WHITE PAPER | 2016 年 8 月

モデルベースのテストに関する上位 5 つの誤解

モデルベースのテストとは、難度が高く、プロジェクトを遅らせ、本格的なアジリティを妨げるものでしょうか？あるいは、テストをより体系的で変化に対応できるものにし、ビジネスと IT の緊密なコラボレーションを促進するものでしょうか？

Huw Price
アプリケーション・デリバリ

ca technologies

目次

セクション 1

はじめに

モデルベースのテストは目新しいものではありませんが、テストへのモデル化の応用はいまだに限られています。アジャイルの登場以降は特にその傾向があり、純粋なアジャイルまたはハイブリッド・アプローチのいずれかを採用しているチームではテストのモデル化はあまり見られません。

CA Agile Requirements Designer チームでは長年、モデルベースのテストを推奨してきました。私たちはこれまで目にしてきたこのアプローチへの疑問や異議、あからさまな拒否をもとに、最もよくある誤解のうち 5 つを特定しました。

1. モデル化するにはシステムが大きすぎる / 複雑すぎる
2. すべての反復またはプロジェクトの開始時にモデルを作成する時間はない
3. 完全なモデルを維持するだけでは変化に対応できない
4. フォーマルなモデル化はエキスパート以外には難しすぎる
5. モデル化はアジャイルではない / アジャイルの枠組みに合わない

これらの懸念に対応するために、フローチャート・モデルを使用してモデルベースのテストが提供するメリットのいくつかを示します。

セクション 2

すべてのテストはいずれにせよモデルを使用

異論は些末なことが多く、一部はモデル化が以前期待を果たさなかったときに受けた悪評から来ています。Paul Gerrard 氏が述べているように [1]、すべてのテストでシステムのモデル化は行われており、違いはモデル化が黙示的か明示的かということです。

たとえば、現在モデルベースのテストよりもっと広く受け入れられているビヘイビア駆動開発（BDD）では、少なくとも黙示的にモデル化が使用されています。BDD は中心的な利害関係者からナレッジを引き出し、たとえば Gherkin 仕様言語を使用して、それをビヘイビア駆動開発の要件に保管します。

要件はこのとき必ずしもフォーマルなモデルとして保管されるわけではありませんが、テスト時にはテスト担当者が黙示的にモデルを構築します。ネガティブ・テストを実施する際は、たとえば、設定節があるのに Gherkin 仕様のトリガーが発生しない場合の対応を考えます。すべてのテストケースの設計がこのようにモデルを黙示的に使用している場合、モデルを明示化するメリットはあるのかという疑問が起こります。

セクション 3

誤解 1 : モデル化するにはシステムが大きすぎる / 複雑すぎる

システムを原因と結果の論理に分割できる場合、モデル化は可能です。たとえばフローチャートによってシステムを「もし～なら・・・する」という一連のステートメントに分割し、プロセスの階層に関連付けることができます。システムは実際にこのような論理手順によって構成されています。[2] したがって、モデルベースのテストを採用すべきかどうかは、大規模あるいは複雑なシステムを前にしたとき、他の設計方法に比べてモデル化の方が実質的なメリットがあるかどうかという問題になります。そのようなメリットのいくつかについて、以下に概要を示します。

コンポーネント化と抽象化

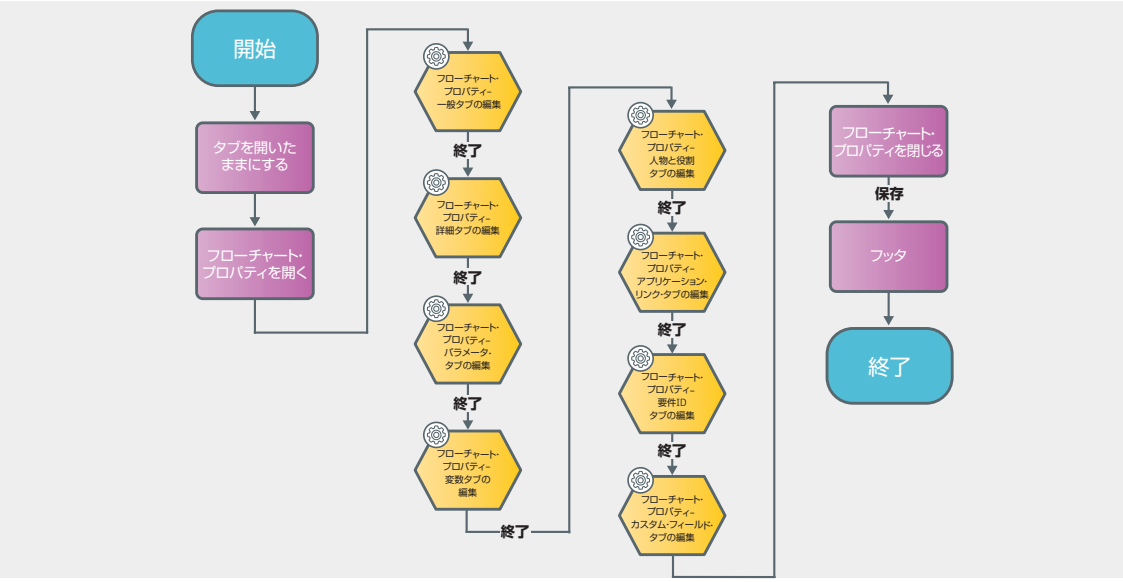
システムが非常に大規模または複雑な場合、モデル化によって論理が管理しやすくなる場合があります。システム全体をモデル化する必要はありません。システムを論理コンポーネントに縮小すれば、抽象化とコンポーネント化が可能になります。これに対して分野のエキスパートは、自分が専門技術を持っている部分のみ設計できます。

