

Interface-Driven, Model-Based Test Automation

Dr. Mark R. Blackburn, Robert D. Busser, and Aaron M. Nauman
Software Productivity Consortium



Thursday, 1 May 2003

Track 8: 9:15 - 9:55

Room 251 D - F

この資料では、インターフェイスの定義を明確にして、一体化した要件モデリングを行うことで、テストケース、テストドライバーの自動生成をサポートできるアプローチを紹介する。そしてテキストレベルの要件から、コンポーネントやシステムのインターフェイスの観点で、再利用性の高いモデルを、テストエンジニアが開発する方法にフォーカスする。ここで話題の中心にする、インターフェイス駆動 (*interface-driven*) によるモデリングは、要件駆動のプロセスや、モデルベースのテスト自動化を改善することで知られている。そして経験者により、大規模プロジェクトにモデルベーステストを適応させる場合に必要となる、見識とアドバイスが提供されている。

モデルベースのテスト自動化は、コスト削減に寄与し、要求仕様上の欠陥の早期検出を可能にし、テストのカバレッジを改善する [1, 2, 3, 4, 5, 6]。産業界によるモデルベースのテスト自動化の経験から、自動テスト生成をサポートするための実践的な手法として、要件モデリングにインターフェイス駆動で解析を行うことが見識として得られている。ちなみに、この資料では、インターフェイスの用語は、ルーズな定義の下使用する。

Interface とは、コンポーネントの入出力であり、入力をセットして、状態や履歴の情報を含み、そして結果としての出力を得る、仕組み・構造を伴うもの。モデルの再利用性向上や、テストドライバー用の関連マップ情報を得るために、インターフェイスの解析を合わせて、テキストベースの要求仕様をモデリングすることが、推奨される。

Test-driver mappings は、モデル上の変数と、テスト対象システムのインターフェイス間の関連を定義するもの。実産業界の大規模システムの検証に対して、キーとなる資産の活用を支援し、組織の統合をサポートしながら、モデルと関連するテストドライバーマッピング情報を拡張する方法を理解するうえで、役に立つ知見が得られている。

我々はモデルベースのテスト自動化手法として、Test Automation Framework (TAF) を 1996 年から適応してきた。TAF は、システムとソフトウェアの欠陥を排除し、自動テストを支援するために、政府機関や市販の様々なモデルベース開発ツール、テスト生成ツールを統合している。この TAF は、SCR (Software Cost Reduction) のような要件を主眼にしたモデリング手法や、デザイン情報の表現に主眼を置いた、UML (Unified Modeling Language) 制御系システム (航空宇宙や自動車など) のモデリングに採用される Mathwork 社 Simulinkなどをサポートする。

モデル変換により、要件ベース、あるいはデザインベースのモデルはテストスペシフィケーションに変換される。そして T-VEC は、テストスペシフィケーションを用いてテストを生成する、TAF のテスト生成部品である。T-VEC はテストベクタ生成、テストドライバー生成、要件ベースのテストカバレッジ解析、テスト結果解析、レポートをサポートする。ここで生成されるテストベクタ (**Test vectors**) は、入力と期待値、そして要件からテストに至るトレーサビリティ情報を含む。テストドライバー用のマッピング情報とテストベクタは、テストドライバ

ー生成機能への入力となり、テストドライバーが生成される。そしてテストドライバーは実装されたシステムに対して、テスト実行される。

参考文献として、要件モデリングを解説する資料[7]、その他要件モデリングと自動テスト生成の事例[3, 8, 9, 10] などがある。Aissi氏は、テストベクタ生成に関する実績と、いくつかの市販ツールを紹介している[11]。Pretschner氏とLotzbeyer氏は、TAFに類似する、モデルベースのテスト生成を含んだ、エクストリームなモデリングについて簡単に論じている[12]。モデルベースのテストには様々なアプローチがあり、Robinson氏は、ツールや資料、提唱者などへの有用なリンクを持つWebサイトを運営している[13]。

