ソフトウェア・プロダクトライン開発とテストプロセス自動化による工業化促進

黒岩　正司
富士設備工業株式会社

キーワード
プロダクトライン開発、フォーマルメソッド（形式手法）、
テストベクタ自動生成、要件からテストへのトレーサビリティ

概要

近年企業は，大規模・複雑さを増すソフトウェア開発のスピードを加速し，生産性向上によるコスト削減，高い品質を確保するために，あらゆる開発成果物を体系的に管理して再利用するといった工業化策である，プロダクトライン開発への取り組みが戦略として求められている．
また高信頼性システムにおいて，ソフトウェア開発プロセスの７０%をも占めるテストの多くは，手作業に頼っている．さらにこれらテストは，デザインレビュー・コードの静的解析・ユニット/インテグレーション/システムテストなど，開発ライフサイクルの各フェーズで分断されており，要件やコードなどの修正・変更があった場合の追跡が極めて困難となっている．

プロダクトライン開発において，テストの自動化と再利用を促進すること，また要件からのトレーサビリティにより，全体を可視化して管理を効率化させ，迅速かつ正確に検証できる仕組みが必要である．
これらの課題に対し，開発組織は独自手法や各種ツールによる対処を試みているが，ツールの断片化により開発フェーズ間の垣根が埋まらないのが現状であり，体系的な再利用や自動化が進んでいない．

そこで現在入手可能となっている，ソフトウェア・プロダクトライン開発やテストプロセスの自動化を，支援するツールの統合を図り，ソフトウェア・プロダクトライン・ライフサイクルに於けるテストプロセス自動化の，有効性，課題などを調査した．

1. はじめに

ソフトウェア・プロダクトライン（SPL）は，“特定マーケットや業務・使命で固有の要件を満たす共通の管理された機能を共有する一連のソフトウェア集約システムであり，共通のコア資産を用いて所定の方法で開発される”と SEI（The Carnegie Mellon Software Engineering Institute）は定義している．例えば携帯電話は，電話機能，カメラ，テキスト管理，電話帳管理，インターフェイス，ミュージック機能，OS などのコア資産を持ち，新機種はこれらの多くを取り込んで，新しい機能が盛込まれる．プロダクトライン開発を支援するツールでは，あらゆる開発成果物を体系的に管理し，それら資産からバリアントごとの要求仕様書，デザインモデル，コード，テストなどが自動生成される．

テストプロセスの自動化を支援するには，要求仕様からのトレーサビリティが取れる仕組み，派生要求への対処などが，各種フェーズ間の隔たり無しに実施され，所定のプロセスが施行される仕組みが求められる．また関係者間での情報の共有に耐えるには，あらゆるフォーマットのデータに対応できることも必要となる．また，テストベクタ，ドライバーの自動生成や，実行環境ごとでコンパイル・ダウンロード・実行を自動化するテストハーネスが再利用できることでテストの効率が改善されなければならない．

今回の調査では，プロダクトライン開発で体系的に管理される資産から，バリアントごとの要求仕様書，デザインモデルを自動生成させた．このデザインモデルからソースコードと，テストベクタを自動生成させて，テストプロセス管理支援フレームワークを用いて，要件からテストに至るトレーサビリティを実現させる

ことで，ソフトウェア・プロダクトライン・ライフサイクルに於けるテストプロセス自動化の評価を試みた．

おおよその手順は，以下の通り．

要求仕様書（DOORS）は，プロダクトライン開発のフィーチャ・モデルに同期され，コア資産であるデザインモデル（Simulink）と相関される．そしてフィーチャの選択により，バリアントごとの要求仕様書，デザインモデルが自動生成される．生成されたモデル（Simulink）に対し，テストベクタの生成を試みることで，モデル上の欠陥・矛盾を解析する．テストプロセス管理支援フレームワークを用いて要件ごとにソースコードを割り振り，テストを実行することで，静的解析・テストベクタの実行・カバレッジ結果が，自動的に要件にマップされ，トレーサビリティのエビデンスを得る．

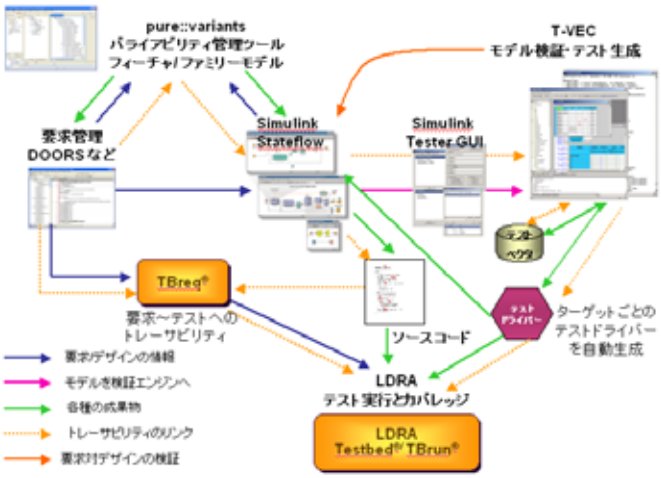

図 1. SPL とテストプロセス自動化フレームワーク

2. 各種ツール

2.1. プロダクトライン開発管理支援ツール

従来のフィーチャ・モデルベースの手法ではなく，pure::variants では製品の機能・課題をフィーチャ・モデルに，プラットフォームなどソリューションをファミリー・モデルに保存する．

ファミリー・モデルはコンポーネントで構成されて，これらの相互接続や制限，条件なども含む．ファミリー・モデル内の各コンポーネントはプロダクトライン内で 1 つ以上の機能要素に相当（クラス，オブジェクト，関数，変数，ドキュメントなど）する．

