空調冷熱機器の熱源機開発における マイコン変更時の同一性検証（単体試験）の自動化

和歌山支所　機器組込ソフトウェア技術部　機器技術第二課

・小門　希
・目方　徹
・田井　一

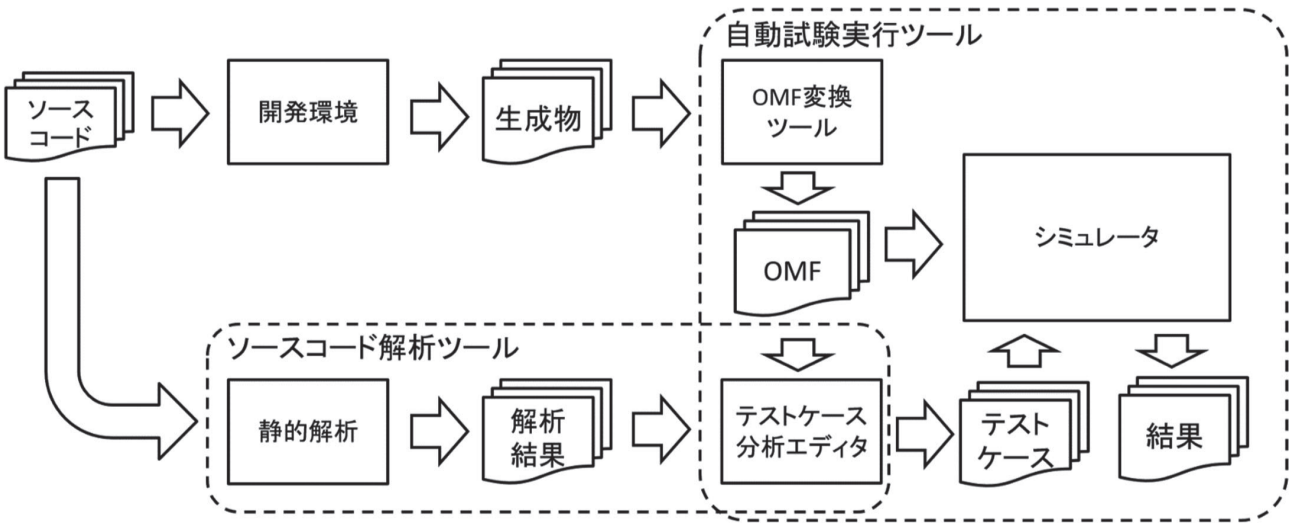
要　旨

空調機等，冷熱システム機器の製品開発を開始して以降，性能改善やマイコン生産中止により，マイコン変更対応が数年周期で発生している。マイコン変更に伴い，コンパイラも変更となる。高級言語で書かれた同一のソースコードであっても，異なる機械語に翻訳されるため，マイコン変更前に動作が保証されていたプログラムでも，マイコン変更後は保証できなくなる。

当社では，マイコン／コンパイラ変更の際，マイコン変更前後でソースコードに変更が発生していない箇所に対しても動作保証するため，シミュレータを用いて全て手動で同一性検証を実施してきた。しかし，変更していない全てのソースコードに対して同一性検証を行なうには，ライン数が多く，多大な時間を要しており，開発している全ての機種に対して今後も周期的に作業が発生するため，大きな課題となっていた。

この課題を解決するため，winAMS という自動試験実行ツールに着目した。本ツールは，マイコン変更前のソースコードより，自動で試験パターンの抽出，試験の実行及び試験手順書の生成まで行なうことができる。さらに，その試験手順書を用いて，マイコン変更後のソースコードに対して自動で試験を実行し，結果判定を行なうことができる。よって本ツールを用いて，マイコン変更時の同一性検証作業の効率化を図った。

今回，空調冷熱機器の熱源機における，16bit マイコンから 32bit マイコンへのマイコン変更開発にて，winAMS ツールの調査を実施し，本ツールを用いた同一性検証自動化の手順を確立し，マイコン変更開発で初めて適用した。結果として，従来と同等の品質を保ったまま，同一性検証の工数を削減することに成功し，約 17％の開発費用削減に繋げることができた。



