQt for MCUsアプリケーション内で使用されるC/C++バックエンドロジックをインストルメント化します。

コンパイラ & ツールチェーンサポート

Cocoは GCC、Clang、Microsoft Visual Studio、Intel C/C++、Oracle/Sun Studio、Mono C#、QNX (qcc, q++)、IAR、ARM/Keil µVision、Green Hills、HighTec、 Atmel Studio、Wind River Diabなど、幅広いコンパイラと統合できます。

一部の組込みコンパイラにはCoco Cross-Compilation Add-onが必要です。設定ファイルにより追加のツールチェーンにも対応できます。お使いのツールチェーンが一覧にない場合は、お問い合わせください。

技術概要

ビルドシステム、IDE & テストフレームワーク

ビルドシステム & IDE：
CMake、MSBuild、Visual Studio、Eclipse、Qt Creator

テストフレームワーク：
GoogleTest、Boost.Test、NUnit、Qt Test、CppUnit、Catch2 Cocoはテストフレームワークを置き換えるものではなく、テストの書き直しも不要です。インストルメンテーションはビルドプロセス中に適用され、既存のワークフローに組み込まれます。

レポーティング & インテグレーション

Cocoは、エンジニアリング・マネジメント・コンプライアンス用途に対応した柔軟なレポーティングを提供します。結果はインタラクティブなCoverageBrowserで確認でき、ソースレベルの可視化やビルド間の並列比較が可能です。

エクスポート形式：HTML、XML、CSV、JUnit、Cobertura、EMMA-XML、SonarQube互換形式、プレーンテキスト

CI/CD インテグレーション：Jenkins、GitHub、GitLab — カバレッジデータはパイプライン実行中に自動収集され、品質ゲートやダッシュボードに活用できます。

レポートはCIダッシュボード・監査・長期的なトレーサビリティに対応しています。

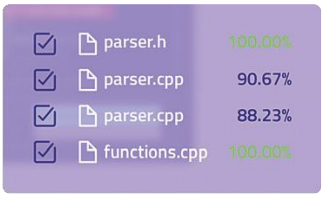
インストールガイド・リリースノート・ステップバイステップのチュートリアルを活用して、Cocoをすぐに使い始めましょう。カバレッジを迅速かつ効率的に導入できるよう設計されています。シンプルなプロジェクトのインストルメント化、Coco Test Engineによる新しいテストデータの発見、カバレッジ分析の基礎理解まで、順を追って学べます。コードカバレッジの計測に特化した専用ガイドで、さらに深く掘り下げることもできます。

Cocoのドキュメント一覧：https://doc.qt.io/coco/

Cocoを活用できる方々

- **開発者**は、どのコードパスが実行されたかを正確に把握でき、より迅速でピンポイントな修正が可能になります。
- **QAエンジニア**は、自動テストと手動テストによってアプリケーションのどの部分が検証されたかを可視化でき、テストを推測ではなくデータに基づいて進められます。
- **テストマネージャーとチームリード**は、カバレッジの進捗を追跡し、リスクのある箇所を特定し、自信を持ってリリース判断を下すための客観的なメトリクスを得られます。
- **コンプライアンス**および**安全担当チーム**は、安全規格への適合と規制要件の充足を証明するための、計測可能で監査対応のエビデンスを得られます。

Cocoでテストの有効性を数字で示す



Cocoはチームのすべてのメンバーに必要なものを提供します。開発者はどのコードが実行されたかを正確に把握し、QAエンジニアはデータドリブンなテストでギャップを解消し、テストマネージャーはCRAPスコアで複雑度とカバレッジのギャップを組み合わせてすべての関数をランク付けし、安全担当チームはTÜV Saar審査済みの標準準拠エビデンスで監査要件を満たせます。

Cocoのコードカバレッジ計測について詳しく知る

Qt Group

qt.io/coco

Qt Group (Nasdaq Helsinki：QTCOM)はグローバルなソフトウェア企業です。産業界のリーダーと150万人を超える世界中の開発者が信頼を置き、ユーザーに愛されるアプリケーションやスマートデバイスを作成しています。UIデザインから、ソフトウェア開発、組み込みシステムの最適化、そして品質管理まで、製品開発サイクル全体を通してお客様の生産性向上を支援します。Qt Groupのお客様は70以上の業界で180か国以上に広がっています。Qt Groupの従業員数は約1100名、2025年の売り上げは2億1630万ユーロでした。

公式ホームページ：https://www.qt.io/ja-jp/