モデルベース開発とコード解析を用いた組込みソフトウェアの開発

（先進的な設計・検証技術の適用事例報告書 2015年度版より編集部要約）

本事例は、アルプス電気株式会社（以降、アルプス電気）の自動車向け組込みソフトウェア開発における、「設計・検証コストの改善」事例である。
自動車の高機能化・電子制御化に伴い、一台の自動車に搭載されるECU（電子制御ユニット：Electronic Control Unit）の数は年々増加し、また要求される機能も複雑化している。ECUの増加に伴い、実現するソフトウェアにおいては、ソースコード量の増加やECU間のネットワーク制御が必要になり、ソフトウェアの品質が全体の性能・信頼性に大きく影響を与えると共に、設計・検証コストが増大する課題が表面化してきた。本事例では、これらの課題を解決するために「モデルベース開発」と「コード解析」を適用した独自の手法であるMDD（Model Driven Development）にTDD（Test Driven Development）の思想を取り入れ、モデルベース開発とテスト駆動開発を融合させプロセスの洗練化を図ったものを活用した。その結果、従来開発に比べて開発工程が減ることでコストが削減でき、視覚的に理解しやすいモデルを検証することで、検証効率と品質を向上することができた。また、コード解析では、市販コード解析ツールと自社開発ツールを組み合わせることで、人手で行っていた検証作業をツールに置き換え、スキルによらない検証が行え、検証コストの削減につなげることができた。

1　技術・手法を導入した理由や経緯

1.1　モデルベース開発（MBD：Model Based Development）

年々加速度的に大きくなる車載ソフト規模の市場要求と、開発能力の関係を図１に示す。

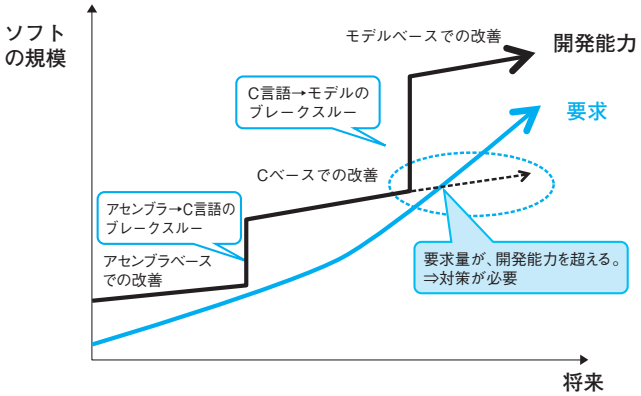


図１　モデルベースの開発の必要性

開発現場での地道な改善活動により開発能力は毎年少しずつ向上しているが、それだけでは近年高まるソフトウェア開発の要求量と要求スピードについていくことができない状況となってきている。かつて車載ソフトの開発現場ではアセンブラベースの開発からC言語ベースの開発に移行することで大きな開発能力向上のブレークスルーを経験した。しかし、現在ではもはやC言語ベースの改善の積み重ねでは市場要求についていくことは困難になりつつある。C言語ベース開発から移行する第２のブレークスルーとして、制御系を含む製品開発の手法として注目されている「モデルベース開発」を設計に適用することにした。

一般的にモデルベース開発の目的は、「複雑なシステムを効率的に開発すること」で、下記のような利点がある。

① 仕様書の明確化と再利用性の向上
② シミュレーションをしながら仕様を決定できる
③ 仕様書から自動でコードが生成されるために、バグが入りにくい
④ 制御対象と制御装置を同時に開発できる

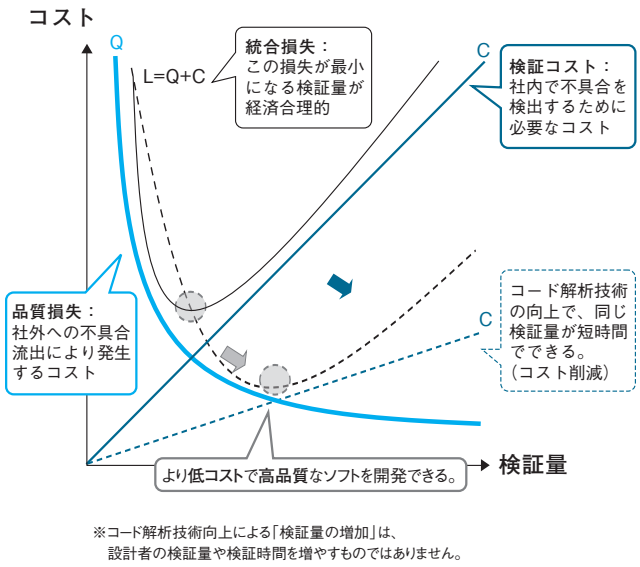
モデルベース開発の手法はソフトウェアの開発規模の増大にも対応できる手法であり、モデルの作成ができると、ソースコードが自動生成されるので、プログラミングのミスが防止でき、品質の向上が見込めると共に生産性の向上も期待できる開発手法と言える。