個別のコンポーネントをより高いレベルのプロセスの階層のもとに組み込み、たとえばマスター・フローチャートの下にサブプロセスをまとめて関連付けることが可能です。個々の利害関係者はそれぞれの役割を果たすために必要な粒度までドリルダウンできます。テスト担当者やビジネス・アナリスト（BA）はより高いレベルのフローを使用し、システムについて把握したり、コンポーネントの統合状況を幅広く理解したりできます。また、個々のテスト担当者はそれぞれのコンポーネントの詳細を確認し、テストを実施するために必要な情報を得ることができます。

分野のエキスパートは自分が十分理解している機能を指定できますが、このアプローチのもう 1 つのメリットは再利用性です。コンポーネントはモデル化されれば、モデル内で共有し他のテスト担当者が使用することが可能です。図 1 に示すように、これはマスター・フローにサブフローをまとめて結び付けるだけで簡単に行えます。これによって、システム内にある反復されたコンポーネントや機能を簡単に組み合わせ、新しいテストを作成できます。たとえば、ユーザ・インタフェース上で使用されるボタンやフィールドは中央のリポジトリからモデル化し共有できます。同様に API テストでは個別の API をモデル化しテストしてから、サブフローをまとめてチェーン・テストのような、さらに複雑なタイプのテストを構築できます。

図 A

黄色で示した再利用可能なコンポーネントが結合され、マスター・フローを構成しています。黄色いブロックはそれぞれ、より低レベルのプロセスがフローチャートとしてモデル化されています。



完全性

フォーマルなモデル化の実際のメリットが明らかになるのは、完全なモデルから始める場合以外の代替手段を検討するときです。アジャイル開発と継続的デリバリによって、要件の収集方法は変化しました。現在テスト担当者の多くは、ウォーターフォール・プロジェクトの開始時に有効であった文書による仕様よりむしろ、絶え間ない変更要求とユーザ・ストーリーに直面しています。これは変化するユーザ・ニーズの速さに対応しようとする取り組みを反映していますが、同時に新たな課題をもたらします。テスト担当者は多数の関連のないストーリーをつなぎ合わせ、コンポーネント間の依存性を含め一貫したシステムを構築する必要があります。

関連性のない直線的なストーリーを使用して、その場しのぎのやり方でシステムを設計すると、不完全なものができあがります。比較的シンプルなシステムでも、その論理には数千というパスが含まれるため、異なるストーリー間の点を結ぶことは、組織的な方法で行わなければほとんど不可能です。このような要件から導出されるテストは通常、目的とする機能のごく一部しか網羅せず、特にネガティブ・テストはたいてい軽視されます。実際、当社が行ったテストケースの監査では、わずか 10 ～ 20% の機能網羅率が標準的です。

フォーマルなモデル化を使用して不完全性を回避

既存の要件とユーザ・ストーリーを体系的にモデル化するプロセスは、不完全性への対策に役立ちます。線形的なストーリーを一貫したシステムに結びつけることで、設計者は機能的論理の重複について考え、起こり得る入力と出力や、トリガーがない場合の対応策を考慮せざるを得なくなります。

要件で論理パスが設定されていない場合、これは非常に明確です。たとえば、フローチャートの 1 つのブロックに 1 つの出力しかない場合があります。フローチャート・モデルは計算上正確であるため、自動推論や完全性検証を適用し、モデル化プロセス中に不足している出力があれば特定できます。