フィーチャ・モデルでは，製品ごとのフィーチャと，それらの相互関係を定義する．これらのモデルにより，プロダクトファミリー製品の全ての変動要素，共通要素が一貫して管理され，体系的に再利用できるようになる．[1]，[2]

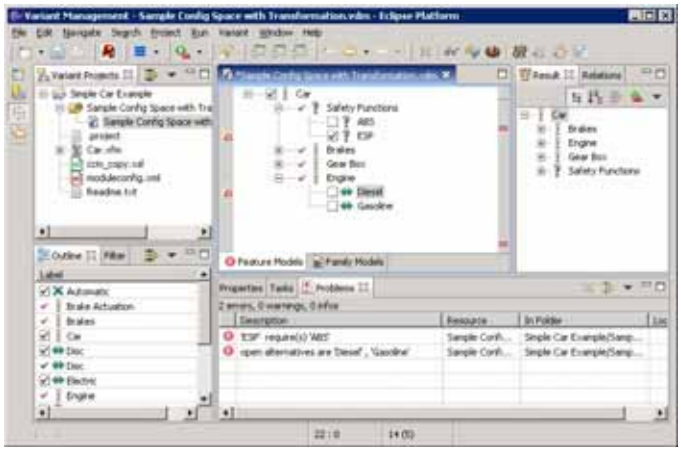

図 2. pure::variants

2.2. モデルベース検証・テストベクタの自動生成

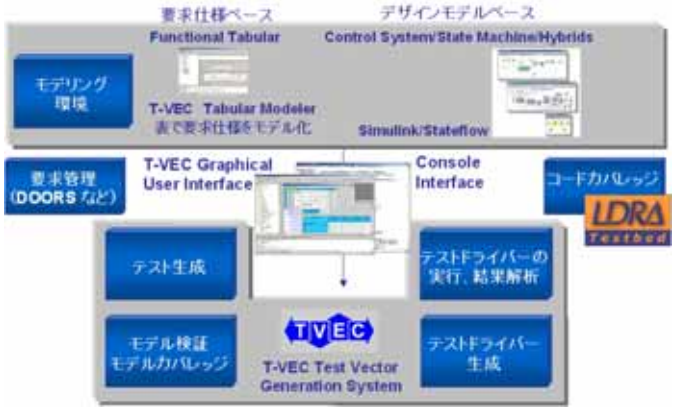

図 3. T-VEC モデルベース検証基盤

T-VEC は検証に先駆けて要求モデル（TTM）やデザインモデル(Simulink/Stateflow)を独自の階層的 Disjunctive Normal Form(DNF)に変換する．この形式に於ける各機能要求（入出力間の振舞い）の入力域に対するコンストレインツは Domain Conversion Path(DCP)として抽出される．このパスに対し，テストベクタ生成システムは，上流の制約を受けて伝播されてくる入力範囲で期待出力との組合せを生成する．この組合せ（テストベクタ）を生成できない DCP は，

モデル上の矛盾として検出される（モデル検査）．また，このテストベクタ生成機能は，フロート，ダブルなど各種データ型，リニア／ノンリニア式に対応しているため，矛盾無く生成されたテストベクタは，実システムに対する入力と期待値としてテストに用いられ，モデルに対するコードの一致性の検証が行える．T-VEC は，以下の機能で構成される．

- T-VEC Tabular Modeler（TTM）
- Simulink Tester（SL2TVEC）
- T-VEC Vector Generation System（VGS）

TTM は，要件管理を提供し，要件のモデリングをサポートする．SL2TVEC は，デザインモデルである Simulink や Stateflow を変換し解析する．VGS は，これらモデルの解析，テストベクタ生成，テストドライバー生成，テスト結果解析，結果レポート生成などを行う．

T-VEC ツールは, フォーマルメソッドを利用したモデルベース検証ツールである．高信頼性ソフトウェアの Verification & Validation をサポートする目的で開発された．航空機開発におけるガイドライン FAA/DO-178B,C に関わり，モデルベース開発とその検証についてのガイドライン制定に貢献するなど実践的に産業界で使用されるツールである．例えば，NASA のスペースシャトル引退後の次世代有人宇宙船 (Project Orion)の, 主契約企業であるロッキードマーチン社は, このプロジェクトに T-VEC の TTM 要求仕様モデルと Simulink のモデル検証機能をサブコントラクタも含めて採用することを表明している．

2.3. テストプロセス管理支援フレームワーク

LDRA ツールには，テストプロセス管理支援フレームワークである TBreq に，静的解析／カバレッジ解析を行う Testbed，そして単体テスト，リグレッションテストを自動化する TBrun が有機的に統合されている．これらテスト機能は，ホスト・シミュレーションから実ターゲット実行まであらゆる環境をサポートする．そのため，要求から実システム動作環境のテストに至る，広範なテストプロセス全ての自動化を支援し，開発者・テスト担当者は，開発初期段階で欠陥を識別して要件への照会ができる，高度に洗練された統合環境を享受できるようになる．そのため，要求仕様ベースのテストが実ターゲット・システムで行われたエビデンスが求められる，航空・宇宙・防衛システムなど高信頼性マーケットで 30 年以上に渡って採用されている．

各種機能の概要は，以下の通り．

LDRA TBreq は，DOORS などの要件管理ツールや Word・Excel などのドキュメントから各種要件をインポートし，各要件に領域を割り当て，ソースコードをマッピングする．これらの仕組みを介して，Testbed や TBrun でソースコードをテストすることで，要求仕様，デザイン，テスト仕様，構成管理，テストデータ，カバレッジなど検証結果を関連付ける作業を効率化し，これらのトレーサビリティを管理してエビデンスを残すことができる．