1.2 コード解析

設計においてはモデルベース開発を適用したが、検証については、下記の理由から各種ツールを利用した「コード解析」を実施することとした。

① 人による検証からツールによる検証への置き換え
② 顧客要求に応じた各プロジェクトのサポート
③ ソフトウェア品質の可視化

コード解析を実施することで、今後増え続けるであろう検証負荷を軽減し、開発効率及び開発品質を向上する。図２に示すように、コード解析の目標は、統合損失（品質損失＋検証コスト）を最小にする点を求めると共に更にそれを小さくすることである。



※コード解析技術向上による「検証量の増加」は、
設計者の検証量や検証時間を増やすものではありません。

図２　コード解析の目標

2 スムーズな導入と活動定着のための事前準備や工夫

2.1 ワーキング活動と開発体制

MBD開発に興味のある有志を中心にワーキンググループ（WG）を構成し、２週に１回程度の頻度で定期的に勉強会や討論会（いわゆる小集団活動）でツールのカスタマイズなどを実施した。MBDのWG活動については既に10年以上継続している。WG活動の結果として、従来のハンドコーディングと同等のコードサイズを実現するMBD開発ができるようになった。

また、コード解析においても同様にコード解析WGを構築し手法の学習や習得に努めてきた。活動は２週に１時間程度の周期で実施してきた。

2.2 開発工程ごとの解析項目

解析種別により複数の解析ツールを用途に応じて使い分けている（表１）。また、図３に示すように、コード解析（専任者）が使用する解析ツールと開発工程、コード解析（設計者）が使用する解析ツールと開発工程を、それぞれ決めている。

表１　コード解析ツール一覧

解析種別	検証項目の例	ツール
ランタイムエラー	配列オーバーフロー、ゼロ割など	Polyspace®
MISRA-C	MISRA-C 2004対応	QAC©+M2CM
ソフトウェアメトリクス	サイクロマティック複雑度、実行行数	QAC©+自作
コーディングルール	社内コーディングルールへの適用	QAC©+自作
ソフトウェア構造	タスク間インターフェースなど	Imagix4D
コードクローン	コードクローン率	CCFinder

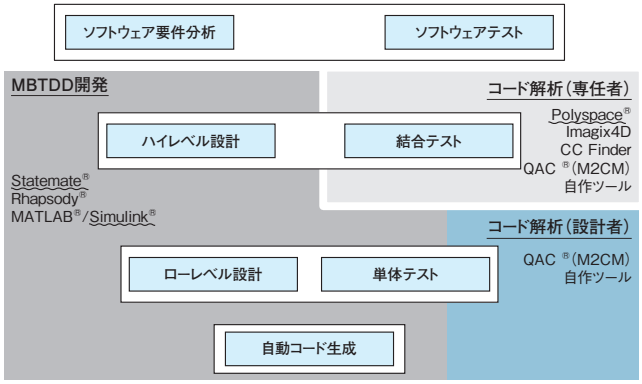


図３　開発工程ごとの専任体制

3 効果測定の方法とその結果

3.1 モデルベース開発

(1) 適用状況

現在、アルプス電気では適材適所でモデルベースの開発を実施している。適用対象事例を表２に示す。適材適所のモデルベース開発では、モデル仕様に基づいて開発プロセスを構築する手法である広義のMBD（Model Based Development）と、組込み制御システム開発の狭義のMBDがあり、車載用組込みソフトで狭義のMBDは全体の20%程度にしか適用できない。このため、残りの80%についてはMDD（Model Driven Development）で開発し効率化を目指している。

