HEARTLAND.DATA
ハートランド・データ株式会社

ストレスフリーな動的カバレッジ計測手法

－テスト自動化を見据え、動的カバレッジ計測は新たなフェーズへ－

ハートランド・データ株式会社
セールス&マーケティング部

動的カバレッジ計測の現状

■増えるカバレッジ計測作業

近年、「機能安全」といった規格準拠の開発プロセスが望まれる中、組込みソフトウェアを筆頭に、ソフトウェアにおけるカバレッジ分析の効率化に注目が集まって来ています。その一方で、「カバレッジの分析は工数がかかる」、「カバレッジ分析をしているが、品質向上の効果は感じられない」といった現場の声も少なくありません。

■動的カバレッジ

現場の負担を減らす手法として、動的カバレッジ計測が用いられています。実際のターゲットを使用し且つ既に持っているテストケースを実施することで、テストの事前準備に工数をかけずに計測を行えるのが特長です。

ソースコード上から必要なパラメータを導き出し、作成されたドライバーとスタブからカバレッジ率100%を目指す手法も存在しますが、動的カバレッジ計測はこれらの手法とは一線を画し、既存のテストケースを使用してテストケースの抜け・漏れがあるかを分析するのが特長です。

この手法を用いることで、カバレッジを計測する本来の目的である「テストケースが十分に網羅されていないコードを限りなく少なくすること」が可能になります。

■動的カバレッジの限界による工数の増加

工数をかけずにカバレッジの計測ができる動的カバレッジ計測方法にも限界があります。

1つ目はオーバーヘッド問題です。カバレッジ計測に必要な経路情報を出力するためには何らかのテストコードを入れなければなりません。当然ながら、動作タイミングへの影響が大きければ、計測した結果が正しいかどうか判断しかねるケースも起こり得ます。

2つ目はリソースの限界です。カバレッジ計測のために作成されたコードは元のコードよりも大きくなります。リソースに余裕のないターゲット環境では、一度に全領域をテスト対象にすることは難しくなります。

テストする領域を分割し複数回に分けてカバレッジ計測をすることで、上記課題は改善しますが、回数が増えた分、カバレッジ計測にかかる工数は必然的に増加します(図1)。

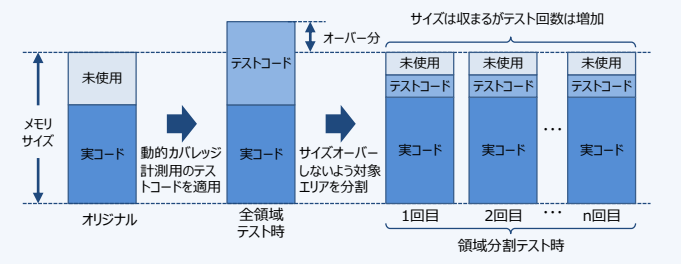


図1: テストコードによるコードサイズの増加

動的カバレッジ計測に必要な情報とその量

■動的カバレッジ計測に必要な情報とは

一般的に、動的カバレッジを計測するためには、ソフトウェアのトレースデータに代表されるように、ソフトウェアがどのように動いたかを示す情報が必要になります。

特に、組込み機器の世界では、ファームウェアのトレースデータは貴重な情報源となっており、出力の仕方は多岐に渡ります。ハードウェアレベルでトレースデータを出力できるI/Fを持ったプロセッサもあれば、ツールベンダーや開発者自身がソフトウェアレベルでトレースデータを出力する仕組みを独自に実装しているケースもあります。

■トレースデータのデータ量

トレースデータとして出力されるデータ量について考えてみます。単位時間当たりの各トレースポイントの総実行回数をP_i、1つのトレースポイントに対して必要なデータ数をSとしたとき、単位時間当たり出力されるトレースデータ量R_iは、次の式で表すことができます。

$$R_i = S \cdot P_i$$

また、処理能力の高いプラットフォームになればなるほど、トレースデータ量は増える傾向にあります。正確な動的カバレッジを計測するためには、トレースデータの欠落はあってはなりません。

そこで、大量に送出されるトレースデータを途切れることなく保存するための条件を考えてみます。単位時間当たりのCPUから送出可能なトレースデータの転送レートをR_{cpu}、データ解析用のPC側のストレージへの書き込みレートをR_{pc}としたとき(図2-1)、単位時間当たりのトレースデータ量R_iは次の条件を満たす必要があります。

$$R_i \leq min\{R_{cpu}, Rpc\}$$

さらに、トレースデータを確実に保存することも考慮すると、PC側のストレージ容量をM、トレース時間をTとしたとき、トレースデータ総量Rは次のような制約も満たす必要があります。