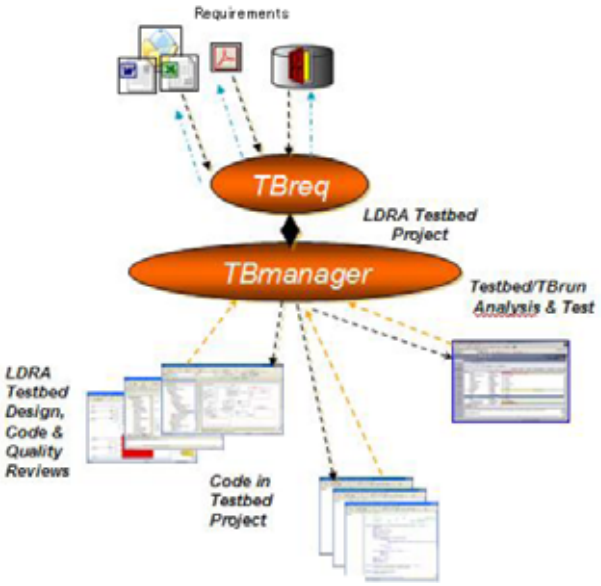

図 4. TBreq ワークフロー

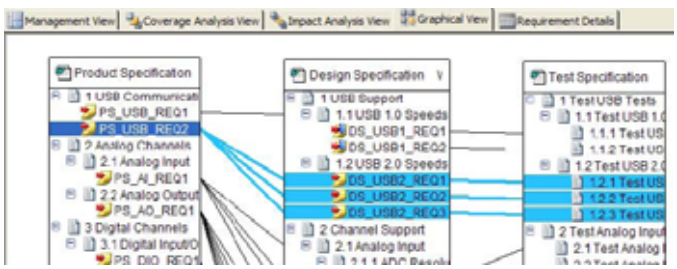

図 5. TBreq グラフィカルビュー

LDRA Testbed は，ソースコードの静的解析とカバレッジ解析を提供するツールである．静的解析では，独自の構文解析エンジンを持ち，プログラミングスタンダードチェック，複雑度解析，データフロー解析，コードのビジュアル化（コールグラフ，フローグラフ）や詳細レポートを生成する．カバレッジ解析では予めタグをソースコードに埋め込み，テストデータを実行することでテストされていない領域（未実行パスやデッドコード）を特定できる．ステートメント，ブランチ，MC/DC カバレッジなどのレベルで測定できる．

LDRA TBrun は，テストハーネス／ドライバーを自動生成し単体テストを自動化する．追加のコーディングやスクリプトは不要で，ユニット単位のテスト実行を容易にかつ効果的に行える．テストケースと結果は保存され，リグレッションテストに対する解析の自動化に利用される．ソースコードの変更箇所を自動検出し（関数の引数が増えた，型が変更されたなど），保存されているテストに対して変更が必要な部分はレポートされるため，テストデータのメンテナンスが効率よく行える．

3．SPL とテストプロセスの統合

上述で紹介したツールを用いて，ＳＰＬとテストプロセス自動化支援フレームワークとの統合を，下記要求仕様を基に行った．

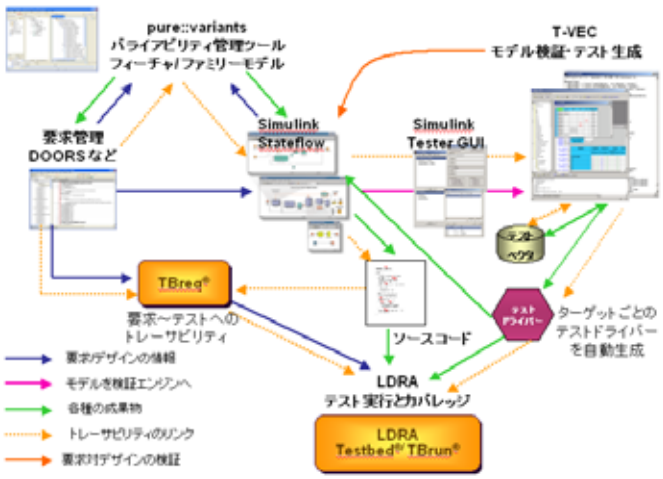

図 6. SPL とテストプロセス自動化フレームワーク

3.1. 要求仕様書とデザインモデリング

温度センサーにより温度を監視し，温度に合わせてバルブの開閉を調整するフローレギュレータ(flowRegulator)装置を考える．

- 入力：温度（範囲：0~300）
- 定数：Low : 120，High : 180
- 温度が Low より小さい時，バルブは Closed
- 温度が High より大きい時，バルブは Open し，温度が Low を下回るまで，バルブは開いている

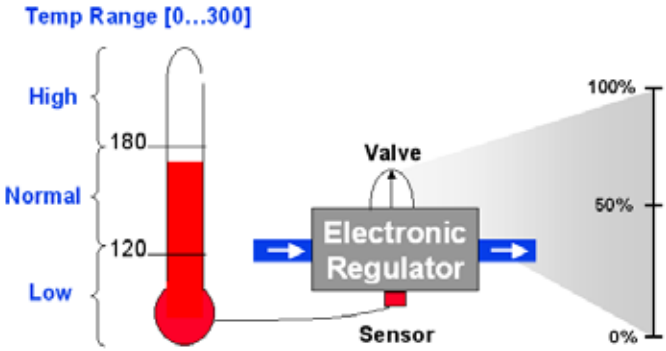

図 7. フローレギュレータ

flowRegulator モデルは，2 つのサブシステムを持つ．

- temperatureSensor
- flowControlLogic

temperatureSensor は，入力温度の値に前回値を使った処理(Y[n]=0.7U[n]+0.3Y[n-1]) を行い，その結果を-100～300℃の範囲でフィルタリングし出力する．

flowControlLogic は，temperatureSensor の値をチェックする．それにより，flowRegulator バルブの開閉が調整される．バルブが Closed の時は出力値 0%，バルブが Open の時，出力値は以下の方程式で表す．