表2　モデルベースの開発状況

名称		適用対象項目	手法	使用ツール
広義のMBD	狭義のMBD	制御系アルゴリズム開発	制御ブロックモデリング	MATLAB®/Simulink®
	MDD	アプリケーション開発（製品機能実装部）	構造化モデリング	Statemate®
		プラットフォーム開発（ハードウェア制御部）	オブジェクト指向モデリング	Rhapsody®

(2) MBTDD開発

アルプス電気ではMDD（Model Driven Development）にTDD（Test Driven Development）の思想を取り入れ、モデルベース開発とテスト駆動開発を融合させてプロセスの洗練を行うMBTDD（Model Based Test Driven Development、アルプス電気の造語）開発を実施している。MBTDD開発では、MDDのモデルシミュレーションの代わりに、モデルから生成されるコードについて直接テストを実施する。ここにTDDの思想を取り入れテストとモデルの洗練を繰り返し実施する。コードではなくモデルを洗練することで本当の意味での設計の洗練になる。

図4に人手によるコーディングをしていた従来プロセスとMBTDDプロセスの比較を示す。

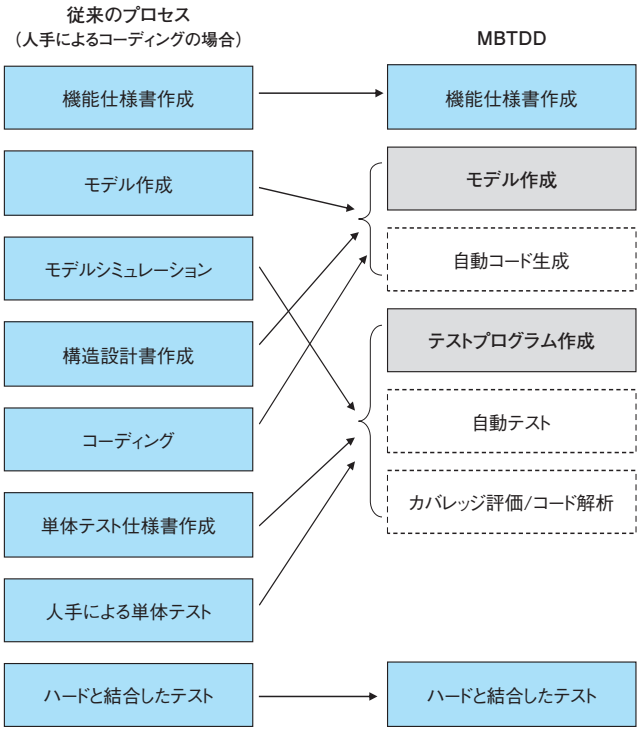


図4　従来のプロセスとMBTDDプロセスの比較

従来プロセスでもモデルは使用していたものの、その主な目的はシミュレーションによるモデルの動作検証であった。モデルシミュレーションで動作検証が済んだモデルをコーディングするために必要な情報を記述した構造設計書を作成し、それに基づいて人手でコーディングしていた。また単体テストも単体テスト仕様書を作成し、人手による単体テストを実施していた。

MBTDDでは、モデルの中に従来の構造設計の情報を埋め込み、最適なコードが生成されるようにカスタマイズしたツールのコード生成機能を用いてモデルからボタン一つでコードを生成している。また単体テスト仕様書をプログラム化し実施を自動化した。

これらの改善によりモデルシミュレーション工程をなくし、モデルができたらボタン一つでコード生成し単体テストプログラムを走らせることで従来のモデルシミュレーション、コーディング、人手による単体テストの工程をなくすことができた。また単体テストの実施を自動化し何度でも簡単に実施できるようにしたことに合わせ、この工程で簡単にテストカバレッジ及び設計者自身によるコード解析もできるような環境を整備した。結果としてモデル作成工程とテスト工程を行ったり来たりしながらモデルの洗練と単体テストの洗練を行うようになった。

(3) MATLAB®/Simulink®の活用事例

MATLAB®/Simulink®をカーナビディスプレイの開閉制御に活用した例である（図5）。実機での動作検証前にMATLAB®/Simulink®でシミュレーションし、システム成立性の確認を行った。