$$R = \sum_{i=1}^{T} S \cdot R_i \leq M$$

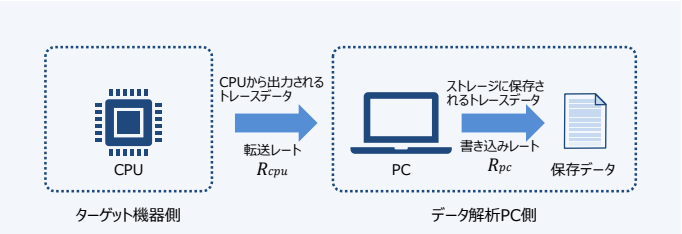


図2-1: トレースデータの転送レート

■トレースデータ出力時の負荷

トレースデータ出力時のCPUに与える負荷を考えます。トレースデータ出力にかかる負荷はオーバーヘッドと呼ばれ、トレースポイントが実行されるたびにその負荷時間が発生します。ハードウェアレベルでトレースデータが出力できればオーバーヘッドはゼロになりますが、ソフトウェアレベルで出力するときは、増加するオーバーヘッドをケアする必要があります。

トレースデータの出力に掛かる時間をT_{trc}、ある時点での単位時間当たりのトレースポイントの実行回数をP_iとしたとき、その間のオーバーヘッド時間T_{ov_i}は次のように表すことができます。

$$T_{ov_i} = Ttrc \cdot P_i$$

一方、単位時間Δt当たりの、CPUがアイドル状態であった時間をT_{idl_i}、CPUがアクティブ状態であった時間をT_{act_i}としたとき一般的に次の式が成り立ちます。

$$\frac{T_{idl_i}}{\Delta t} + \frac{T_{act_i}}{\Delta t} = 1$$

上式の$T_{idl_i}/\Delta t$は単位時間当たりのCPUの余力(余裕度)、$T_{act_i}/\Delta t$は単位時間当たりのCPUの負荷率を意味します。単純計算で、CPUがアイドル状態であった時間をオーバーヘッド時間T_{ov_i}に充てることができれば、システム的には何とか動作可能と言えます。逆に、T_{ov_i}がT_{idl_i}を超えてしまうと、システムとしては破綻してしまい、トレースデータ出力による影響が大きく出ている状況と言えます(図2-2)。

したがって、トレースデータ出力よる負荷は、少なくとも次の条件を満たす必要があります。

$$\frac{T_{ov_i}}{\Delta t} = \frac{T_{trc} \cdot P_i}{\Delta t} \leq \frac{T_{idl_i}}{\Delta t}$$

負荷の高い処理では、単位時間Δt当たりに出力されるトレースポイントの数も当然多くなりますので、システムを破綻させることなく動的カバレッジ計測を実施するためには、オーバーヘッド時間T_{trc}を限りなく小さくする設計及び実装は必須と言えます。

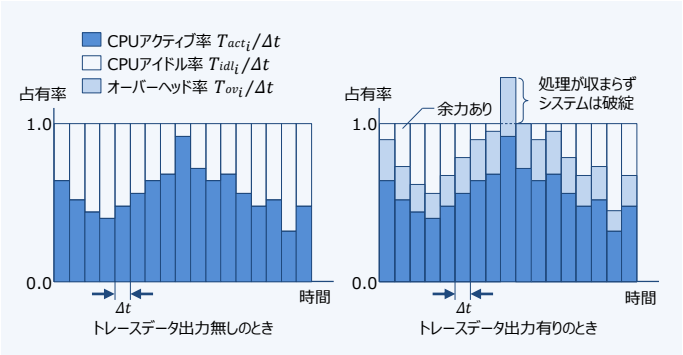


図2-2: トレースデータの転送レート

動的テストツール「DT10」のアプローチ

■動的テストツール「DT10」が抱える課題

弊社が開発・販売を行う動的テストツール「DT10」(以降、DT10)は、その名の通りソフトウェアを動的にテストすることで動的カバレッジを計測することのできるツールです。また、DT10は時間計測も得意とし、様々な角度からパフォーマンスの分析が可能なツールです。

DT10はどんなプラットフォームでも汎用的に動的カバレッジを計測できるように、ソフトウェアレベルでトレースデータを出力することを前提としています。したがって、トレースデータ出力に伴うオーバーヘッドが発生します。当然、トレースデータ量が多くなれば、大容量のストレージメディアも必要になります。DT10を使用し十分な成果を上げるためには、常に次の条件をケアしつつ作業を進める必要があります。

$$\frac{T_{ov_i}}{\Delta t} = \frac{T_{trc} \cdot P_i}{\Delta t} \leq \frac{T_{idl_i}}{\Delta t}$$

$$R = \sum_{i=1}^{T} S \cdot R_i \ \leq M$$

これは、逆に言えば、条件を満たすことができなければ、計測データに信頼性がなくなり、再度やり直しが発生することを意味します。ツールを使用し効率アップを目指すところが、何度もやり直しが発生し、かえって効率を悪くしてしまう可能性もはらんでいると言えます。これに加え、ターゲットとなる機器側のリソースの制約があれば、トレースデータを出力するためのコードを一度に全エリアに適用できないため、「領域分割テスト」を行わざるを得ず、さらに作業効率を悪くしかねません。