なぜインターフェイス駆動のモデリングなのか？

要件からモデルを先に開発することが適切に思えるが、テストを目的としてモデルを開発する場合は、テスト対象システム、コンポーネントへのインターフェイスの分析と同時に行ったほうが良い。振舞いの要件をモデリングする場合、一度インターフェイスと動作が理解されることで、通常シンプルになり、展開させやすくなる。なぜなら一般にテキストで定義される振舞いの要件は、インターフェイスを介してアクセスされるオブジェクトを象徴する変数の観点からモデル化されるので。

検証モデル (*verification model*) は、コンポーネントのインターフェイスの観点で仕様化された一種の要件である。インターフェイスからの検証モデリング (Verification modeling) は、テストエンジニアがテスト対象コンポーネントの、特定インターフェイスに対してテストを開発することに類似している。テストエンジニアの役割が、要件から検証モデルを開発する上で求められる。要件エンジニアは、場合によってはシステムエンジニアとして、テキストレベルの要求仕様書に加えて、何らかの分析モデルも開発する。デザインエンジニアは、システムアーキテクチャのコンポーネントを特定することに主眼を置き、コンポーネントのインターフェイスをテストエンジニアに提供する。

これらの要件とインターフェイスは、テストエンジニアによって用いられ、フォーマルな検証モデルが構築される。テストエンジニアはTAFツールを用い、モデル解析を介して仕様の矛盾を正し、加えてテストベクタとドライバーを生成する。そしてテキストレベルの要求仕様書、モデル、そしてフォーマルな検証モデルが完成すると、それらはデザインエンジニアと実装担当者に提供される。

そしてコードが生成 (実装) されると、自動生成されたテストドライバーを用いてテストが実行される。TAFの変換ツールは、検証モデルをT-VECの形式に変換し、そして要件からテストに至るトレーサビリティ情報を有するテストベクタとテストドライバーが生成され、実装上の欠陥が要件に対して追跡できるようになる。デザインエンジニア/実装担当者は、実装したシステムをテストするために、このテストドライバーを継続的、かつイテレーティブに活用できる。

Figure 1では、検証モデリング工程の概観を紹介している。モデル担当者は様々な入力情報を提供されている。プロセスが開始時は、仕様は未確定で貧しい情報であることが一般的だが、ここでの入力情報は、システム要件とソフトウェア仕様 (e.g., System Requirements Specification [SRS], Software Requirements Specifications [SWRS] など)、ユーザドキュメント、インターフェイスコントロールの仕様書、アプリケーションプログラムのインターフェイス仕様書、既存のデザインやテストスクリプトなど。

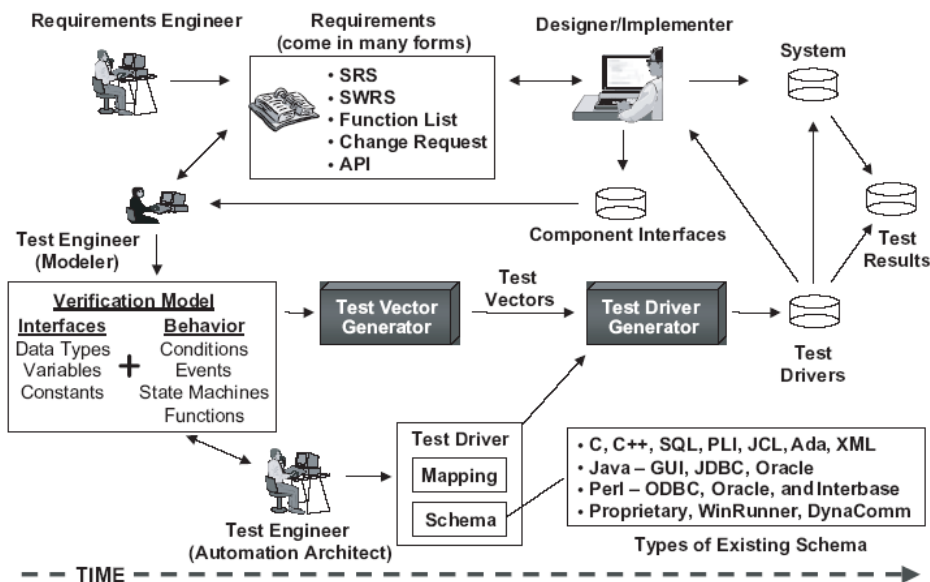


Figure 1: Interface-Driven, Model-Based Test Automation Supports Continuous Verification and Validation