また、モデルとコード解析を使った作業（図8）を繰り返すことで、ロバスト性が高く、高品質のソフト開発を行うことができた。

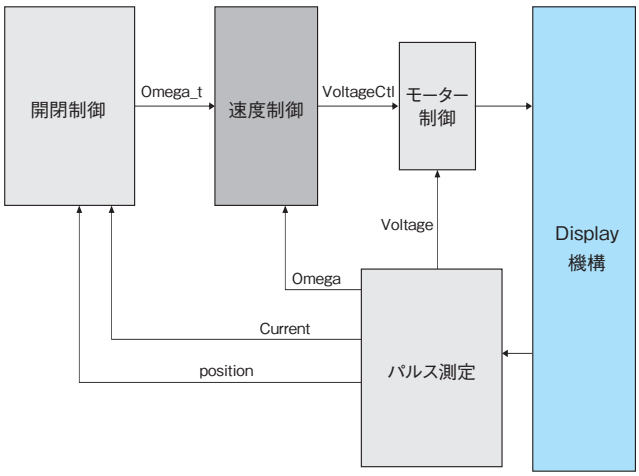


図5　MATLAB®/Simulink®の活用事例

3.2　コード解析

3.2.1　コード解析ツールの選定

アルプス電気では複数の市販コード解析ツールと自社開発ツールを組み合わせて使用しており、より精度の高いコード解析を達成するため、市場のコード解析ツールのベンチマーキングを実施しツール選定を行っている。以下にその評価方法と結果を示す。

(1) 評価方法

解析ツールの選定では以下の５つのステップで評価方法を定義している。

① 社内ソフトウェアでよく実装してしまう不具合を選別
　⇒過去の不具合について洗い出しを行い、類型化を図る
② 不具合がない３種類のソースコードを用意
　⇒３種類のソースコードを使い解析ツールを評価
　● Statemate®ツールによる自動生成コード
　● Rhapsody®ツールによる自動生成コード
　● MATLAB®/Simulink®による自動生成コード
③ 各ソースコードに、選別した不具合を埋め込む
　⇒②のソースコードに意図的に不具合を挿入
④ 社内外の解析ツールで不具合埋め込みコードを解析
⑤ 解析結果を分析し、結果を比較

(2) 解析ツールの評価基準

理想的なコード解析ツールとは、①すべての不具合を検出すること（＝不具合検出率が高い）、及び、②正しいコードを誤って不具合としない（＝有効警告率が高い）ことである。評価基準を表３に示す。

表３　評価基準

評価項目	計算式	基準
不具合検出率	検出された埋め込み不具合の数/埋め込み数	高いほうが良い
有効警告率	埋め込み不具合に対する警告数/総警告数	高いほうが良い

(3) 評価結果

図６に市販のコード解析ツール（ツールA ～ F）と自社開発ツール（ALPS）の評価結果を示す。自社開発ツールは、とくに重要な不具合検出率が一番高く、有効警告率も中間に位置しており、現在のツール選定の妥当性が確認できた。また、同時に、本評価結果を元に改善点も明確になり、改善することができている。

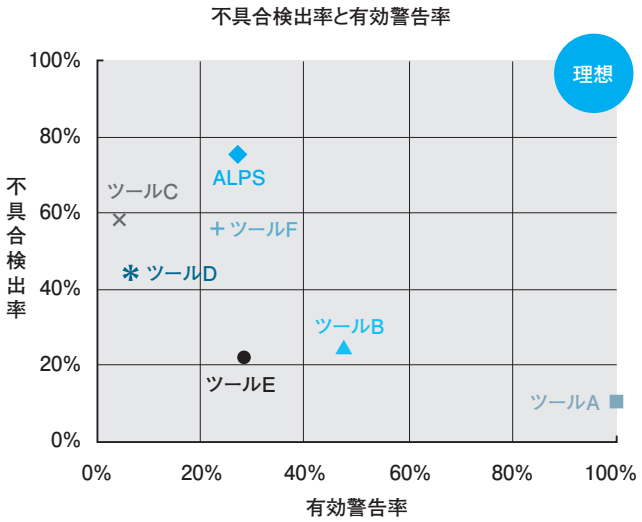


図６　解析ツールの評価結果

3.2.2　ソフトウェア品質の可視化

(1) ソフトウェア品質モデル

ISO/IEC 9126-1（Software engineering - Product quality - Part1: Quality model）で定義しているソフトウェア品質モデル（表４）の中から「信頼性、効率性、保守性、移植性」の４つの品質特性を使いソフトウェア品質を評価している。