Out=((sensorTemperature- 120) / (300 - 120))*100

これら要求仕様は DOORS で記述される．

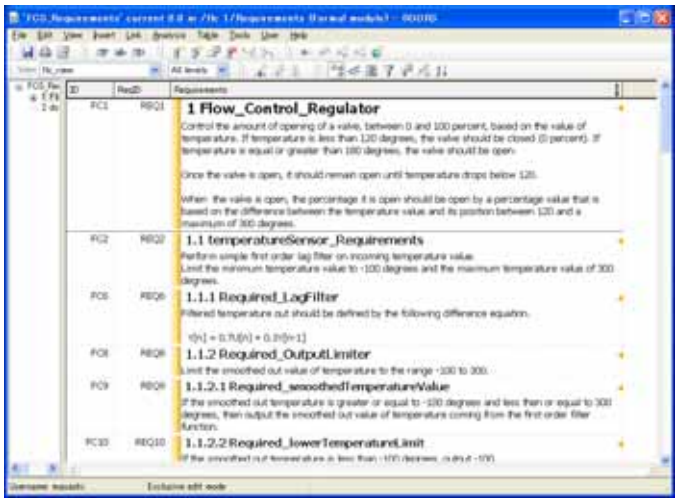

図 8. 要求仕様書（DOORS）

この要求仕様に基づき Simulink デザインモデルを作成する．その際，フィルタリング処理（Saturation ブロック or サブシステム）をバライアビリティとするモデルを作成する．

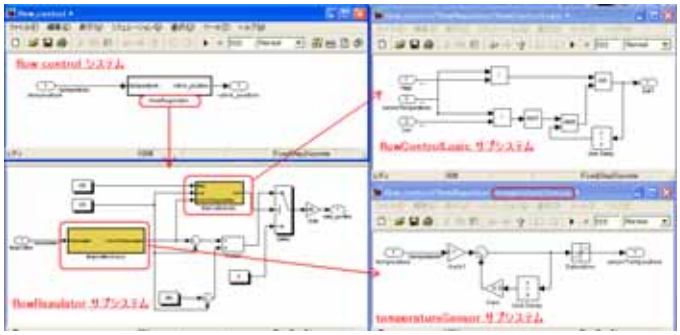

図 9. Simulink モデル（Saturation ブロック）

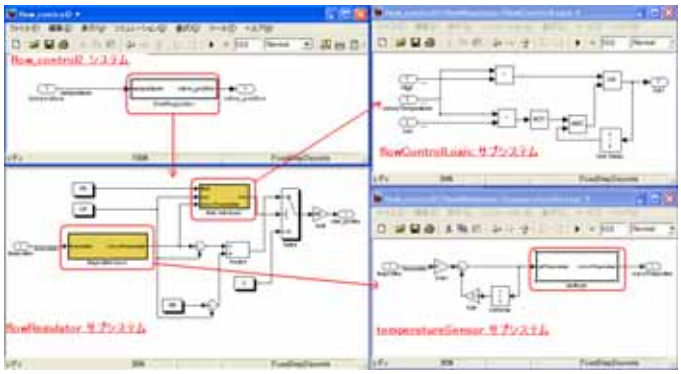

図 10. Simulink モデル（サブシステム）

また，temperatureSensor サブシステムの前回値を使った処理の有無もバライアビリティとする．

3.2. ソフトウェア・プロダクトライン

Simulink モデルは，pure::variants のファミリー・モデルにインポートされ，これらをフィーチャモデルの各バリエーションに関連付けさせて，プロダクトラインのコア資産として再利用できるようにする．ファミリー・モデル内は，サブシステムや Simulink ブロック，ブロック間の信号線として分類される．

DOORS で管理される製品ファミリーの要件モジュールは，pure::variants のフィーチャ・モデルに同期され，プロダクトラインの資産となる． DOORS 上に変更が入ればフィーチャ・モデルに反映され，フィーチャ・モデルに変更が入れば，DOORS 上にエクスポートできる．

今回の例では，Simulink モデルで実装したバリエーション（フィルタリング処理や，温度センサーの前回値処理）を，バリエーションポイントとしてフィーチャ・モデルに追加し，それらを DOORS 上に，エクスポートした．