検証モデルは、モデルと一つ以上のテストドライバーマッピングで構成される。テストドライバーは、オブジェクトマッピングとスキーマ（パターン）で構成される。オブジェクトマッピングは、テスト対象システムのインターフェイスと、モデルのオブジェクトとの関連を取る。スキーマはテストケースを実行するための一定のパターンを定義する。モデルは一般にインクリメンタルに開発される。そしてテストベクタ生成機能は、モデル化されたテスト不可能な要件を検出できる。（要件上の矛盾など）

SCR 手法のような表形式のモデリングは、テストエンジニアにとって、効果的で、かつ相対的に容易に学習ができていてる[2]。デザインエンジニアは通常、ステートマシン（状態遷移）あるいは UML の表記などでモデルを開発している。しかしながら、テストエンジニアは表形式に要件をモデル化することが容易であることを、TAF のユーザーやプロジェクトリーダーは認識している[8]。SCR 手法のツールでサポートされるモデリングの表記は、明確な構文（シンタックス）と意味論（セマンチックス）にて、振舞いの要件を、厳密で分析可能に定義することができる。

様々なモデリングについて

仕様記述言語は、一般にグラフィカルなモデリング環境がサポートされることで、モデルを記述・描写する。仕様記述言語で、システムとソフトウェア要求とデザイン情報の、抽象的な描写ができる。Cooke et al.氏は、仕様記述言語の特徴を分類する一覧を作った[14]。Figure 2では、あらゆる仕様記述言語と無関係に、仕様の目的に応じて3つの仕様カテゴリーを図解する。Cooke et al.氏は、ほとんどの仕様記述言語は、通常異なったクラスの仕様を統合したハイブリッド型のアプローチをベースにすると指摘している。

要求仕様書（Requirements specifications）は環境とシステム間の境界を定義し、結果、システム上の制約を既定する。機能仕様書（Functional specifications）は、コンポーネント間のインターフェイスの観点で振舞いを定義し、デザイン仕様書（Design specifications）は、

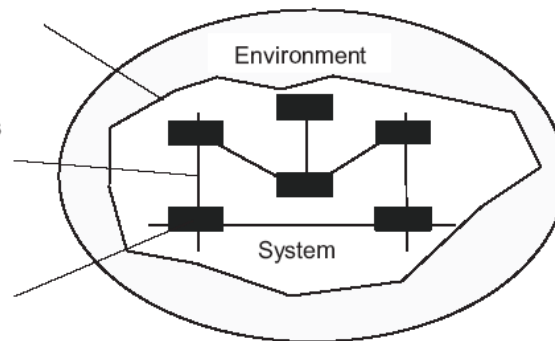
コンポーネントそのものを定義する。ここで、この資料で紹介する 検証モデル (verification model) は、機能仕様書に分類される。

Figure 2: *Specification Purposes*

Requirements Specification: Defines the boundary between the environment and the system.

Functional Specification: Defines the interfaces within the system.

Design Specification: Defines the component.