1．まえがき

1．1　背景

空調機等，冷熱システム機器の製品開発を開始して以降，性能改善やマイコン生産中止により，マイコン変更対応が数年周期で発生している。マイコン変更に伴い，コンパイラも変更となる。高級言語で書かれた同一のソースコードであっても，異なる機械語に翻訳されるため，マイコン変更前に動作が保証されていたプログラムでも，マイコン変更後は保証できなくなる。

当社では，マイコン／コンパイラ変更の際，マイコン変更前後でソースコードに変更が発生していない箇所に対しても動作保証するため，シミュレータを用いて手動で同一性検証を実施してきた。しかし，変更していない全てのソースコードに対して同一性検証を行なうには，ライン数が多く，多大な時間を要しており，開発している全ての機種に対して今後も周期的に作業が発生するため，大きな課題となっていた。

今回，これらの課題に対して，空調冷熱機器の熱源機（以下熱源機）における，16bit マイコンから 32bit マイコンへのマイコン変更開発において，ツールを用いて，試験仕様書の自動作成及び試験を実施（同一性検証）する手法を適用した。さらに，一度作成した試験仕様書を再利用することによって，今後のマイコン／コンパイラの変更に対し，検証に要する工数削減を見込むことができる。本稿では，これらの手法について紹介する。

1．2　コンパイラ変更によるソースコード翻訳の差異

マイコン変更に伴い，コンパイラも変更されることによって，異なる機械語に翻訳される例を下記に示す。

```
例：signed int val;      （val は 16bit 型）
    if（val==0x8000）{
        TRUE;
    }else{
        FALSE;
    }
```

上記例のような構文において，val に 0x8000（-32768）が格納されている場合，ベースの 16bit マイコンにおいては，もちろん TRUE となる。しかし，32bit マイコンでは結果が FALSE となり，同構文にも関わらず動作が異なる。これは，32bit マイコンでは，if 文判定時に以下のように 32bit 長で判定されるためである。

左辺の val　　：符号ありのため，0xFFFF8000
右辺の 0x8000 ：符号なしのため，0x00008000
よって 32bit マイコンでは上記の if 文を右上のように修正する必要がある。

修正例：if（val==（int）0x8000）{

このような差異は，マイコンメーカー側では保証されないため，開発側にてマイコン／コンパイラの変更前と変更後で，プログラムが同様に動作することを保証する必要がある。

2．同一性検証

2．1　シミュレータを用いた手動試験（従来手法）

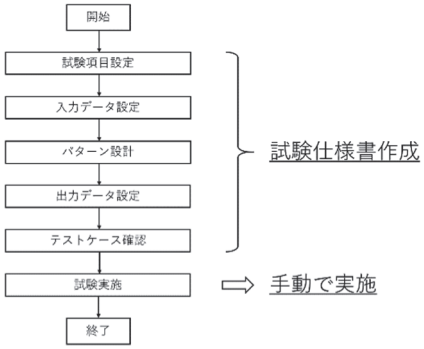
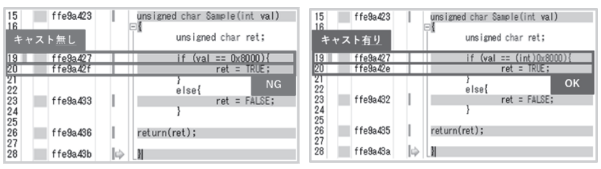


図１．シミュレータを用いた手動試験

図１にシミュレータを用いた従来の手動試験の流れを示す。図１から，試験仕様書作成と試験実施の２つの工程に分けられる。図２に 32bit マイコンにおける手動試験の例を示す。図２．(a) は１．２節で示した修正前の構文に対する試験の例であり，図２．(b) は修正後の試験の例である。試験仕様書通りに変数に値を設定し，１ステップずつプログラムを実行する。図２．(a) において，条件文にあらゆる値を設定しても 19 行目が TRUE にならず，20 行目が実行されないため，期待通りの動作とならない。一方，図２．(b) では，条件文に 0x8000 を設定すると，TRUE となり，期待通りの動作となる。このように手動試験では，マイコン／コンパイラ変更により，図２．(a) のような想定外の動作にならないかの検証を行なう必要がある。手動試験は，試験設計と試験実施を手作業で実施する必要があるため，ライン数が多い場合，多大な工数が必要になる。