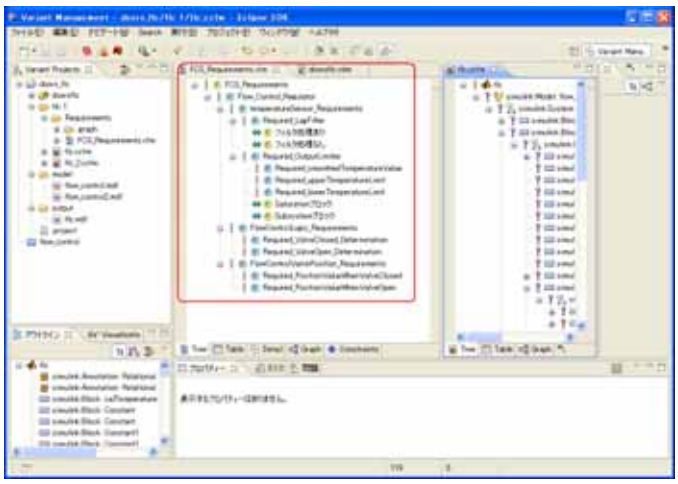

図 11. フィーチャモデル

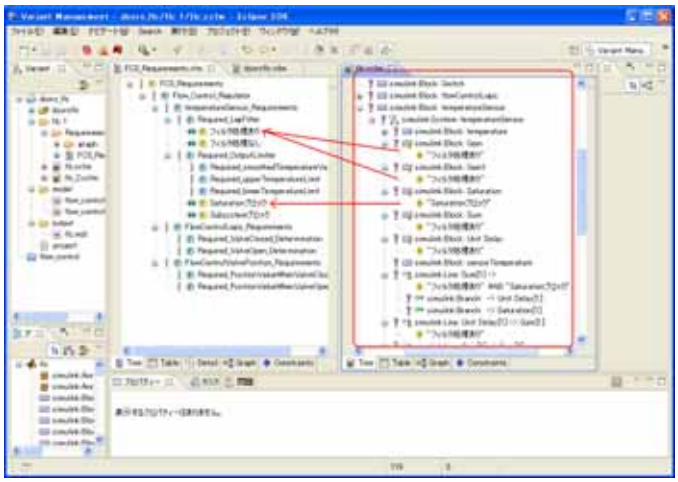

図 12. ファミリーモデル

フィーチャの選択からバリアントを設定し，“ トランスフォームモデル ” を実行することで，バリアントの要求仕様書，Simulink モデルが自動生成される． そして各要件，モデル部品が，どのバリアントに組み込まれたかのトレーサビリティのレポートが容易に取れるようになる．

今回は，“ 前回値を使ったフィルタ処理あり ”，“ フィルタリング処理として Saturation ブロックを使用する ” といったバリアントを生成させた．

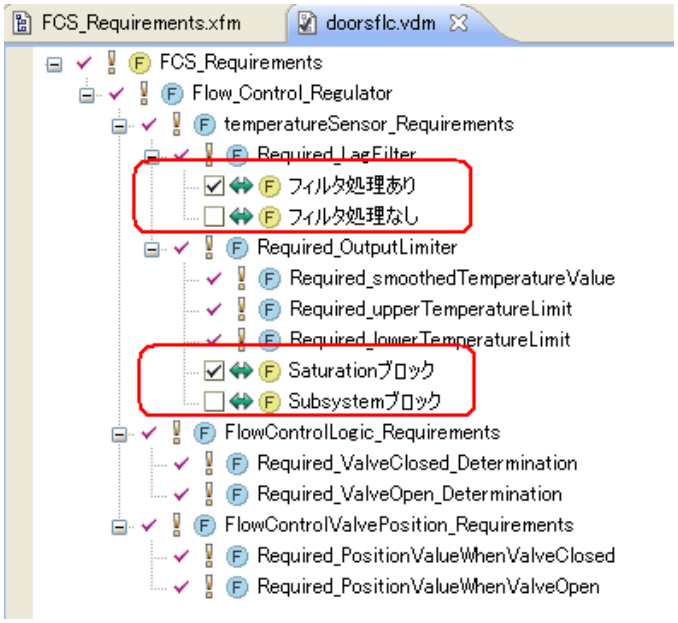

図 13. バリアントモデル

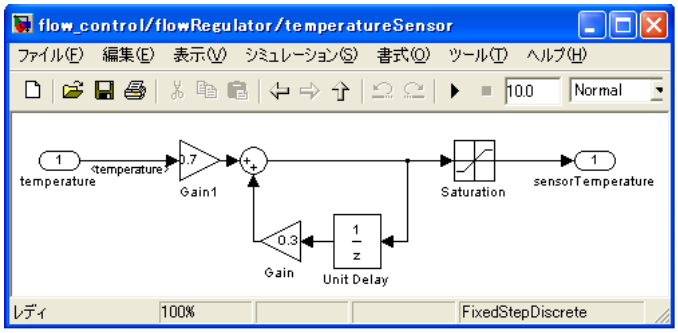

図 14. 生成された Simulink モデル

このモデルからコード自動生成ツールである RTW（Real Time Workshop）を用いて，ソースコードを生成させる．

3.3.モデル検証とテストプロセスの自動化

T-VEC のテストドライバー生成機能を用いて，Simulink モデルの検証結果から得た，入力と期待出力の組合せ（テストベクタ），モデル対ソースコードのマップ情報，ソースコードへのパス情報などを LDRA ツールが読み込めるテストケース・ファイル（.tcf）として生成する．ここで，Simulink のブロック情報に要件 ID を持たせておくことで，生成される各テストベクタに要件 ID が振られ，モデル，要件へのマップが取れるようになる．この仕組みを利用することで，TBreq を介して要件ごとに割り振る，ソースコード，テストケース・ファイルの抽出が容易になる．また，ソースコードに対しては，Testbed に読み込ませることで，カバレッジ測定の為のタグ付けも行われる．

TBrun を起動し，テストを実行するためのドライバー・ハーネスなどを自動生成させる．これらは，TBrun を介してコンパイル・リンクされ，実行される．結果は，ファイルに格納され，テストの Pass/Fail 判定とカバレッジの解析が行われる．静的解析・テストベクタの実行・カバレッジ結果が，自動的に要件にマップされ，トレーサビリティのエビデンスを得る．以上の成果物，ソースコードに対する一連のテストベクタ，ドライバー，テスト結果は TBrun により管理され，リグレッションテスト時の自動化が支援されるようになる．