Note: D. Cooke et al., 1996

“デザイン for テスト” とインターフェイス

振舞いの要求をモデリングする前に、テスト対象システムのインターフェイスを理解することが最善であり、そのことによりテスト対象システムの実入出力へテストドライバーのマップを得るためのインターフェイスを確実に得ることができる。もしインターフェイスがフォーマル化（形式化）されず、あるいは完全に理解されなくても、要件モデルは開発できるが、テストドライバー生成に必要なオブジェクトマップ情報は、インターフェイスが明確にされた時点で、用意されなければならない。

その場合、オブジェクトマッピング構築作業はより複雑となる。なぜならモデルの実体はコンポーネントのインターフェイスにマップされないのだ。加えて、もしコンポーネントのインターフェイスが他のコンポーネントと結合 (coupled) していると、一般にそれらコンポーネントは、個々のインターフェイスから完全に制御できない。そして、モデリングやテストの工程も困難になる。以下 Figure 3 で、概念としてのコンポーネントとインターフェイスの描写を考察してみる。

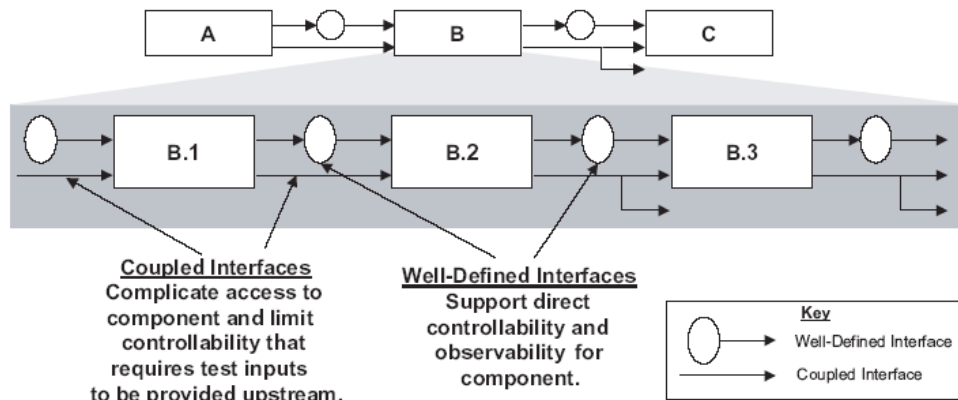


Figure 3: *Conceptual Components and Interfaces of a System*

体系的な検証のアプローチをサポートする明確な方法がある。それは段階的に実行され、各コンポーネントが、それに割り当てられる要件について完全に検証される。コンポーネントへのインターフェイスは、明示的で、かつ完全にアクセス可能であるべきで、グローバルメモリを用いるか、より良いのは、ゲット&セットのメソッドやプロシジャを介すること (Figure 3 参照)。

例えば、もし上位階層のコンポーネント B からの B.2 コンポーネント入力への入力設定が完全に可能であり、また B.2 機能からの出力も完全に観測できるのであれば、B.2 内の機能性は完全に仕様化して体系的に検証できる。しかしながら、もし B.1 などのような他のコンポーネントからのインターフェイスがアクセス不可能であれば、いくつかの B.2 コンポーネントの機能性は B.1 に結合され、また B.2 へのインターフェイスは B.1 へのインターフェイスを含まなければならない。あるいはコンポーネント A のような、さらに上流のコンポーネントまでも。このようなインターフェイスの結合 (interface coupling) は、テストドライバーのインターフェイス設定をより複雑なものにするだけでなく、振舞いのモデリングが、コンポーネント同士の結合に割り振られてしまう機能性の観点で記述されることになる。

結合 (coupling) は、コンポーネントの再利用を損ねることになり、またシステムコンポーネント間の結合により、リグレッションテストの苦勞と工数を増加させることになる。重度に結合されたシステムのテストに関わる問題は、モデルベーステストを難しくし、またどのようなタイプのテストにも悪影響を及ぼすことになる。我々は、インターフェイス駆動のモデリングを採用することで、結合は削減され、より良いシステムデザインを育むことと、より良いテストの支援が得られることを確認している。

インターフェイスの理解

TAF による著名な成果の一つに、Mars Polar Lander (MPL 火星無人探査機) の事故解析事例がある。NASA は、1994 年 2 月 7 日に MPL プロジェクトを着手し、6 年後の 1999 年 12 月 3 日、MPL は 3500 万マイル以上の飛行をし、予定された着陸まで数分のところで、地上との交信が途絶えてしまった。この開発と打ち上げには、1 億 6500 万ドルが費やされていた。

