特集論文

モデルベース開発におけるSPILSを用いたソフトウェアテストの効率化

三田事業所　技術第2部　技術第1課
前田 頼正、竹内 康尊、杉原 依理未

1．まえがき

近年、当所では車載機器の組込みソフトウェアの開発効率を向上させる目的で、モデルベース開発の導入が増えている。モデルベース開発では、システムやソフトウェアで実現する機能をMATLAB/Simulinkのモデルとして記述することにより、MILS(Model in the Loop Simulation)による仮想検証や自動コード生成など従来の開発手法と比べて効率化が期待できる。開発したソフトウェアの検証は、実機やHILS(Hardware in the Loop Simulation)を用いて行っているが、テストデータの設定や出力データの記録を手作業で行っているため、改善の必要があった。このため、SPILS(Simulator based Processor in the Loop)を用いてソフトウェアをマイコンシミュレータ上で動作させ、自動的にテストを行う環境を構築した。本稿では、ソフトウェアテストの環境と試行によって得られた改善効果について紹介する。

2．SPILSテスト手法

ソフトウェアテストでは、統合ソフトウェアがソフトウェア要件の検証基準を満たすことを検証する。今回構築したテスト手法は、ソフトウェア要件定義にモデルベース開発を導入していることが前提である。

2.1　従来のテスト手法

従来のテスト手法では、実機またはHILSを用いて、製品やマイコンの入出力を対象にテストを実施していた。HILSはマイコンの周辺回路を汎用の模擬装置に置き換え、これらをリアルタイムに制御することにより、実機の動作環境を電気的に模擬するシステムである。テスト対象の統合ソフトウェアは実マイコンまたはマイコンエミュレータで実行する。HILSではアナログやデジタルの電気信号を物理的に入出力するため、ハードウェアの性能やノイズなどの影響でMILSによる仮想検証と同等の精度でテストを行うことができない課題があった。

2.2　SPILSのテスト手法

今回構築したテスト手法では、実機やHILSの代わりにSPILS上で統合ソフトウェアを動作させてテストを実施する。SPILSは、統合ソフトウェアをマイコンシミュレータ上で動作させ、マイコンの周辺回路を含む実機の動作環境をすべてソフトウェア的に模擬するシステムである。SPILSでは、統合ソフトウェアに対するテストデータの入出力を物理的に行うのではなく、データとして疑似的に入出力する。これにより、ハードウェアの性能やノイズなどの影響を受けず、MILSによる仮想検証と同等の精度でテストを実施することができる。実機またはHILSとSPILSの比較を表1に示す。

表1．HILSとSPILSの比較

項目	実機またはHILS	SPILS
実行環境	ハードウェア （実マイコンまたは マイコンエミュレータ）	ソフトウェア （マイコンシミュレータ）
入出力方法	物理的に入力 （電気信号を入出力）	疑似的に入力 （データを入出力）
テスト精度	低い （テスト環境に依存）	高い （テスト環境に非依存）

従来のテスト手法では、HILSを構成する装置に合わせて、テストデータ及び期待値を手作業で設定する必要があり、多くの工数を要していた。一方、SPILSでは、PC上でデータの入出力を疑似的に行うため、これらの作業を容易に行うことができる。MILSから得られた出力データをテスト期待値とすることで、さらなる効率化ができる。すなわち、同じテストシナリオにおいて、MILSで動作させるモデルとSPILSで動作させた統合ソフトウェア双方の出力を比較することでテストを行う。この手法は、一般的にBack to Backテストと呼ばれ、モデルとコードの一致性を確認する。SPILSを用いてBack to Backテストを行う場合、実機やハードウェアが不要であり、PC上で完結してテストを実施できることから、自動化が容易であり入出力やパラメータ数によらず効率的なテストが可能である。図1にこれらの概要を示す。

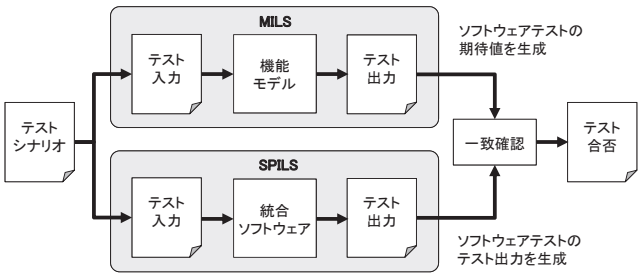

図1．SPILSテスト手法の概要

2.3 SPILSのテスト環境

2.2節図1を実現するため、今回MILSとSPILSで実際に構築したテスト環境の構成を図2に示す。MILSには機能モデリングに用いているMATLAB/Simulinkを使用し、SPILSにはソフトウェア単体テスト用ツールとして使用実績のある、ガイオ・テクノロジー(株)社製のカバレッジマスターWinAMS(以下WinAMS)を使用した。