手順を以下に記述する．

SL2TVEC で Simulink モデルの変換を実行し，T-VEC フォーマル言語にする．

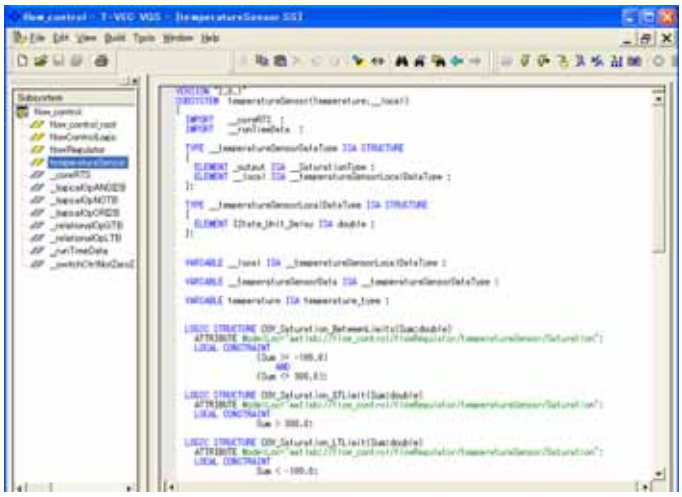

図 15. フォーマル言語

VGS でフォーマル言語をビルドし，テストベクタを生成する．temperatureSensor サブシステムでは，8 件のテストベクタが生成される．これらを VGS の機能を用いてテストドライバー化する．

図 16. temperatureSensor テストベクタ

TBreq で要件からテストまでのトレーサビリティをツリー構造で定義し，要求仕様書（DOORS）をインポートする．

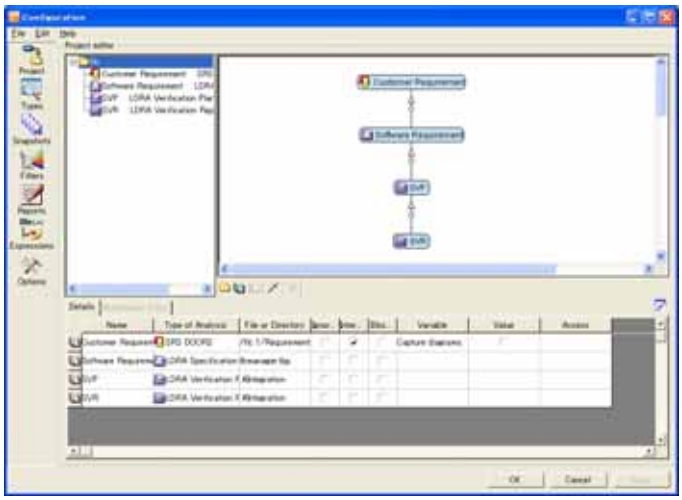

図 17. プロジェクトツリー

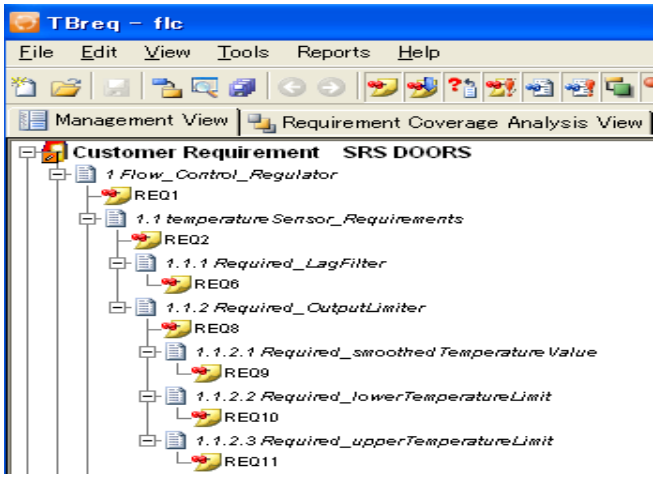

図 18. DOORS 要件のインポート

プロジェクトマネージャーは，DOORS 要件ごとにテスト担当者を割り振る．今回は，要件：REQ8 を一担当者に割り振る．担当者は，自身に割り振られた要件を確認することができる．

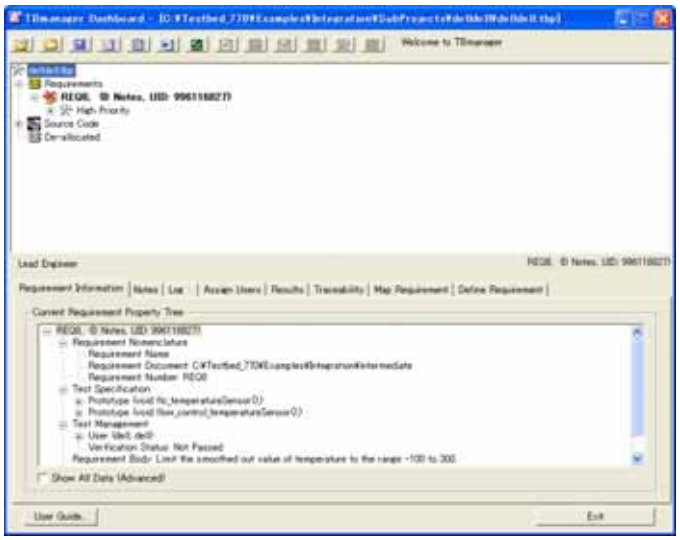

図 19. 要件の割り振り

担当者は，この要件に対してソースコードを割り振り，TBreq から Testbed を呼び出し，TBrun を起動する．生成したテストドライバーファイル（.tcf）を，TBrun にインポートし，テスト対象プログラムと共にコンパイル及びリンクされ，実行される．