我々は事故後、TAF でバグを検出することができるかを、調査する機会を得た。そこで、あえてコードを見ることなく、SCR 手法を基にしたツールで、要求仕様書をモデル化し、テストを自動生成させた。具体的には Touchdown Monitor (TDM 着地モニター) の要求仕様を、TAF ツ

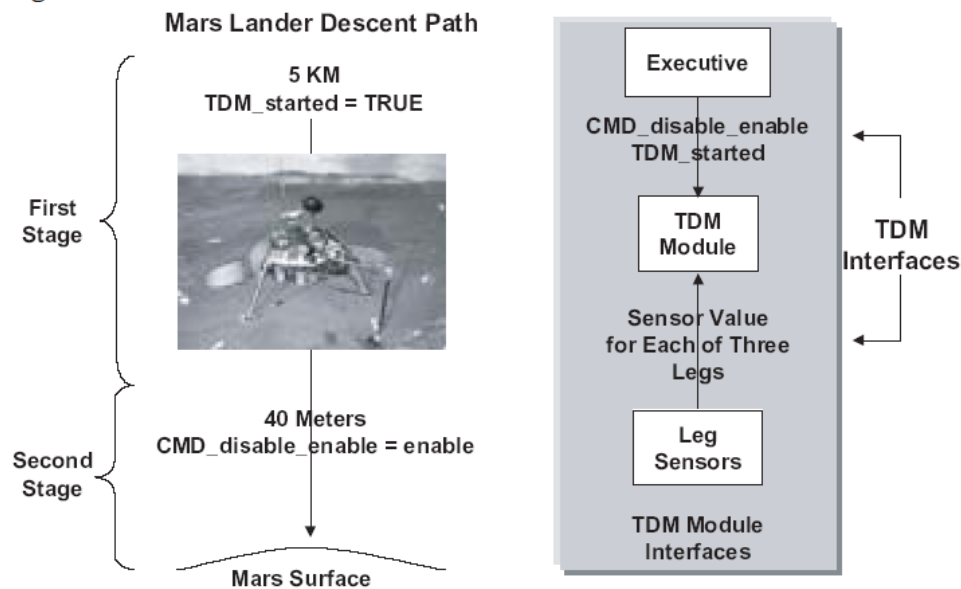
ールでモデル化し、MPLの着陸手順に関連するソフトウェア上のエラーを、24時間以内に発見することができた。

TDMは、火星へMPLが降下する時の2つ段階で、3本の着陸脚の状態をモニターするソフトウェアコンポーネントである。Figure 4にあるように、TDMモジュールを、1秒当たり100回、リアルタイムなマルチタスク実行がコールし、着陸脚センサーからの情報を2つ目のモジュールから受け取る。これら2つのモジュールはTDMへのインターフェイスを構成している。火星上空およそ5 Kmから開始される降下第一段階で、TDMソフトウェアは、3本の着陸脚をモニターする。

各着陸脚には1つのセンサーがあり、これによって脚が着地したことを確認する。各着陸脚を所定の位置に固定するときに、反動でセンサーが誤った着地信号を示す課題は想定されていた。そこでTDMソフトウェアは、この想定されたイベントに対し、2回連続で誤った信号を読み取ると、その着陸脚のセンサーを不良として、マーキングする。そして、火星上空およそ40mから開始される降下第二段階で、TDMソフトウェアは、残りの正常なセンサーをモニターする。その後、1つのセンサーから着地を知らせる2回連続の信号を読み取ると、TDMソフトウェアは、降下エンジンを停止する。