大量の文書や関連性のない一連の図に比べれば、視覚モデルでは不完全性が残っていればそれを特定することがずっと容易です。当社の顧客企業の例では、分野のエキスパート（SME）がテスト担当者の席の近くを通りかかると、テスト担当者が ETL ルーチンをフローチャートにマッピングしていたそうです。SME は一目で機能論理の不足部分を指摘したということです。CA Agile Requirements Designer を使用すれば、モデルから直接ユーザ・ストーリーを導出でき、ユーザまたは SME が視覚的に検証できます。この簡単な技法は、ユーザ・ストーリーの選択で各論理手順を検証する際、必要な機能が除外されていないかどうかを特定するための確実な方法です。

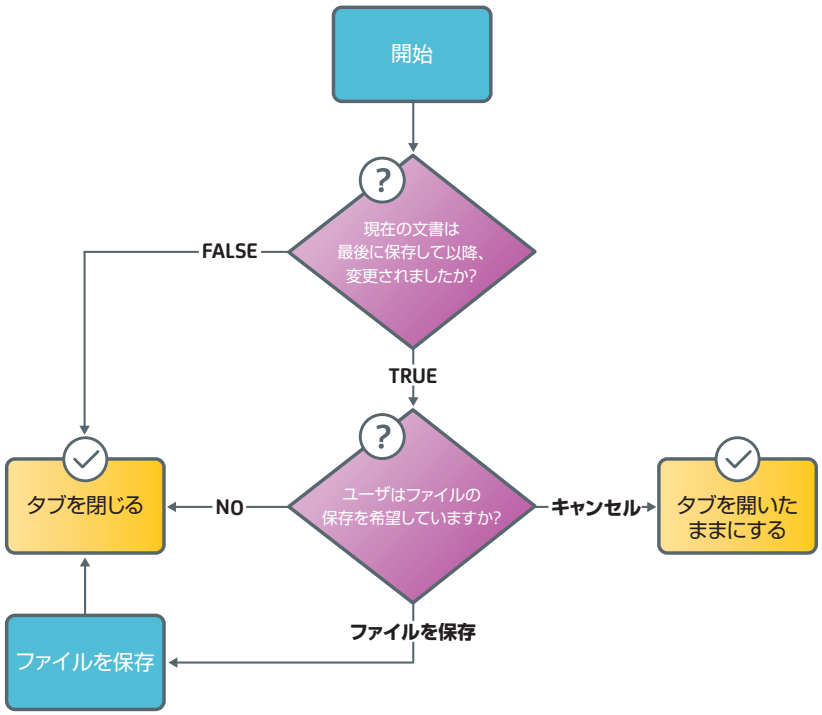
フォーマルなモデル化の使用による曖昧性の排除

不完全性とともに、多くの設計方法では従来の記述された仕様から BPM ダイアグラムやユーザ・ストーリーで使用される言語まで、自然言語が使用されています。こうした自然言語は曖昧性や複数の解釈につながり、テストに反映されるべきシステムの正確性や論理手順からはかけ離れています。このような文書による要件の誤った解釈が欠陥を引き起こすことはよくあります。

一方、フォーマルなフローチャート・モデルは単語の羅列をシステムの中心となる論理手順に分割し、システムの機能を密接に反映させます。したがって、曖昧性が原因の欠陥を避けるために役立ちます。2009 年のある調査では、曖昧性が原因の欠陥は全欠陥の 56% を占めています。[3] 同様に、コペンハーゲンの IT University が 2001 年に行ったケース・スタディでは、製造業者 B&K 社の欠陥の 59% は要件の欠陥であると報告されています。Hyderabad Business School が 2012 年から行った調査では、欠陥の 50 ～ 65% は設計作業の段階で発生していると推定されています。[5]

図 B

フローチャートはシステムを原因と結果の論理に分割します。この高レベルなフローでは、CA Agile Requirements Designer のタブを閉じる操作の背後の機能がモデル化されており、いつシステムがユーザに対して先に作業を保存するよう促すべきかが示されています。



これらの調査が 2001 年のアジャイル宣言の時期にさかのぼっていることは注目に値します。ソフトウェアの設計とテストの方法に対するアプローチと考え方が発展しているにもかかわらず、要件の欠陥の発生頻度とコストに関する数値は一貫しています。調査ではソフトウェア設計に改善の余地があることが指摘され、欠陥の発生と、修復にかかる時間とコストを大幅に減らせる可能性が示されています。