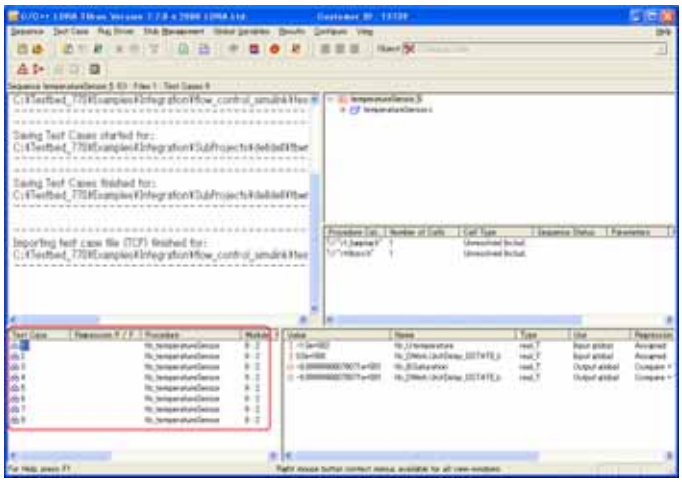

図 20. テストドライバーファイルのインポート

テストのカバレッジは 100%となった．また，期待値と実行値の比較結果は全て OK となり，デザインモデルとソースコードの一致性が確認できた．

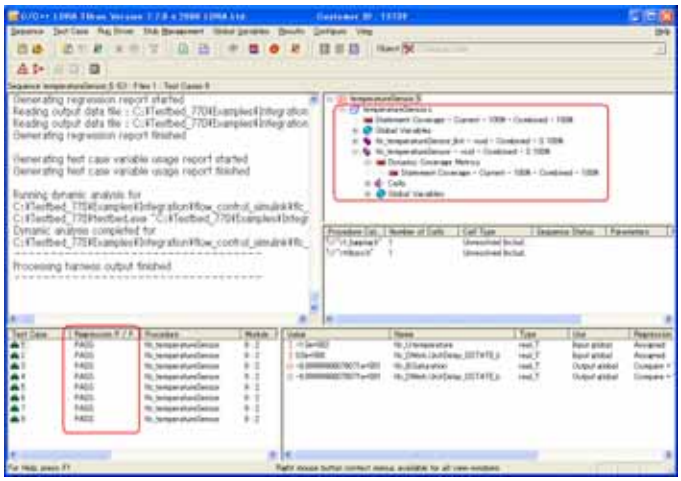

図 21. テスト・カバレッジ結果

GUI を使ってテスト・カバレッジ結果を TBreq に反映し，TBrun と Testbed を終了することで，TBreq では，この要件：REQ8 が検証済みとなる．

図 22. カバレッジメトリクス

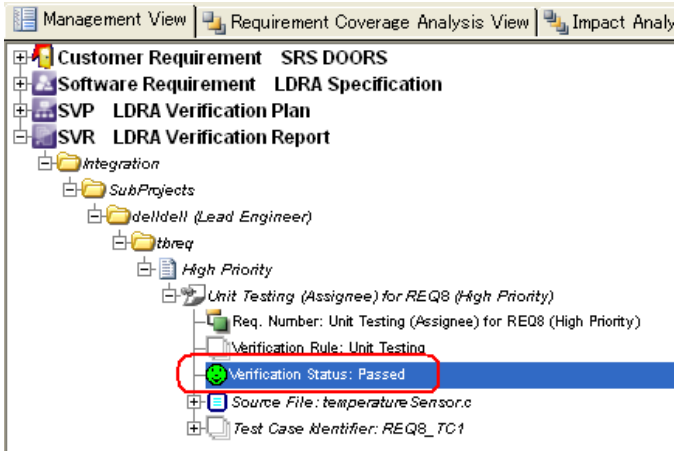

図 23. ベリフィケーション結果

図 23. からプロジェクトツリー内の要件が検証されていることが確認できる．

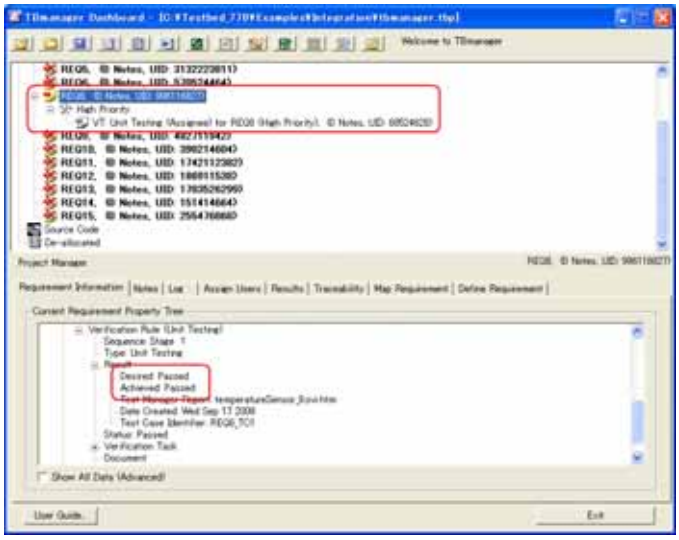

図 24. 要件検証結果

また，以下 Graphical View を参照することで，トレーサビリティがグラフィカルに表示される．このように，要件からテストに至る検証作業とその結果のトレーサビリティがとれる．