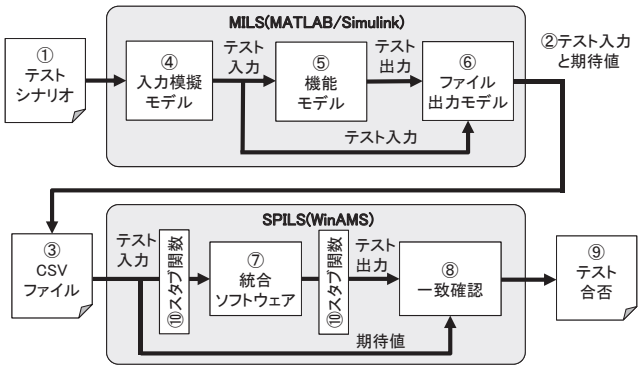

図2．構築したテスト環境の構成

(1)MILSの構成

MILSは、製品に搭載されたマイコンと周辺回路との間で入出力される電圧、電流などの物理的な信号をSPILSで取り扱えるデータに変換する。MILSはテストシナリオ①を元にテスト入力と期待値②を出力し、WinAMSに入力するCSVファイル③を出力する。MILSでは、テスト入力を機能モデルに入力できるように変換するための入力模擬モデル④、ソフトウェア要件定義で作成した機能モデル⑤、機能モデルのテスト出力をWinAMSのCSVファイルに出力するためのファイル出力モデル⑥を接続して動作させる。

テストシナリオは、入力模擬モデル内の信号発生ブロックに波形を設定したり、ファイルを直接読み込ませて設定するなどの方法で与えることができる。入力模擬モデルに実機上のハードウェアや自動生成コード以外で実装する機能を補う処理を追加し、テストシナリオを詳細に設定しなくとも、効率的にテストを実施できるように工夫した。

(2)SPILSの構成

SPILSは、マイコンに入出力されるデータを変数として取り扱い、統合ソフトウェアをマイコンシミュレータで疑似的に実行して結果を出力する。SPILSではMILSで出力したCSVファイルをWinAMSに読み込ませて統合ソフトウェア⑦を実行すると、テスト出力と期待値の一致確認⑧が行われ、テスト合否⑨が得られる。

SPILS上で統合ソフトウェアを実マイコンと同様に動作させるためには、マイコン周辺機能の代わりに外部から入出力を与える必要がある。WinAMSにマイコン周辺機能を模擬する機能はないが、任意の関数を元のソースファイルとは別のソースファイルに定義したスタブ関数に置換する機能を備えている。この機能を利用してマイコン周辺機能の制御関数をテスト用に加工したスタブ関数⑩で置換し、SPILS上で統合ソフトウェアにテストデータを入出力できるように変更する。前記を含め、テストのために統合ソフトウェアで変更が必要な処理と変更内容を表2に示す。

表2．統合ソフトウェアに対する変更内容

処理	変更内容
無限ループ	マイコンシミュレータ上でプログラムの実行を阻害する処理を無限ループとならないように加工する。典型的な処理としては、タイマなどのマイコン周辺機能のレジスタポーリングがある。
ハードウェア入出力処理	ハードウェアとの信号入出力処理をスタブ関数で置換し、変数を介して統合ソフトウェアにデータを疑似的に入出力できるよう加工する。例えば、A/Dコンバータであれば A/D変換値、通信回路であれば送受信データを入出力できるようにスタブを設計する。
テスト開始/終了位置の定義	統合ソフトウェアにデータ入力を行うポイントと、統合ソフトウェアからデータ出力するポイントを定義する。WinAMSではそれぞれのポイントでダミー変数に値を書き込むことで設定できる。

2.4 SPILSによるテスト手順

2.3節図2のSPILSテスト環境を用いて、テストを実施するまでの手順を図3に示す。(1)～(4)の作業は、テストの準備として一度だけ行う必要がある。(5)～(6)の作業はテストシナリオ毎に、繰り返し実行することができる。各作業の詳細と考慮すべき点について説明する。