B&K のケースはこれに当てはまり、同社では要件の欠陥防止策によって使用性に関する問題が 70% 減り、実装に関する欠陥は 20% 減少しました。また、そのプロジェクトは初めて予定どおりにデリバーできたプロジェクトになりました。つまり、要件を前もってフォーマルにモデル化することには明らかなメリットがあるのです。Hyderabad Business School の調査によれば、それはユーザ・エクスペリエンスの改善と予定どおりのデリバリに表れ、プロジェクト・コストも 64% 削減できる可能性があります。

セクション 4

誤解 2：すべての反復またはプロジェクトの開始時にモデルを作成する時間はない

要件のフォーマルなモデル化にどれだけの時間がかかっても、ほとんど常に、手作業のテストケース設計にかかる時間を節約した分の方が大きいでしょう。モデルベースのテストは大量のテスト作業を設計段階へと前倒しするため、テストケース、データ、自動テスト、期待される結果の作成を自動化できます。

手作業のテスト設計では時間が無駄になり品質が低下

静的な文章による要件や一律の図は、自動テストケース設計には適応できません。テスト担当者は手作業でテストを導き出し、それをスプレッドシートやテスト管理ツールに保管しています。これは多大な労力を要し時間がかかるプロセスで、当社の顧客の例では 6 時間をかけて作成できるテストは 11 個だけということがありました。テストの 70% を手作業で行っている限り [7]、変化する要件にはほとんど対応できず、テストはほぼ常に次のスプリントへと持ち越されることになるでしょう。

静的な要件から手作業でテストケースを導き出すことは、システマティックではなくエラーも起きやすく、テスト担当者がどのようなテストを作ろうとしてもすべて山積みになる可能性があります。前述のとおり、比較的シンプルなシステムでも、非常に優秀なテスト・チームの予想さえ超える数の起こり得るパスが存在します。手作業で導き出したテストケースの監査では、通常は平均 10 ～ 20% の機能網羅率しかありません。2015 年 7 月に CA が調査した 112 人のうち 42% がソフトウェアの主な課題として、テスト網羅率が不十分なために欠陥と再作業が発生していると述べています。

テストの実施段階を自動化することはテストの作成時間を排除せず、自動化の枠組みの多くはスクリプト作成に大きく依存しています。テストの記述時にテスト・チームが予期しなかったような、重要な機能をテストする方法はありません。

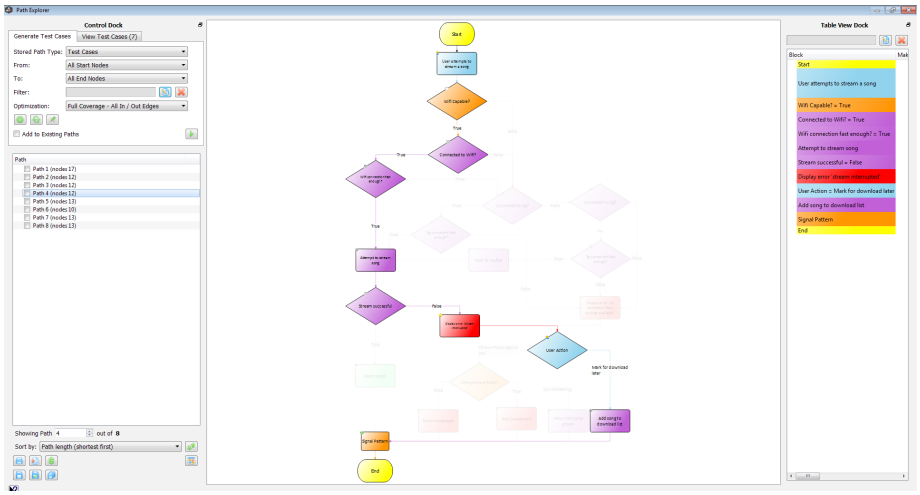
最適化されたテストの自動導出

モデルベースのテストはテストの実施を超えた大量のテスト作業を自動化する方法で、また、最適なテスト網羅率に必要な体系化をもたらします。フローチャートは計算上正確なモデルで、システムの機能論理に支えられており、自動同位分析を使用して起こり得るパスをすべて特定できます。これらのパスはテストケースと同等で、設計で設定されたすべての機能を反映し、最大の機能テスト網羅率を確保します。

特に優れているのがテスト最適化に関することです。テストケースを手作業で直線的に導出すると、多数のテストケースで同じ論理が繰り返されます。当社の監査担当者によると、4 倍のテスト過剰が平均的な状況だということです。モデルベースのテストでは、テスト手順を自動的に異なる順序で組み合わせ、最小セットのパスで最大網羅率を確保することが可能です。複数の最適化方法が存在しますが、1 つの例では CA Agile Requirements Designer を使用することでテストの数が 326 から 17 へと削減され、同時に網羅率は 100% に改善しました。