MPLに何が起こったか、絶対的な確認手段は無い。考えられているのは、1つ以上のセンサーから、上空40mに達する前(第一段階)に、2つの連続した信号を読み取り、その着陸脚センサーの情報がTDMプログラムメモリに保持されていたこと。そしてMPLが上空40mに達したとき、TDMの状態が変更され、降下第一段階の着陸脚センサーに相当するメモリを読んだ。そしてメモリが2回連続の信号読み取りを示し、エンジンスラストが火星上空40mで停止されたと、開発エンジニアは確信している。様々な方法で要件のデザインと実装を行えるが、この設計上の本質的な欠陥は、プログラム変数が、不良センサーの情報を保持していたこと。(そしてプログラム上にエラー(バグ)を誘発した)

Figure 4: Mars Polar Lander Details



企業に於ける優良事例・教訓

インターフェイス駆動のモデリングは、開発が完了してから行うこともできるが、開発中に取り入れることで飛躍的な効果を得られることが確認されている。理想的には、テストエンジニアが開発エンジニアと並行して作業に参加し、インターフェイスを固定すること、要件を洗練すること、そしてイテレーティブなテストと開発をサポートする、モデルを作ること。テストエンジニアが数百、数千行ものテストスクリプトを書く代わりに、製品の要求仕様（多くの場合不十分なドキュメントから）を、モデルの形式にすること。そうすることで、テストベクタとテストドライバーが自動生成される。

イテレーティブな開発において、コンポーネントの振舞いや、インターフェイス、あるいは要件の変更に応じて、モデルは修正されテストベクタやテストドライバーを再生成し、再実行できる。主要な効果は、テストを開発と並行して進行できること。Lockheed Martin 社などユーザーは、50%以上のテスト工数が削減できること。早期段階での要件の解析により、要件上の欠陥（矛盾、一貫性の無さ、機能間の相互干渉など）を排除し、飛躍的に手戻りを削減できたこと。を公表している

その他の事例

TAF は、航空宇宙、医療機器、飛行ナビゲーション・ガイダンス・表示システム、自動飛行装置、飛行管理、コントロールロー、エンジンコントローラー、航空管制・衝突回避システム、など様々なドメインのクリティカルなアプリケーションに採用されている。また、クリティカルではないアプリケーションでも、データベース、クライアント-サーバーシステム、Web ベースアプリケーション、自動車、通信システムなどにも採用されている。また、それらによるテストドライバー生成は、C, C++, Java, Ada, Perl, PL/I, SQL, など様々な言語や、独自の言語、またテスト実行システム(e.g., DynaComm, WinRunner) などあらゆる実行環境で活用されている。そして殆どのユーザーが、50%以上の検証・テスト工数を削減している[2, 15]。

まとめ

この資料では、モデルベースのテスト自動化を支援するための、インターフェイスの解析を要件モデリングに一体化させることに関する、実践的な指針を提供した。ここで紹介したモデルベースのテスト手法とツールは、飛躍的にテスト実行に関わる費用と工数を削減できること、そしてまた、要件上の欠陥を検出することで費用のかかる手戻りが削減されることも、実証されている。

インターフェイスを定義することによりテストの容易性に貢献できるというこれらの提言は、全てのテスト形態に対して有効である。複数の組織により、インターフェイス駆動なモデルベースのテストの導入により、システムのインターフェイスを早期に安定化させることと、共通に利用できるテストドライバー生成機能を一度設定すれば再利用が容易であることなどの効果を確認されている。最後に、検証モデルを製品と並行して開発することは有益で、要件の欠陥を早期に発見できることから手戻りを削減できる。