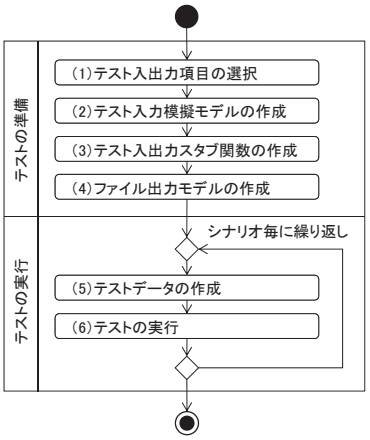

図3. テスト手順のワークフロー

(1)テスト入出力項目の選択

テスト仕様を分析し、統合ソフトウェアに入力するデータと、統合ソフトウェアから出力して期待値と比較する

データを抽出する。多くの入出力項目を扱えるようにするとテスト環境の汎用性が高まるが、環境構築やテストデータの設定が煩雑になるため、最小限の入出力項目に絞って、環境を準備するようにした。

(2)テスト入力模擬モデルの作成

2.4節(1)で抽出した入力項目のテストデータを機能モデルに入力できるように変換する入力模擬モデルを作成する。機能モデルで統合ソフトウェアの一部の機能のみをモデリングしている場合は、自動生成コード以外で実装される機能を入力模擬モデルで補完する必要がある。このとき、必要に応じて製品のハードウェアに搭載された回路もプラントモデルとして入力模擬モデルに追加し、システムレベルの信号を取り扱えるようにすることで、テストシナリオの設定を効率的に行うことができる。

(3)テスト入出力スタブ関数の作成

2.4節(1)で抽出した入出力項目のテストデータを統合ソフトウェアに入力、または出力として取り出すためのスタブ関数を作成する。入力用スタブ関数は、テスト入力模擬モデルが出力するテストデータを関数の戻り値や引数に渡されるポインタなどを介して呼出し元にテストデータが伝達されるよう実装する。出力用スタブ関数は、呼出し元から引数で渡された値や、静的変数の値を出力用変数に格納し、WinAMSに出力して期待値と比較できるように実装する。

(4)ファイル出力モデルの作成

ファイル出力モデルでは、手順(1)で抽出した入出力項目について、入力模擬モデルと機能モデルを実行するMILSの信号と、SPILSで実行するテスト入出力スタブ関数を含む統合ソフトウェアの入出力変数を結びつける。表3にMILSの信号とSPILSの変数との対応を示す。

表３．MILSの信号とSPILSの変数の対応

MILSの信号	SPILSの変数
入力模擬モデルの出力信号	入力用スタブ関数にテストデータを入力する変数
機能モデルの出力信号	出力用スタブ関数に入力されたデータを格納する変数

(5)テストデータの作成

2.4節(2)、(4)が完了したMILSを使用し、WinAMSに読み込ませるCSVファイルを作成する。CSVファイルはテストシナリオ毎に下記手順で作成する。

(a)テストシナリオに基づき、MILSのテスト入力模擬モデルに信号を設定または読み込ませる。

(b)MILSでシミュレーションを実行し、CSVファイルを出力する。

(6)テストの実行

2.4節(3)で作成したテスト入力スタブ関数を含む統合ソフトウェアと2.4節(5)で作成したCSVファイルをWinAMSに読み込ませてテストを実行する。テスト手順を以下に示す。

(a)統合ソフトウェアのソースファイルにテスト入出力スタブ関数を定義したソースファイルを追加してビルドし、テスト用の統合ソフトウェアを作成する。

(b)WinAMSにテスト用の統合ソフトウェアとCSVファイルを読み込ませる。

(c)統合ソフトウェアの元の関数呼出しをテスト入出力スタブ関数に置換する設定を行う。

(d)WinAMSでテストを実行する。テスト結果はCSVファイル及びHTMLファイルに出力される。

３．SPILSテストの実例

SPILSを用いたテスト手法の有効性を確認するため、当所で開発しているプラグインハイブリッド自動車(以下PHEV)用DC/DCコンバータのソフトウェアテストに適用し、評価を実施した。当該製品は、新たにモデルベース開発を導入して開発を進めており、評価対象として適当と判断した。

3.1　製品の概要

DC/DCコンバータは、PHEVの走行用バッテリーの電圧を補機用12Vバッテリーの電圧に降圧して供給する装置である。上位コントローラからCAN通信やハードワイヤ信号で指令を受け取り、補機用12Vバッテリーの充電制御、DC/DCコンバータのフィードバック制御、故障診断などを行う。図4に製品の機能構造を示す。これらの機能のうち、マイコンのハードウェア制御を除く処理を機能モデルとしてモデリングしている。