(a) 試験結果（NG の場合）(b) 試験結果（OK の場合）

図２．手動試験の例

２．２　自動試験（今回適用した手法）

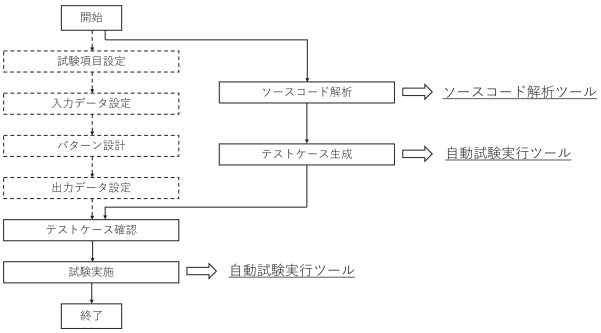


図３．ソースコード解析ツールと自動試験実行ツールを用いた自動試験

図３に今回採用した，ソースコード解析ツールと自動試験実行ツールを用いた自動試験の流れを示す。マイコン変更前の開発環境において，解析ツールを用いて対象のプログラムの解析を行なう。次に，自動試験実行ツールに上記の解析結果を入力として，試験パターンを自動生成し，試験結果を出力する。さらに，マイコン変更後の開発環境において，同じ試験パターンを用いて試験結果を出力する。同一性検証における自動試験は，試験パターン作成及び試験実行の工数を削減できる。

３．ソースコード解析ツールと自動試験実行ツールを用いた自動試験の手法

本手法による同一性検証は，マイコン変更前と変更後の異なる開発環境において，プログラムが同じ結果になることを検証し，品質を保証する。以下に自動試験の手順を示す。ただし，１～４はマイコン変更前，５はマイコン変更後の開発環境で実施する。

1. マイコン変更前のベースソースコードの解析
2. 試験パターンの生成
3. 試験パターンの確認・修正
4. 試験仕様書の作成
5. 結果の出力（マイコン変更後の開発環境）

今回，16bit マイコンから 32bit マイコンへのプログラムの移植を行なった。表１に使用する試験ツールを，表２に試験環境を示す。

表１．試験ツール

ソースコード解析ツール	Case Player[1] v6.5
試験自動実行ツール	カバレッジマスター winAMS[1] v6.5

表２．試験環境

OS	Windows 10 Enterprise x64 バージョン 1909
プロセッサ	Intel（R）Core（TM）i7-8650U
RAM	32.0 GB

３．１　ソースコードの解析

最初に，Case Player で既存のソースコードの解析を行なう。その結果として図４に示すフローチャートが生成物として出力される。

図４から，正確にプログラムの解析が行われていることがわかる。

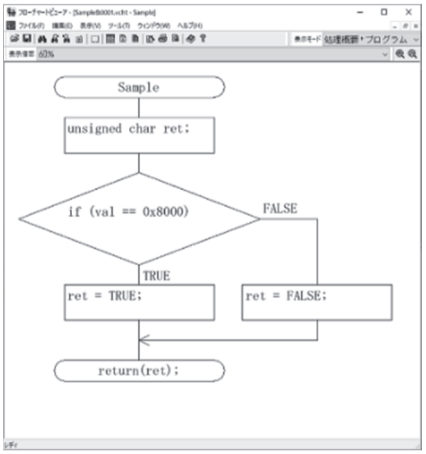


図４．生成されたフローチャート

３．２　試験パターンの生成

次に，Case Player によるソースコードの解析結果をもとに winAMS を用いて，試験パターンの生成を行なう。図５に生成された試験パターンがテスト分析エディタによって展開された画面を示す。

図５．テスト分析エディタ

この画面では，0x8000 と 0x8001 が入力値として自動

で記載されている。追加でパターンを設定したい場合，入力値を手入力する必要がある。自動生成された試験パターンによっては，Case Player による解析が十分に行われず，不完全な試験パターンが生成されることがある。また，同一性検証という観点において，TRUE，FALSE を満たす試験パターン以外に，１．２節で述べたコンパイラ変更によるソースコード翻訳の差異についても検証を行なう必要がある。そのため，自動生成された試験パターンを確認し，不十分であれば手動で修正を行なう。