これらが、まさにDT10が抱える課題です。

■課題解決に向けた「DT10」のアプローチ

DT10を使用し快適に動的カバレッジ計測を行うためには、以下の3つの課題を同時に解決する必要があります。

① トレースデータ出力時のオーバーヘッドの削減
② トレースポイントのコードサイズの削減
③ トレースデータ量の削減

これらの課題を同時に解決する鍵は、トレースデータとそこから得られる情報とのトレードオフにありました。

確かに、トレースデータを逐一出力し保存できれば、ソフトウェアの実行タイミングや実行順序の情報も取得できますが、これらの情報は、テストケースに対するコードの網羅率を計測する動的カバレッジ計測では冗長な情報となってしまいます。そこで、DT10では時間や順序の情報を含まない「どの部分のコードが実行されたか」の情報に着目し3つの課題に対してアプローチしていきました。

まず、①のオーバーヘッド削減については、トレースデータの常時出力は行わずに、メモリに蓄える方式をとります。トレースデータの外部への出力処理が、データ出力のオーバーヘッドを大きくする要因となるためです。また、「どの部分のコードが実行されたか」の情報だけが保持できればいいので、トレースポイントの位置情報をメモリのビットに割り当てて保持するようにします。こうすることで、使用メモリも最小限に抑えることができます(図3-1)。

次に、②のコードのサイズの削減については、トレースデータの出力処理がメモリのビット操作に置き換わるため、パラメータの受け渡しや関数コールが発生しないため、コードサイズを小さくすることが可能になります。

最後に、③のトレースデータ量の削減についてです。必要なトレースデータのサイズは、最大でも全てのトレースポイント数となるため、予め必要なデータ量を算出することができます。

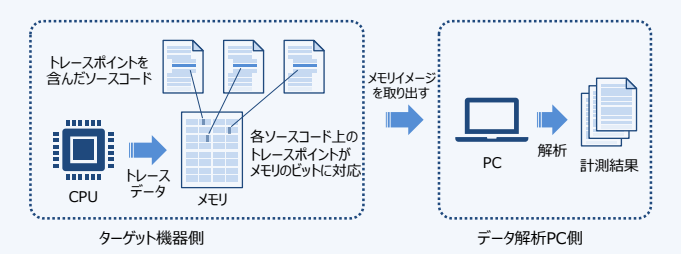


図3-1: トレースデータの保存イメージ

新しい「カバレッジ特化モード」

■「カバレッジ特化モード」として新規に機能実装

DT10のVer13.0.0では、前述の動的カバレッジ計測時の課題を解決すべく、「カバレッジ特化モード」が新たに機能実装されました。文字通りカバレッジ計測に特化していますので、「カバレッジ特化モード」下では、経路情報や実行時間・周期時間といった、時間や順序に関する情報が必要となる解析を行うことはできません。ですが、カバレッジ計測が目的であれば、従来の方法より強力な動的カバレッジ計測方法となります。

■「カバレッジ特化モード」の実証実験

この「カバレッジ特化モード」によって、3つの課題がどのくらい改善されるのかを実機環境にて検証してみます。今回使用する実機は、表4-1にあるようなARMのCortex-M3コアの載った評価ボードを使用し検証を行います。

CPUコア	Cortex-M3	使用ROM	53kB
動作クロック	16MHz	使用RAM	6.8kB
ピン数	64ピン	タスク数	10タスク
ROMサイズ	64kB	CPU負荷率	平均71%
RAMサイズ	8kB	全TP数	1102個
RTOS	FreeRTOS	トレース出力I/F	汎用ポート(6本)

表3-1: 評価ボードCPUスペック

「カバレッジ特化モード」による効果を分かりやすくするため、「通常モード」で使用したときと「カバレッジ特化モード」で使用したときに、どのような違いが出たかを次の3つの観点から比較して行きます。

① トレースデータ出力時のオーバーヘッド
② トレースポイントのコードサイズ
③ トレースデータ量

■トレースデータ出力時のオーバーヘッドの削減率

DT10では、トレースポイントに相当するコードとして、テストポイントと呼ばれるコードを使用します。テストポイントは、DT10アプリ上で対象コードに自動で挿入することができます。このテストポイント実行時に掛かる時間がオーバーヘッドとなります。