表４　ソフトウェア品質モデル（ISO/IEC 9126-1）

品質特性	品質副特性	主な内容
機能性	合目的性	目的から求められる必要な機能の実装の度合い
	正確性	
	相互運用性	
	セキュリティ	
	標準適合性	
信頼性	成熟性	機能が正常動作し続ける度合い
	障害許容性	
	回復性	
	標準適合性	
使用性	理解性	分かりやすさ、使いやすさの度合い（いわゆる「使い勝手」、「使いやすさ」、「操作性」の概念）
	習得性	
	運用性	
	魅力性	
	標準適合性	
効率性	時間効率性	目的達成のために使用する資源の度合い
	資源効率性	
	標準適合性	

品質特性	品質副特性	主な内容
保守性	解析性	保守（改訂）作業に必要な努力の度合い
	変更性	
	安定性	
	試験性	
	標準適合性	
移植性	環境適応性	別環境へ移した際そのまま動作する度合い
	設置性	
	共存性	
	置換性	
	標準適合性	

(2) ソフトウェア品質の可視化の目的

ソフトウェア品質について、４つの品質特性を５段階評価したものをレーダーチャートで可視化している。これにより次の効果が得られる。

① ソフトウェア品質状況を数値化して定量的に把握する
② 定量的データに基づき、品質の良否を判断できる
③ 品質が悪い場合の改善ポイントが明確になる

また、その結果として、次のような点で品質の良いソフトウェア開発が可能になる。

① 不具合が除去されている
② 仕様変更に容易に対応できる
③ 再利用が可能である

3.2.3　コード解析WGの成果

各プロジェクトの品質状況を可視化し、モニタリングすることで、早期に適切な対処を行っている（図７）。

従来レビューで活用していたチェックリストを見直し、コード解析で代用できるものを削減した。その結果、30％以上のチェック項目がコード解析で代用できることが分かり、レビュー時間の削減に大きく貢献した。また、レビューからコード解析に検証手法を変えたことで、設計者のスキルに依存しない、安定した検証が短時間で行えるようになった。

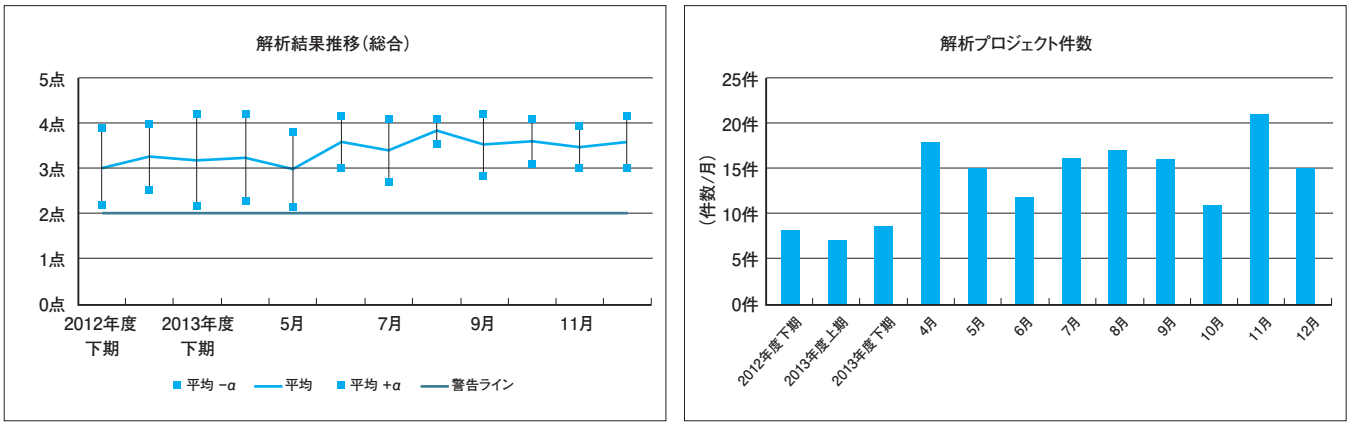


図７　品質状況のモニタリング

3.2.4　複数の市販解析ツールと自社開発ツールの活用

市販ツールには、それぞれ得意な解析分野があるが、一方でアルプス電気のソフト開発では必要としない機能もある。アルプス電気では各市販ツールにおいて解析のためのパラメータの設定をカスタマイズしている。更に独自のフィルターをかけることで、効率的な解析を可能としている。また、市販の解析ツールのカスタマイズで対応できないものはツールを自作し活用している。