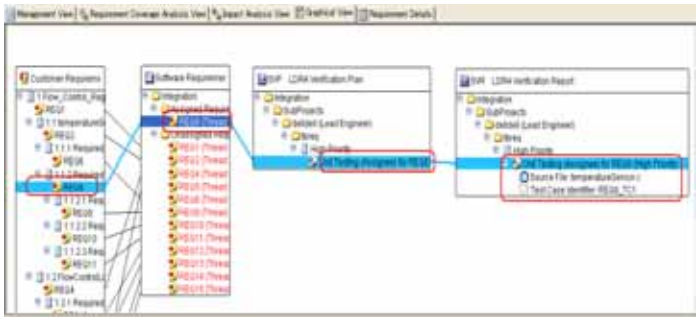

図 25. トレーサビリティ

4. まとめ

プロダクトライン開発で体系的に管理される資産からバリアントを自動生成し，テストプロセス自動化フレームワークを用いて要件がテストされた証明を取得するまでの統合化を確認した．またこの統合に際し，プロダクトラインから生成される Simulink モデルに要件 ID を包含させることで，テストベクタやコードを要件ごとに割り振る作業を自動化し，テストプロセスを効率化させることができた．

さらに実ターゲット環境に於いても，テスト実行環境に合わせたテストハーネスを入れ替えるだけで，バリアントの成果物，プロセス等の全てをそのまま利用できることを確認した．（ターゲット：ルネサス社ＳＨデバイスターゲット，ICE：ローターバッハ社）

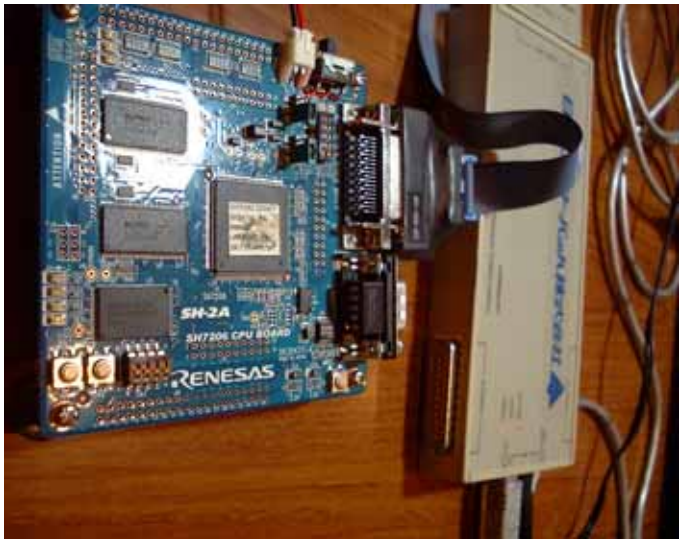

図 26. ターゲット環境

従来型のテストは，一製品開発ライフサイクル内の限られたフェーズにのみフォーカスされたものが多い．しかしながら，プロダクトライン開発では，複数のバリアントが同時に開発され，その中でプラットフォームなどコア資産の変更が頻繁に行われる．調査したテストプロセス自動化フレームワークならば，あらゆる資産の変更の影響を，迅速かつ正確にテストすることが可能となる．

以上，ソフトウェア・プロダクトライン・ライフサイクルに於けるテストプロセス自動化にフォーカスしたが，要件やソースコードなどの変更を管理し，それらの成果物をプロダクトライン開発のコア資産として登録・管理させる仕組みも必要である．特に，自動車やコンシューマ機器のように単一のソフトウェア・プロダクトラインから複数のソフトウェア・プロダクトラインが派生されるような肥大化した製品ファミリー間の，バージョンやバリエーションの識別・管理を含めたソフトウェア・プロダクトライン・ライフサイクルの調査を，今後行いたい．

また今回はテストベクタ生成にデザインモデルを

使用したが， 2.2.T-VEC で紹介した要求モデルを検証モデルとしてバリアントごとに自動生成させて，テストベクタ生成と，要件からテストに至るトレーサビリティの検証も，取り組む予定である．

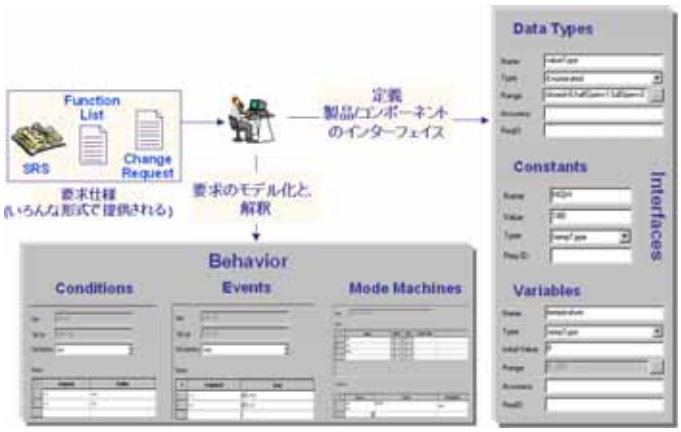

図 27. TTM による要求仕様のモデル化

リファレンス

[1] Software Product Line Engineering with Feature Models
Danilo Beuche, pure-systems
Mark Dalgarno, Software Acumen
http://www.pure-systems.com/fileadmin/downloads/pure-variants/tutorials/SPLWithFeatureModelling.pdf
[2] Technical White Paper
Variant Management with pure::variants
pure-systems GmbH
http://www.pure-systems.com/fileadmin/downloads/pv-whitepaper-en-04.pdf
[3] モデルベース開発におけるテスト自動化フレームワーク
富士設備工業株式会社 電子機器事業部
http://www.fuji-setsu.co.jp/files/aistTVEC.pdf