「通常モード」と「カバレッジ特化モード」でのオーバーヘッド時間を図4-1に示します。コードの最適化やDT10の機能である「高速テストポイント」を使用することで、テストポイントの出力によるオーバーヘッドをある程度減らすことはできますが、「カバレッジ特化モード」を使用することで大きく削減できることが分かります。因みに、同環境下でCPU負荷率を計測した結果は表4-2のようになりました。「通常モード」では、システムが破綻してCPU負荷は計測不可能でしたが、「カバレッジ特化モード」では、負荷率は3ポイント増加しただけで、システムへの影響が最小限に留まっていることが確認できます。

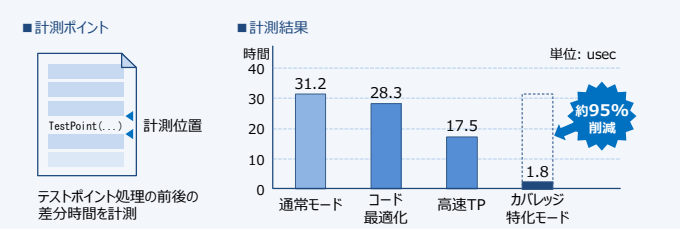


図4-1: テストポイントのオーバーヘッドの差

モード	CPU負荷		
	最小	最大	平均
オリジナル	30%	89%	71%
通常モード	計測不可	計測不可	計測不可
カバレッジ特化モード	32%	92%	74%

表4-1: モードによるCPU負荷の違い

■トレースポイントのコードサイズの削減率

今回対象としているソースコードに、DT10を使用してテストポイントを挿入したときのROMサイズの変化を表4-3に示します。この表が示すように、「カバレッジ特化モード」時のコードの増加率は「通常モード」に比べて11ポイントも少なくなっています。この理由は、特にCortex-M3のビットバンドに依るところが大きいと言えます。「通常モード」ではテストポイントの位置情報をパラメータとして情報出力用の関数に渡す必要がありますが、「カバレッジ特化モード」では、テストポイントに対応するメモリのビットを1にするだけで済むのでその差が大きく表れた結果です(図4-2)。因みに、表4-3は、同環境でビットバンドを使用しないときの削減率を表しています。「カバレッジ特化モード」では「通常モード」に比べ、パラメータ数が1つ減るのでROMサイズの削減が期待できます。

モード	使用ROMサイズ		
	オリジナル	TP挿入後	増加率
通常モード	53kB	63kB	約18%
カバレッジ特化モード	53kB	57kB	約7%
差分	–	6kB削減	11ポイント削減

表4-2: モードによるコードサイズの違い(ビットバンド使用)

モード	使用ROMサイズ		
	オリジナル	TP挿入後	増加率
通常モード	53kB	63kB	約18%
カバレッジ特化モード	53kB	60kB	約13%
差分	–	3kB削減	5ポイント削減

表4-3: モードによるコードサイズの違い(ビットバンド不使用)

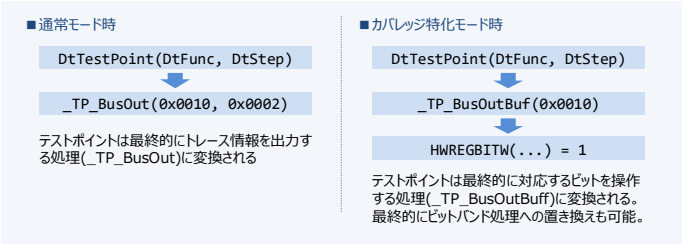


図4-2: モードによるテストポイントの処理の違い

■トレースデータ量の削減率

仮に連続的にテストポイントの情報を出し続けたときのトレースデータ量を「通常モード」と「カバレッジ特化モード」それぞれで算出し比較してみます。結果、表4-4のような算出結果が得られました。「通常モード」では、時間に比例してトレースデータ量は増えて行きますが、「カバレッジ特化モード」では、テストポイントの個数で絶対数が決まるため最大値を超えることがないことが分かります。

トレース時間	通常モード時		カバレッジ特化モード時
	1000TP/msec相当	10TP/msec相当	
1秒	16MB	160kB	最大17.6kB
1分	960MB	9.6MB	最大17.6kB
1時間	57.6GB	576MB	最大17.6kB
4時間	230.4GB	2.3GB	最大17.6kB

※テストポイント総数1102。1テストポイント当たりのトレースデータ量は16バイト。
1kB=1000B, 1MB=1000kB, 1GB=1000MBとして算出。

表4-4: トレース時間とトレースデータ量

■「カバレッジ特化モード」の効果

検証結果が示すように、「カバレッジ特化モード」は、CPUに与える負荷、コードの増加量、トレースデータ量いずれにおいても従来のDT10が抱えていた課題を解決できる機能であると言えます。また、カバレッジ計測作業の効率面からも、「カバレッジ特化モード」を有する「DT10」が、開発者にとって、より強力な武器になり得ることがデータによって裏付けられたとも言えます。ストレスフリーな動的カバレッジ計測を実現するツールと言っても過言ではありません。