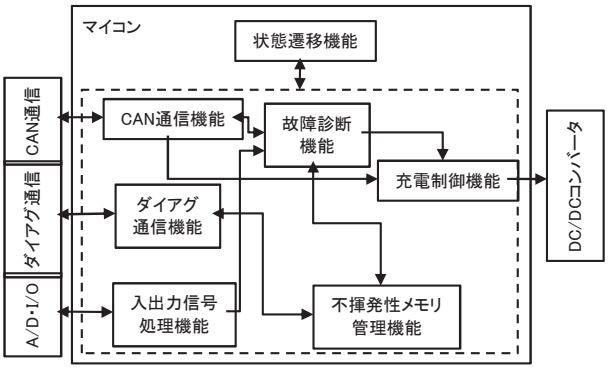

図４．DC/DCコンバータ機能図

3.2　テストの準備

(1)テストシナリオの作成

テストシナリオの作成は、時系列データを視覚的に設定できるMATLAB/SimulinkのSignalBuilderブロックを用いて行う。図5にSignalBuilderブロックと時系列データ設定画面を示す。

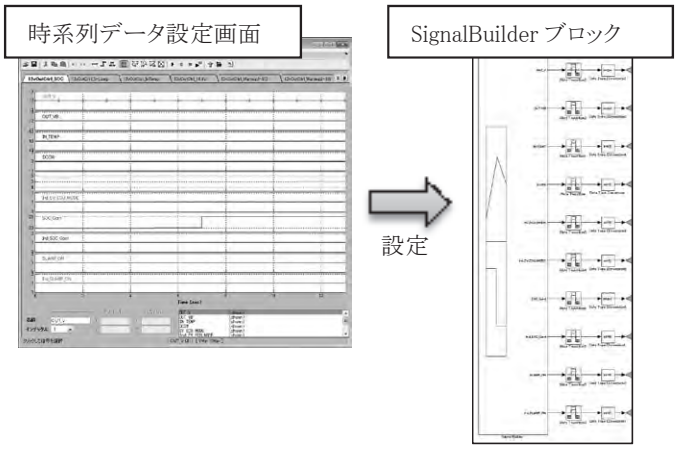

図５．テストシナリオ作成

(2)期待値の作成

MILSで機能モデルを実行することにより期待値を得る。機能モデルには12Vバッテリーの充電を実現するための状態遷移、充電制御、故障診断の一部の機能を盛り込んだ。図6にCSVファイル出力用モデルとCSVファイルの出力結果を示す。CSVファイルは行が時間軸、列がテスト入出力項目である

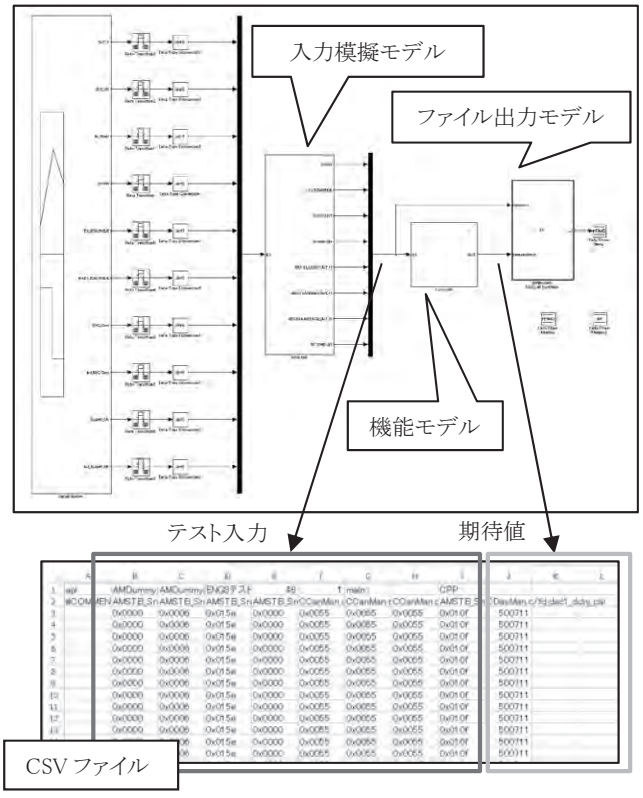

図６．期待値の作成

(3)WinAMSテスト環境

2.3節(2)表2に示した統合ソフトウェアに対する変更内容において、変更が必要になった処理を示す。

(a)無限ループ

マイコンのタイマで所定時間経過をポーリングする処理が無限ループとなっているため、比較を削除したスタブ関数で置換することで無限ループを回避した。

while構文の条件式でタイマステータスレジスタのビットテストを行っていたが、条件式をコメントアウトすることで無限ループを回避し、プログラムの実行を阻害しないようソースコードを変更した。