3.2.5　自動生成におけるコード解析の必要性

モデルベース開発は、モデル自体の文法チェックや一貫性のチェックが実施でき、また、視覚的に理解しやすい開発手法である。しかし、自動コード生成をするためにはプログラム言語に近い記述も必要であり、人為的なミスやスキル不足による間違いがそのまま自動で生成されたコードに反映されてしまう。そのため、自動生成コードでもコード解析で不具合の検証を行う必要がある（図８）。なお、制御系アルゴリズム開発にはMBTDD開発を適用していないため、シミュレーション（図８）が必要だが、MBTDD開発が適用できるアプリケーション開発やプラットフォーム開発では不要になる。

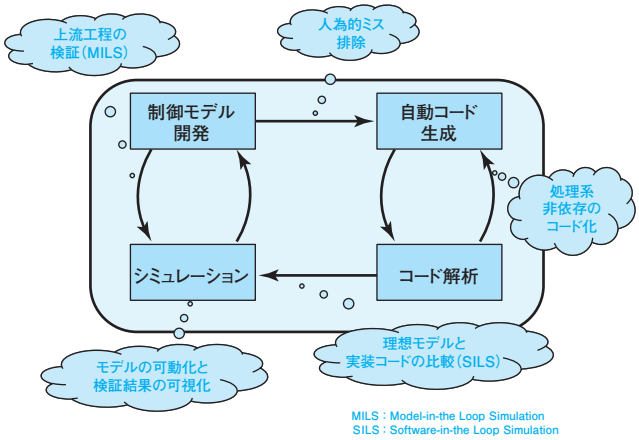


図8　モデルとコード解析を使った開発(MATLAB®の例)

4 技術や手法の導入後の改善活動、今後の課題

MBD開発導入後は、モデルの中に設計情報を書き入れているので、あいまいになりがちな設計書を作成する必要はなくなった。ソースについても自動生成されるので、コーディング作業は不要になった。今後、モデルベース開発では、モデル用のライブラリの充足、及び、マイコン依存部やデザインパターンなどの実装を簡単にするフレームワークを整備することで、再利用率を上げる予定である。コード解析では、有効警告率を25%から40%程度まで向上させることを目標に、Polyspace®、Imagix4Dなどのツールの解析設定のカスタマイズ、及び、自作のツールの改良を実施する。

5 結果と考察

(1)品質について

新手法を導入する際に、品質の目標をバグ件数半減としていたが、結果は表5に示すように大幅に改善できた。バグの数が減ったことと、バグが出ても原因の特定や対策ができるようになりバグ対策の仕事が劇的に減った。打ち合わせの多くを占めていたバグ対策が激減し、モデルやテストの洗練に多くの時間を割く文化が育成された。

表5　品質の結果

プロジェクト	バグ件数
A	約1/4
B	約1/3

(2)生産性について

新手法を導入する際に、開発速度の目標を2倍としていたが結果は表6に示すように改善できた。

表6　開発速度の結果

プロジェクト	実績
A	約2.5倍
B	約2.0倍

従来レビューで活用していたチェックリストを見直し、コード解析で代用できるものを削減した。その結果、30%以上のチェック項目がコード解析で代用できることが分かり、レビュー時間を大きく削減できた。

(3)開発工程ごとの専任体制による効果

開発工程ごとに専任者によるコード解析の体制を構築した(図3)ことにより、結合テスト段階で、専任者による高度な解析を行うことができ、ソフトウェアの信頼性を確保できた。

6 まとめ

車載向けECUソフトウェアの開発において、モデルベース開発とコード解析を導入することにより、品質向上と生産性向上の目標を十分達成することができた。具体的には、Statemate®、Rhapsody®、MATLAB®/Simulink®などのモデルベースツールをカスタマイズしTDDと組み合わせて使用することと、QAC®、Polyspace®、Imagix4D、自作ツールなどのコード解析ツールを適材適所に使用することで、従来の人手によるコーディング及び人手による検証に比べ、劇的に開発効率と品質を改善した。また、レビューからコード解析に検証手法を変えたことにより、設計者のスキルに依存しない、安定した検証が短時間で行えるようになった。

参考文献

[1] アルプス電気株式会社 宇治川 尚悟：車載向けECUのコード解析とモデルベース開発への取り組み MATLAB EXPO 2013

[2] 先進的な設計・検証技術の適用事例報告書 2015年度版
http://www.ipa.go.jp/sec/reports/20151118.html