■ 「カバレッジ特化モード」の設定方法

「カバレッジ特化モード」は、DT10でプロジェクト作成する時に、「カバレッジ専用ドライバを利用する」を「True」に設定するだけで使用可能です(図5-1)。操作方法の詳細や「カバレッジ特化モード」用のサンプルドライバのダウンロードは、弊社DT10サポートサイトのチュートリアルを参照ください。

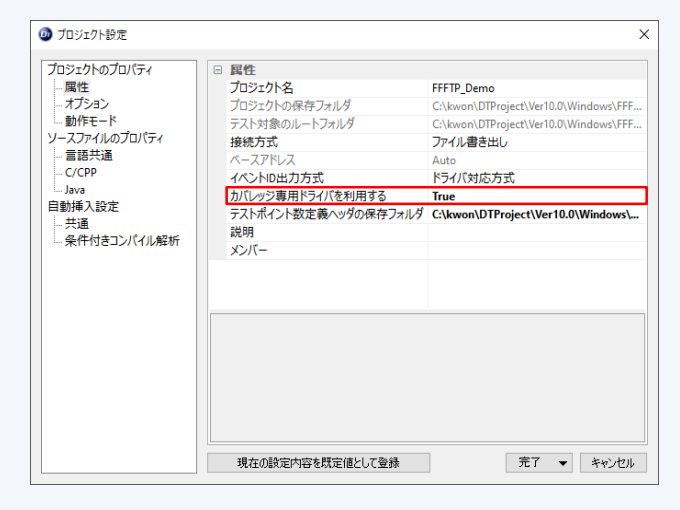


図5-1: プロジェクト設定ダイアログ

「DT-Assistant」のご紹介

■ツール導入からテスト自動化まで

ハートランド・データでは、お客様が多種多様なツールを導入し成果を上げるまでをサポートするサービスである「DT-Assistant」を提供しています。

DT-Assistant

「DT-Assistant」は、弊社製品であるDT10をはじめとするテストツールの導入からテスト自動化まで、特にテストに特化したソリューションを提供するサービスです。また、同サービスでは、機能安全に準拠した開発プロセスの中で、ツールを最大限に活用するためのプロセスの構築やテストの実施を支援する「DiET」サービスも提供しております。

以下は「DT-Assistant」サービスの一例です。

- テスト自動化環境構築サービス
- Jenkinsプラグインサービス
- アプリケーションカスタマイズサービス
- 各種ドライバサービスパック
- サンプルドライバ カスタマイズサービス
- ハードウェア カスタマイズサービス
- 導入作業代行サービス
- 動的テスト網羅性計測サービス
- パフォーマンス計測サービス
- 不具合解析代行サービス
- DT10 Automotive Edition導入サービス
- DT10 Automotive Editionプロジェクト適用サービス

ツールは、単に導入しただけでは現場に根付かず、十分な成果が上げれられないケースが多々あります。今回フォーカスした「カバレッジ特化モード」も同様で、いかに有用な機能であっても使用するフェーズやエビデンスの管理がルール化されていないと最大限の成果は得られません。ツールをいかにうまく開発プロセスに組み込んで運用できるかに掛かっています。「DT-Assistant」サービスは、導入したツールの価値を最大限に高めるサービスとも言えます。

■組込み機器向けテスト自動化ソリューション

「DT-Assistant」のサービスの1つに、「テスト自動化環境構築サービス」があります。同サービスでは、特に、組込み向けのテスト自動化のための環境構築を行っています。テスト自動化環境の構築例の1つに、図5-2のような構成のテスト自動化環境があります。専用ハードウェアを介して、ブラックボックステストを自動で行い、同時に入出力データを監視し動作の合否判定を行います。

テストを自動化し、さらに「カバレッジ特化モード」と組み合わせることで、開発終盤の限られた時間の中での回帰テストもストレスなく実施することができます。

テスト自動化環境の構築をご検討されている方、もしくは、各種ツールの現場へスムーズな導入をお考えの方、弊社までお気軽にご相談ください。

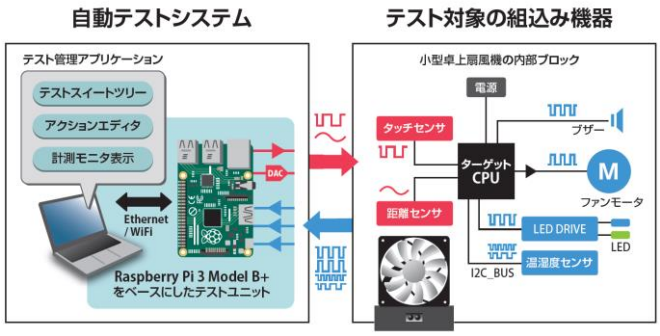


図5-2: 組込み機器向けテスト自動化システム構築例

HEARTLAND.DATA

ハートランド・データのテスト自動化について
直接お話しされたい方は、WEBのお問い合わせ
フォームよりご連絡をお願い致します。

ハートランド・データ株式会社
セールス&マーケティング部
本社
〒326-0338 栃木県足利市福居町361
TEL 0284-22-8791 / FAX 0284-22-8792
URL https://hldc.co.jp