References

1. Rosario, S., and H. Robinson.
"Applying Models in Your Testing Process, Information and Software Technology." 42.12 (1 Sept. 2000).
2. Kelly, V., E. L. Safford, M. Siok, and M. Blackburn.
"Requirements Testability and Test Automation," Lockheed Martin Joint Symposium, June 2001.
3. Blackburn, M. R., R. D. Busser, A. M. Nauman, R. Knickerbocker, and R. Kasuda.
"Mars Polar Lander Fault Identification Using Model-Based Testing." Proc. in IEEE/NASA 26th Software Engineering Workshop, Nov. 2001.
4. Busser, R. D., M. R. Blackburn, and A. M. Nauman.
"Automated Model Analysis and Test Generation for Flight Guidance Mode Logic." Digital Avionics System Conference. Indianapolis, IN, 2001.
5. Statezni, David.
"Test Automation Framework, State-Based, and Signal Flow Examples." Twelfth Annual Software Technology Conference. Salt Lake City, UT, 30 Apr.-5 May 2000.
6. Statezni, David.
"T-VEC's Test Vector Generation System." Software Testing and Quality Engineering. May/June 2001.
7. Heitmeyer, C., R. Jeffords, and B. Labaw.
"Automated Consistency Checking of Requirements Specifications." ACM TOSEM 5.3 (1996): 231- 261.
8. Blackburn, M. R., R. D. Busser, and A. M. Nauman.
"Removing Requirement Defects and Automating Test." STAREAST. Orlando, FL, May 2001.
9. Blackburn, M. R., R. D. Busser, and A. M. Nauman.
"Eliminating Requirement Defects and Automating Test." Test Computer Software Conference, June 2001.
10. Blackburn, M. R., R. D. Busser, A. M. Nauman, and R. Chandramouli.
Model-Based Approach to Security Test Automation. Proc. of Quality Week, June 2001.
11. Aissi, S.
"Test Vector Generation: Current Status and Future Trends." Software Quality Professional 4.2 (Mar. 2002).
12. Pretschner, A., and H. Lotzbeyer.
Model-Based Testing with Constraint Logic Programming: First Results and Challenges. Proc. of 2nd ICSE Intl. Workshop on Automated Program Analysis, Testing, and Verification. Toronto, Canada, May 2001.
13. Robinson, H. Model-Based Testing
<www.model-based-testing.org>.
14. Cooke, D., A. Gates, E. Demirors, O. Demirors, M. Tankik, and B. Kramer.
"Languages for the Specification of Software." Journal of Systems Software 32 (1996): 269-308.21.
15. Safford, Ed L.
"Test Automation Framework, State-Based and Signal Flow Examples." Twelfth Annual Software Technology Conference. Salt Lake City, UT, 30 Apr.-5 May 2000.

About the Authors



Mark R. Blackburn, Ph.D., is a Software Productivity Consortium fellow with 20 years of software systems engineering experience in development, management and applied research of process, methods, and tools. He has made more than 30 presentations at conferences and symposia, and is involved in consulting, strategic planning, proposal and business development, as well as developing and applying methods for model-based approaches for requirement defect removal and test automation.



Robert D. Busser is a principal member of the technical staff of the Software Productivity Consortium. He has more than 20 years of software systems engineering experience in development, and management in the area of advanced software engineering, and expertise in software engineering processes, methods and tools. He has extensive experience in requirement and design methods, real-time systems, model-based development and test generation tools, model analysis, and verification.



Aaron M. Nauman is a senior member of the technical staff of the Software Productivity Consortium. He has a wide range of systems and applications development experience in both real-time and information systems domains. Nauman is involved in the development of model-based transformation tools for automated model analysis and test generation. He has experience in object-oriented technologies, distributed and client/server systems, Web-based and components-based software and systems integration.

この資料は、Software Productivity Consortium（現 Systems and Software Consortium）の研究成果として、2003年に発表されたものです。現在 TAFのコアとなる T-VEC（形式手法を用いた、モデル検査・テストベクタ/ドライバー生成機能）や、SRC手法を採用した TTM モデリングツールは更なる進化を遂げて、産業界で広く採用されています。

興味深い事例として、某医療機器メーカーでは、既存のテストシナリオから、TTM を用いて検証モデルを作成し、テストベクタとドライバーを自動生成しています。そしてこの検証モデルは再利用され、派生開発で重荷になっていたテストの工数を飛躍的に削減することに成功しています。

これらツールの詳細は、T-VEC Technologies 社（www.t-vec.com）あるいは富士設備までお問合せください。



富士設備工業株式会社 電子機器事業部
〒591-8025 大阪府堺市北区長曾根町1928-1
Tel: 072-252-2128 www.fuji-setsu.co.jp