３．３　試験パターンの確認・修正

図６に新たな入力値として０を設定したパターンを示す。パターンと入力値を設定後，修正後の試験パターンの作成を行なう。図７に修正後の試験パターンを示す。生成された試験パターンに対して，試験することによって，出力値が自動で入力される。

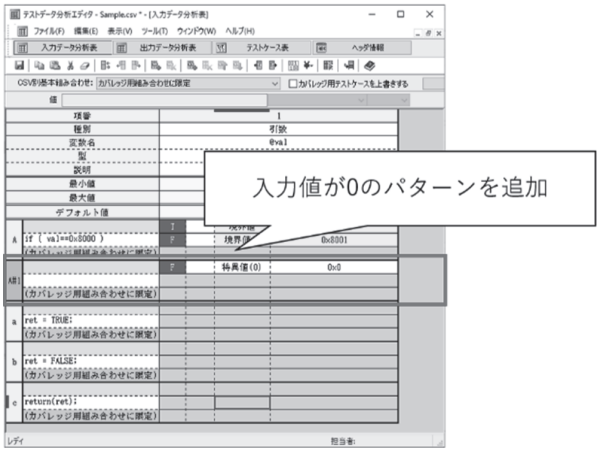


図６．パターンの追加

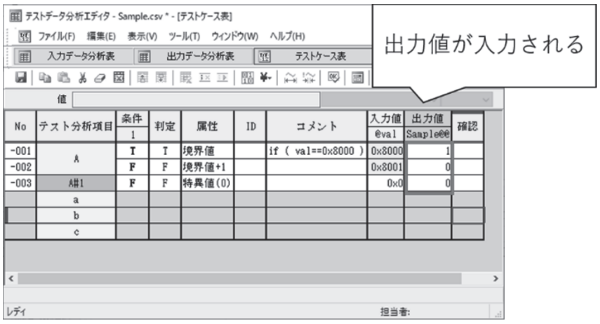


図７．修正後の試験パターンの作成

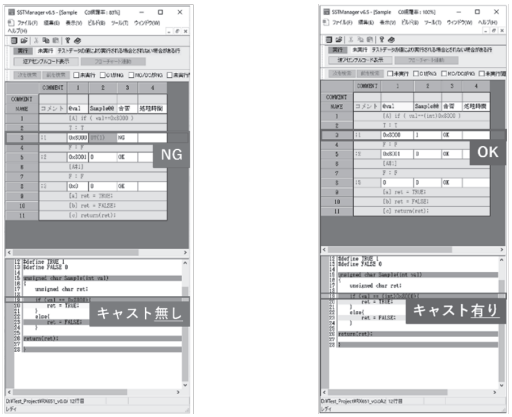
３．４　試験仕様書の作成

winAMS の機能の一つであるテストデータ分析エディタを用いて，テスト結果を確認する。図８に，試験結果，網羅率を示す。テスト結果が全て OK で，かつ網羅率が 100% になっていれば，試験仕様書の作成は完了となる。

図８．テスト実施結果

３．５　結果の出力（マイコン変更後の開発環境）

３．４節において作成された試験仕様書をマイコン変更後の開発環境を用いて試験を実施する。図９に結果を示す。図９．(a) の試験結果が NG であることから，0x8000 の解釈に差異が生じていることがわかる。これは，１．２節において述べたように，16bit マイコンと 32bit マイコンでは，ビット数が異なるため，同一の結果を得ることができなかった。一方，図９．(b) では，試験結果が OK となっていることがわかる。これは，16bit マイコンから 32bit マイコンへプログラム移植時，キャスト演算子を用いて，ビット数の違いに対する処理を施したからである。



(a) 試験結果（NG の場合）　(b) 試験結果（OK の場合）

図９．テスト実施結果（32bit マイコン）

次章に本章で述べた手法を用いて，熱源機のマイコン変更開発にて，同一性検証を行なった結果を示す。

４．結果

本稿で採用した手法を用いて，16bit マイコンから32bit マイコンへのプログラムの移植を行なった。表３に結果を示す。

表３．同一性検証結果

手法	手動試験（従来手法）	自動試験（適用手法）
関数の合計 ［個］	2238	
適用した関数［個］	1439	799
１人月あたりの進捗関数［個］	267	476