HEARTLAND.DATA
ハートランド・データ株式会社

知っておきたいMC/DCの計測アルゴリズム

～ISO 26262規格で求められるMC/DCの概要～

ハートランド・データ株式会社
プロセスマネジメント事業部

ASILにおけるMC/DCの立ち位置

自動車向けの機能安全規格であるISO 26262:2011のPart6で定義されている「ソフトウェアレベルの製品開発」では、ソフトウェアのユニットテストについて推奨するテスト手法等を細かく規定しています。要求通りに動作することはもちろんのこと、不具合を発生させても異常動作しないことを確認しますが、安全度水準に従い、ユニット単位のカバレッジを計測して、ユニットの分岐パス全てをテストすることで、カバレッジを計測する目的である「テストケースが十分か、不要なコードが含まれていないか」を保証する必要があります。

どういったカバレッジの計測が推奨されるのかは、実際に発生するリスクの重要度によって分類されるASIL(Automotive Safety Integrity Level)に応じて決定します。(表1-1 ,1-2)

規格では、「ステートメントカバレッジ」「ブランチカバレッジ」「MC/DC」と3つのカバレッジ要件が明示されており、最も厳しい要求となるASIL-Dになると、「MC/DC」の計測が強く推奨(++)となります。「ステートメントカバレッジ」や「ブランチカバレッジ」については、どのような基準でカバレッジ計測を行うのか広く知られていますが、「MC/DC」については、どのように計測を行っているのか知らない方も多いのではないでしょうか。

ここでは、MC/DCの概念や実際にどのようなやり方でテストケースの導出を行っているのかを解説します。

表1-1: ASILの決定　　ISO 26262-3:2011 Table 4

Severity (過酷度)		Exposure (発生頻度)		Controllability(回避難易度) C1 低	C2	C3 高
低	S1	低	E1	QM	QM	QM
			E2	QM	QM	QM
		高	E3	QM	QM	A
	S2	低	E1	QM	QM	QM
			E2	QM	QM	A
		高	E3	QM	A	B
	S3	低	E2	QM	A	B
			E3	A	B	C
高		高	E4	B	C	D

※QM:Quality Management

表1-2: 構造的カバレッジ要件の決定　　ISO 26262-6:2011 Table 11

手法		ASIL A	B	C	D
1a	ステートメントカバレッジ	++	++	+	+
1b	ブランチカバレッジ	+	++	++	++
1c	MC/DC	+	+	+	++

※+:推奨、++:強く推奨

MC/DCの概要

■MC/DCとは？

MC/DCは、「Modified Condition/Decision Coverage」の略で、「DO-178B」という規格の中で定義された極めて高い品質保証を求められるテストの時に使用されるカバレッジです。MC/DCには、以下の要件を確認する必要があります。

1. プログラムの全入口／出口を少なくとも一回はテストすること
2. プログラムの判定に含まれる全条件は可能な値を少なくとも一回はテストすること
3. プログラムの全判定は可能値を少なくとも一回はテストすること
4. プログラムの判定の全条件は判定の出力に独立して影響することを示すこと

特に、4の要件から、各判定における複合条件の記述ミス(and, or)がないかを確認することができます。さらに、2の要件から判定内の全条件の真偽を確認することで、Condition Coverage相当のカバレッジを確認したことになります。また、3の要件からプログラム内の全判定の真偽を確認することで、Decision Coverage相当のカバレッジを確認したことになります。

「Modified Condition/Decision Coverage」の名称の通り、Condition CoverageやDecision Coverageも内包する考え方になります。さらに複合条件における各条件のあり得る全ての組み合わせを確認するMCC(Multiple Condition Coverage)と比べて、大幅にテストケース数を削減できます。

■MC/DCのテストケース導出方法

MC/DCを計測するにあたり、どのようにしてテストケースを導出すればよいのでしょうか。また、MC/DCの4つ目の要件に「プログラムの判定の全条件は判定の出力に独立して影響することを示すこと」とありますが、実際にテストケースを導出する際に、「独立して影響する」ことをどのように示せばよいでしょうか。具体例を用いてMC/DCのテストケース導出手法を説明して行きます。

各条件の組み合わせを確認する方法として「真理値表」を使います。真理値表は、論理和(OR)や論理積(AND)などを総称して呼ばれる論理演算の全てのパターンと結果を表にしたものです。この「真理値表」を使うことで、MC/DCのテストケースの導出を行うことができます。