(b)ハードウェア入出力処理

DC/DCコンバータの入出力電圧/電流及びパワー素子の温度をマイコンのA/Dコンバータで計測している。A/D変換結果をテスト入力として統合ソフトウェアに入力できるよう、当該処理をスタブ関数に置き換えた。

A/D変換結果を関数の戻り値として呼び出し、元に返すインタフェースになっていたため、A/D変換結果レジスタの代わりに、追加した入力変数を戻り値として返すことで、SPILSから任意のA/D変換結果を入力できるようソースコードを変更した。

(c)テスト開始/終了位置の定義

テストデータは、main関数内の処理先頭で入力し、タスク処理実行後にテストデータを出力する。スタブ関数内にダミー変数の書き込み処理を記述することで、統合ソフトウェアを変更することなくWinAMSに入出力タイミングを指示できる。

main関数の処理先頭で入出力処理を実行する関数が呼び出されていたため、当該関数のスタブ関数を作成し、スタブ関数内でダミー変数の書き込み処理を行うようソースコードを変更した。

3.3　テストの実施

(1)テストシナリオの選定

DC/DCコンバータの機能のうち、DC/DCコンバータの内部温度に応じて12Vバッテリーの充電電圧を制御する機能に対してテストを行った。テストシナリオは内部温度を高温から低温に変化させると、制御仕様に応じて出力電圧が上昇する内容である。テストシナリオに対し、MILSで得られた入力データと期待値を図7に示す。

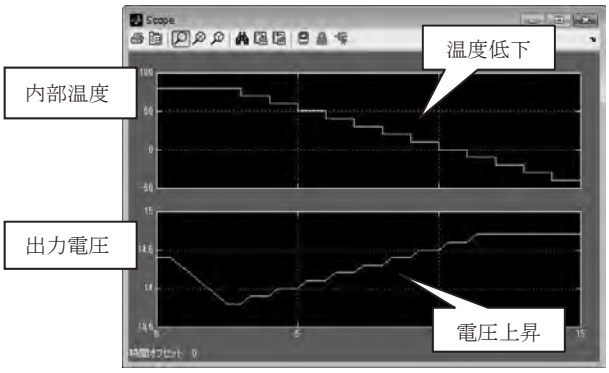

図7．テストシナリオ

(2)テストシナリオの実行

3.2節(2)で作成したCSVファイルをWinAMSに読み込ませて選択し､｢シミュレータ起動｣ボタンを押すとシミュレーションが実行される。シミュレーション完了後、テスト出力データと期待値の一致確認及び合否(OK/NG)判定が自動的に行われる。WinAMSのテスト結果画面を図8に示す。

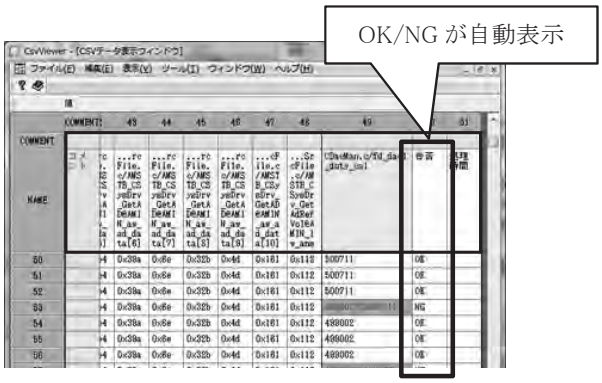

図8．WinAMSテスト結果画面

(3)テスト結果の確認

3.3節(1)で選定したテストシナリオに基づいてテストを実施し、WinAMSでOKの判定が得られた。出力ログについても、全ての入出力項目で機能モデルの出力と統合ソフトウェアの出力が一致することを確認した。これにより、今回構築したテスト環境を用いてソフトウェアテストを実行できることが実証された。

4．改善効果

今回の評価では、テスト環境の構築や発生した課題の対応も含めて作業を行ったため、テスト工数に関するデータは不足しているが、実開発に組み入れた場合の改善効果を見積もった。テスト手法をHILSからSPILSに変更することで、1項目当たりのテスト工数が約50%削減できる見込みである。ソフトウェアテストで行う全てのテスト項目のうち、約86%がSPILSでテスト可能なことから、従来のHILSのみによるテストから、HILSとSPILSを併用するテストに見直すことで、テスト工数が全体で43%程度削減できる見込みである。図9に改善効果の見込みを示す。