表３から，試験対象の合計件数は 2238 件であり，自動試験の適用件数が799件であることがわかる。これは，関数や変数の構造によって，自動試験を適用することが困難であると判断した関数については，従来の手動試験を適用したからである。また，１人月あたりの進捗関数は，１人月を 160 時間としたとき，それぞれ手動･自動試験にて完了した試験項目（関数）の個数を示す。図 10 に自動試験適用による工数削減結果を示す。図 10 は自動試験を適用した 799 件の試験に対して，従来手法で実施した場合の推定値と本手法で実施した場合の実測値の工数を示す。従来手法と適用手法を比較した結果，削減率は 44% となった。

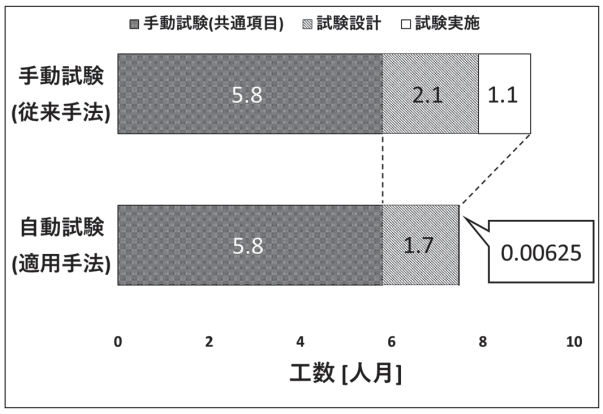


図 10．工数削減結果

５．課題への対応

熱源機のソースコードに winAMS を適用するにあたり最も大きな課題は自動試験の適用率の低さである。前項の結果から自動試験を適用できたのは全体の 36% 程度であり３分の２は winAMS を使用せず従来手法での手動試験となった。今回のマイコン変更開発では，本課題への対策の検討／実施を行なった場合の開発スケジュールへの影響が不明であったため，開発中には解決できなかった。その後の調査で winAMS 適用率の低さの主な原因は以下の３点であることがわかった。

１．C 言語規格である C99 への対応不十分
２．試験対象に戻り値を返す下位モジュールが存在
３．H／W からの応答を期待する下位モジュールが存在

５．１　C 言語規格である C99 への対応不十分

１点目の課題について，コーディング規約では ESCR Ver.3.0（C99 に対応）に従うことになっているが，対象製品は派生開発がメインであり，過去資産を流用することが多く，ソースコード全体として C99 に対応できていないのが現状である。この課題を解決するためには過去資産流用時に現在のコーディング規約に沿うようリファクタリングする必要がある。

５．２　試験対象に戻り値を返す下位モジュールが存在

２点目の課題は，戻り値を返す下位モジュールが存在する場合，適用が困難である。理由は，下位モジュールが任意の値を応答する試験パターンを手動で作成するため。図 11 に戻り値を要求する下位モジュール例を示す。図11に示すような戻り値を要求する下位モジュールが２つ存在する場合，それぞれ TRUE／FALSE となる試験パターン設計を手動で行なう必要がある。

```
void Cj_hi_cut_test(void)
{
/****************/
/*   開始判定   */
/****************/
        if( cj_hi_cut_test_jg_bgn() ){
                Cj_hi_cut_test_st = ON;
        }

/****************/
/*   終了判定   */
/****************/
        if( cj_hi_cut_test_jg_end() ){
                Cj_hi_cut_test_st = OFF;
        }
}
```

図 11．戻り値を要求する下位モジュール例

５．３　H／W からの応答を期待する下位モジュールが存在

３点目の課題は，H／W，マイコンからの応答依存する下位モジュールが存在する場合も適用が困難である。理由は，シミュレータでは再現不可能なため。これらの課題を解決する方法としてスタブ関数と呼ばれる代替モジュールを使用することが考えられる。下位モジュールをスタブ関数に置き換えて疑似的な応答を返すことで下位モジュールの実際の処理によらず期待する結果を試験パターンに組み