二つの条件AとBの論理和(OR)と論理積(AND)を真理値表で表すと、それぞれ図1-1, 図1-2のようになります。

(A || B)
条件Aと条件Bの論理和

No.	A	B	Result
1	T	T	T
2	T	F	T
3	F	T	T
4	F	F	F

図1-1: 論理和の真理値表

(A && B)
条件Aと条件Bの論理積

No.	A	B	Result
1	T	T	T
2	T	F	F
3	F	T	F
4	F	F	F

図1-2: 論理積の真理値表

最初に、条件が2個の論理和(OR)の場合のテストケースの導出を考えてみます。

1. 真理値表の作成

条件は2個で、それぞれの条件でT(真)とF(偽)の二通りのパターンがあるため、全ての組み合わせは最大で4通りになります。この規模であれば特に問題ありませんが、条件数が増えていくに従って指数関数的にテストケース数が増大していくことになります。そのため、各条件の全ての組み合わせをテストしていくことは現実的ではありません。

このため、MC/DCの考え方が必要となります。MC/DCでは、この全ての条件の組み合わせから、できる限りテストの必要のないテストケースを省略して行きます。こうすることで、全体のテストケース数を削減が可能となります。

2. 条件A, Bが判定の出力に独立して影響することを示す

条件Aを基準にしてみると、条件Aが真となった時点で判定結果が条件Bの真偽に関わらず真となることから、No.1とNo.2のテストケースは一つにまとめることができます。

まず、条件A以外の条件、つまり条件Bを”F”(“X”も含める)に固定して考えます。条件Bを”F”に固定することで、図2-1のように条件Aが独立して判定に影響を与えていることがわかります。従って、No.1またはNo.2とNo.4のテストケースは、MC/DCを満たすために必要なテストケースということになります。

次に、同様に条件Aを”F”に固定して考えると、条件Bが独立して判定に影響を与えていることがわかります(図2-2)。このことから、ここで抽出したNo.3とNo.4のテストケースもMC/DCを満たすために必要なテストケースであるといえます。

No.	A	B	Result
1、2	T	X	T
4	F	F	F

X: Don’t Care

条件Aの判定結果によって結果が決まる

図2-1: 条件Bを”F”(“X”も含む)に固定した場合

No.	A	B	Result
3	F	T	T
4	F	F	F

条件Bの判定結果によって結果が決まる

図2-2: 条件Aを”F”に固定した場合

3. 導出したテストケースをまとめる

上記２の手順により、全ての条件について独立して判定に影響を与えるテストケースのペアを導出することができます。今回のケースでは、以下のような結果となります。

・A → No.1またはNo.2、No.4
・B → No.3、No.4

つまり、MC/DCを満たすために必要なテストケースはNo.1またはNo.2、No.3、No.4の3つであることがわかります(図2-3)。

No.	A	B	Result
1、2	T	X	T
3	F	T	T
4	F	F	F

XはDon’t Care
は条件Aのペア
は条件Bのペア

図2-3: 論理和で必要なテストケース

同様に、条件が2個の論理積(AND)の場合のテストケースを考えてみます。全ての条件について独立して判定に影響を与えるテストケースのペアは、図2-4のようなペアとなります。

No.	A	B	Result
1	T	T	T
2	T	F	F
3,4	F	X	F

XはDon’t Care
は条件Aのペア
は条件Bのペア

図2-4: 論理積で必要なテストケース

従って、論理積(AND) で、MC/DCを満たすために必要なテストケースは、No.1、No.2、No.3またはNo.4の3つであることがわかります。

・A → No.1、No.3またはNo.4
・B → No.1、No.2

では、更に条件を増やし、以下のような条件が3つの場合のテストケースを考えてみます。

```
( A || ( B && C ) )
```

条件が3つであることから、全ての組み合わせの真理値表は、図2-5のようになります。

No.	A	B	C	Result
1	T	T	T	T
2	T	T	F	T
3	T	F	T	T
4	T	F	F	T
5	F	T	T	T
6	F	T	F	F
7	F	F	T	F
8	F	F	F	F

条件が3つであるため、テストケースは8パターン

図2-5: 条件が3つの時のテストパターン

この場合においても、まとめられるテストケースが存在します。本来であれば、条件の数がA,B,Cの3個であるため、全ての組み合わせは2x2x2の8通りとなるはずですが、前述の論理和と論理積の例のようにテストケースをまとめてみます。

条件Aが真となった時点で、判定結果が条件B、Cの真偽に関わらず真となることから、No.1～4は一つにまとめて考えることができます。また、条件A、Bがともに偽となった時点で、判定結果が条件Cの真偽に関わらず偽となることから、No.7とNo.8も一つにまとめられます。これらの重複がない状態の真理値表は、図2-6のようになります。