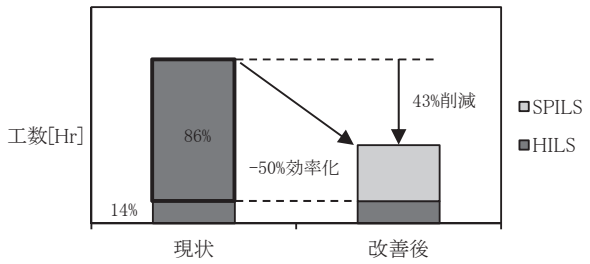

図9．改善効果見込み

従来のテスト手法(実機またはHILS)は、テスト環境の構成がマイコン以外のハードウェアに依存するため、製品特性によってテスト環境や手法が異なっていた。しかし、今回構築したSPILSテスト環境は製品特性に左右されず、テスト環境がマイコンシミュレータ内で完結するため、全ての製品に適用可能である。現在、他の製品に展開すべく、実開発における改善効果を測定中である。

5．今後の課題と応用

SPILSテスト環境の構築と、DC/DCコンバータのソフトウェアテストに適用した結果、明らかになった課題をまとめる。これらの課題については実開発に適用する中で対策を進め、テスト手法として確立させていく。

(1)タイミングの同期

機能モデルの抽象度によっては、MILSとSPILSでテストデータの入出力タイミングを完全に一致させることが難しい。テストに使用する機能モデルは、統合ソフトウェアの実行タイミングに合わせて、入出力タイミングを調整するなどの加工が必要である。

(2)テストの実行速度

WinAMSのシミュレーション速度が実マイコンよりも遅く、テストシナリオのデータ長によってはテスト結果が得られるまでに時間が掛かる。テストデータを変化点のみに間引くなど、改良が必要である。

(3)機能モデルの不具合検出

機能モデルにモデリングミスなどの不具合があっても、自動生成されるコードを含む統合ソフトウェアの出力は期待値と一致するため、不具合を検出することができない。ソフトウェア要件定義の段階で、機能モデルがソフトウェア要件を満たすことをMILSを用いて事前に検証し

ておく必要がある。

本テスト手法は、実マイコンを使用せずPC上で完結するため、自動化が可能である。また、実機やHILSよりも高精度なテストが可能なため、作業者や環境に依存しない再現性の高いテストが可能である。このため、回帰テストへの応用が期待できる。

今回は、ソフトウェアテストを効率化する目的でSPILSを用いたテスト環境を構築したが、本テスト環境はソフトウェア統合テストにも適用が可能である。特に自動車分野向けの機能安全規格であるISO26262では、ソフトウェア統合テスト環境としてSPILSが推奨されており、更なる適用範囲の拡大が期待できる。本テスト環境は、モデルベース開発を取り入れている製品では汎用的に使用できるため、他機種への展開を視野に、課題の対応を進め、テスト手法として確立させていく。

6. むすび

実際にSPILSを用いたテスト環境を構築し、実製品のソフトウェアテストに適用できることを実証した。

現在、当所ではAutomotive SPICEに沿ったプロセス改善を進めている。今回構築したSPILSを用いたテスト環境を実開発に組み入れるには、テスト戦略やテスト仕様書などの文書様式を含め、プロセス面での整備が不可欠である。モデルベース開発を軸とした設計手法及びテスト手法の構築とともに開発プロセスの構築を進め、更なる作業効率とプロセス品質の向上に取り組んでいく。

最後に本テスト環境の構築にあたり、技術的な助言や環境を提供頂いた関係者の皆様には、心より感謝を申し上げる。

執筆者紹介

前田　頼正　　マエダ　ヨリマサ
2003年入社。主にEV/PHEV補機のソフトウェア開発と設計支援に従事。現在、三田事業所技術第2部技術第1課、兼開発部開発3課グループリーダ。

竹内　康尊　　タケウチ　ヤスタカ
2011年入社。主に車載充電器のソフトウェア開発に従事。三田事業所開発部開発3課、兼技術2部技術第1課。

杉原　依理未　　スギハラ　エリミ
2016年入社。主にバッテリーマネジメントシステムのソフトウェア開発に従事。現在、三田事業所技術第2部技術第1課。