込むことができ，自動試験の適用が可能となる。

5．4　改善後の工数削減効果

上記3点の課題を解決することで従来手法によって試験を行なっていた1439件の内，792件に対して自動試験を適用することが可能となり合計1591件，全体の71%の適用が見込める。よって，今後の開発では本対策を実施することで，今回のマイコン変更開発以上の削減効果が期待できる。図12に対策を行った場合の工数削減見込みを示す。図12から，初回適用時は試験設計の時間が必要となるが，課題解決前と比較して22%の改善を見込む。さらに，一度自動試験を適用したモジュールに対して，次回開発の同一性検証では，自動試験の実行時間のみとなるため，大幅な工数削減が見込める。

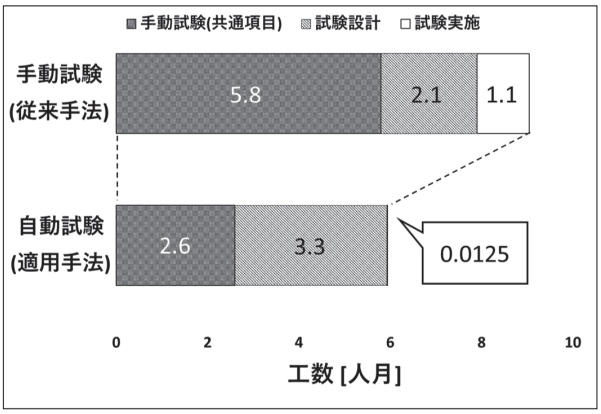


図12.　工数削減効果

図13に次回開発での工数削減効果の見込みを示す。図13から，次のマイコン変更開発において試験設計の時間が不要となるため，全体として70%の工数削減が見込める。

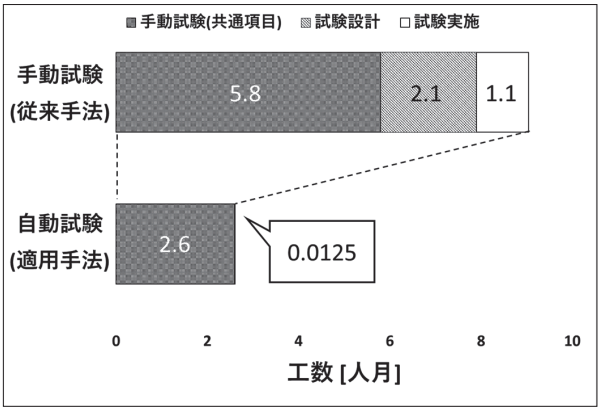


図13.　次開発以降の工数削減効果

6．むすび

今回，ソースコード解析ツールであるCase Playerと自動試験実行ツールであるwinAMSを用いた同一性検証の方法を紹介した。結果として，16bitマイコンを搭載した熱源機から32bitマイコンを搭載した熱源機へのプログラム移植の工数は，自動試験適用箇所において，約44%削減できた。今後開発予定の機種においても同様の成果が期待でき，かつ一度作成した試験仕様書は再利用可能であるため，次のマイコン変更開発では試験仕様書作成分の工数も削減できる。そのため，作業全体として更なる工数削減に期待できる。また，自動試験の適用を進めることで，手動では多大な時間を要する回帰試験の工数も削減でき，開発毎に回帰試験の適用が容易になるため，デグレードの防止や潜在不具合の発見などの品質向上も見込める。

最後に，本開発及び本稿執筆にあたり，ご指導，ご支援いただいた三菱電機(株)冷熱システム製作所の関係者各位に深く感謝いたします。

執筆者紹介

小門　希

2006年入社。業務用産業冷熱機器のソフトウェア開発に従事。現在，和歌山支所　機器組込ソフトウェア技術部　機器技術第二課　グループリーダー

目方　徹

2011年入社。業務用産業冷熱機器のソフトウェア開発に従事。現在，和歌山支所　機器組込ソフトウェア技術部　機器技術第二課

田井　一

2020年入社。業務用産業冷熱機器のソフトウェア開発に従事。現在，和歌山支所　機器組込ソフトウェア技術部　機器技術第二課