No.	A	B	C	Result
1~4	T	X	X	T
5	F	T	T	T
6	F	T	F	F
7、8	F	F	X	F

No.1～4とNo.7～8は、テストケースをまとめることができる

図2-6: テストケースをまとめた真理値表

この4通りのテストケースに絞った真理値表から、各条件が判定の出力に独立して影響することを示すテストケースのペアは、図3-1のようになります。

No.	A	B	C	Result
1~4	T	X	X	T
5	F	T	T	T
6	F	T	F	F
7、8	F	F	X	F

XはDon't Care
は条件Aのペア
は条件Bのペア
は条件Cのペア

図2-7: テストケースのペア

図2-7から、それぞれの条件について独立して判定に影響を与えるテストケースのペアは、以下のような結果となります。

・A → No.1~4のいずれか、No.6
・B → No.5、No.7またはNo.8
・C → No.5、No.6

つまり、MC/DCを満たすために必要なテストケースはNo.1~4のいずれか、No.5、No.6 、No.7またはNo.8の4つとなります。

このように順を追ってテストケースを導出して行くことで、同時に、前述のMC/DCの要件が満たされることになります。

1. プログラムの判定に含まれる全条件は可能な値を少なくとも一回はテストすること。
   → 各条件で真偽ともに確認できる。

2. プログラムの全判定は可能値を少なくとも一回はテストすること。
   → 各判定で真偽ともに確認できる。

3. プログラムの判定の全条件は判定の出力に独立して影響することを示すこと。
   → 各条件の真偽がそれぞれ独立して判定文の真偽に影響することを確認できる。

特に、図2-7のようなペアができるケースでは、条件Aを満たすためのテストケースのペアと条件Bを満たすためのテストケースのペアをそれぞれ確認しておけば、条件Cを満たすためのテストケースのペアは、既に確認済みとなっています。

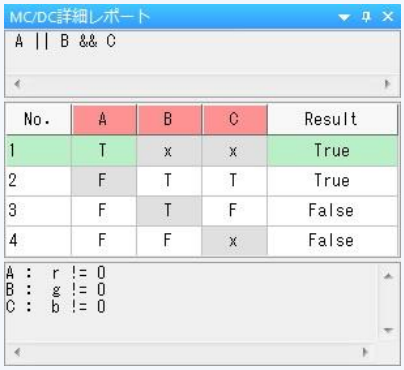
また、条件Aを満たすためのテストケースのペアはNo.1~4のいずれかとNo.7またはNo.8でも良いことになります。ただし、このテストケースのペアを先に選んでしまうと条件Cについては、再度未確認のテストケースNo.6も含めて確認をしていく必要が出てきてしまいます。

一般的に、条件n個に対して必要となるテストケース数は最小で(n+1)個になると言われていますが、条件数が増えると、このようにテストケースのペアの組み合わせ自体も増えいきます。従って、なるべく他の条件で使うテストケースと重複されるものを選ぶことが、効率的にMC/DC計測を進めていくコツと言えます。

MC/DCの動的解析ができるテストツール

弊社では、このように非常に手間のかかるMC/DCのテストケースの導出を自動で行い真理値表を作成することのできる動的テストツール「DT10 Automotive Edition」(以下DT10AE)を提供しています。

最新のバージョンであるDT10AE Ver12.0.0では、MC/DCレポート解析機能(図3-1)に加え、ソースコード構造解析(図3-2)に対応しております。ISO 26262の規格適合をお考えの方は、是非導入をご検討ください。



条件式からMC/DC計測用の真理値表を自動作成

図3-1: MC/DC詳細レポート

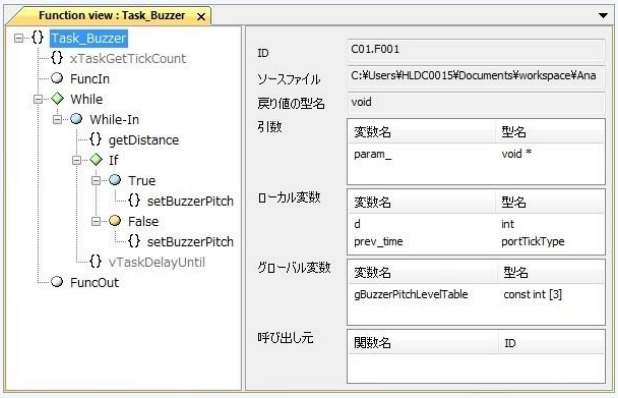


図3-2: ソースコード構造解析

ハートランド・データ株式会社
プロセスマネジメント事業部
本社
〒326-0338 栃木県足利市福居町361
TEL 0284-22-8791 / FAX 0284-22-8792
URL http://hldc.co.jp