図 C

このモデルはモバイル・ソング・ストリーミング・サービスの簡略化バージョンです。これには 71 の考えられるパスがあり、これは機能テストと同じで、グラフ分析を使用して自動で特定できます。全入出力エッジ網羅技法を使用して、パスの数は 8 へと削減されました。



このように自動でテストを導出すれば、手作業のテスト設計に必要な時間のほんの一部しかからず、実行の時間もさらに短くなります。CA Agile Requirements Designer を使用したある例では、100% の機能網羅率を達成する 108 のテストケースを作成するのにかかった時間は 90 分でした。これにはフローチャートを作成する時間も含まれています。

CA Agile Requirements Designer で作成された、最適化されたテストは、自動エンジンにプッシュアウトすることも可能です。これは、Critical Logic の TMX のようなフレームワークを使用してフローのパスをスクリプトに変換するか、個別のノードにコード・スニペットを覆うかして行います。こうした再利用可能なスニペットはその後 Path Explorer によって、適切なデータを備えた、最大網羅率に必要なテストの最小セットへ自動コンパイルされます。

モデル化には本当にそれほど長い時間がかかるでしょうか？

別の点を挙げると、既存の要件のモデル化にはそれほど長い時間はかかりません。オランダの保険会社保険会社 a.s.r. の例では、プロジェクト・マネージャがテスト・チームに 2 週間で新しいシステムを一からテストするよう求めました。不可能なことを試み手作業でテストケースを作成するのではなく、テスト・チームのリーダーは 4 時間でフローチャートとして要件をモデル化しました。1 週間以内にチームは、最大網羅率に必要なすべてのテストケースを導出し、既存の HPE ALM フレームワークを使用して自動でそれらを同期化しました。2 日間で 137 個のテストを実行し、厳しい期限内に厳密なテストを完了させました。

モデル化ツールが企業の幅広いツールと統合できる場合、要件とテスト・アセットを活用すれば設計プロセスはさらに短縮できます。CA Agile Requirements Designer は HPE ALM、CA Agile Central、VersionOne、Microsoft Team Foundation Server からテストケースまたはユーザ・ストーリーをインポートでき、また、JIRA からチケットを、BPMN および XPDL と互換性のあるツールに保管された要件をインポートできます。これらの異なる形式に含まれる論理をインポートして統合でき、不完全性と曖昧性を排除することで、自動テストの設計と実行が可能になります。

セクション 5

誤解 3 : 完全なモデルを維持するだけでは変化に対応できない

この誤解は、完全な仕様を維持すると変化するユーザ・ニーズへの対応ができないという、完全な仕様に関する間違った考えと関係しています。当社の経験では、通常はその逆です。

従来のテスト技法では変化への対応は不可能

静的で断片化した要件と要求に直面すると、変化がシステム・コンポーネントにもたらすインパクトを自動的に特定する方法はありません。テスト担当者は多くの場合、手作業でこれを解決しようとします。これは技術的負債がある場合は特に困難です。複合システムがさらに複雑化し、分野のエキスパートがきちんとした文書を残さずに退職してしまうと、技術的負債は増大します。システム内のすべての稼働パートがどのように関連しているかがわからなければ、一見単純でささいな変化でもシステム障害を引き起こす可能性があり、比較的優先度の低い改善でも何か月もの作業を必要とすることがあります。

テスト・アセットを保守して変更を反映させようという試みもあります。テスト担当者は通常、手作業で既存のテストケースを確認して更新しますが、これは非常に面倒な作業で、変更がシステム全体に正常に反映されたことを確認するために必要な網羅率はほとんど確保できません。網羅率を維持しようとするためにテストが山積みになる場合がありますが、それではテスト過剰の状態が広がり、無効なテストが原因でテストが失敗して時間がかかります。また、チームが既存のテストをすべて廃止して最初からやり直すこともありますが、システムをテストするために必要なテストの数が増えると、すぐにスプリント・サイクルに対応できなくなります。

自動の依存性分析とテストの保守

変化する要件に対応するには、以前のテスト・アセットがリアクティブで再利用可能なものでなければなりません。変更が行われたときにはこれらを活用することが必要で、別の利害関係者が同じ作業を繰り返すべきではありません。フローチャート・モデルはこれを可能にします。フローチャート・モデルは完全ですが動的で、依存性分析と自動テスト保守を可能にします。

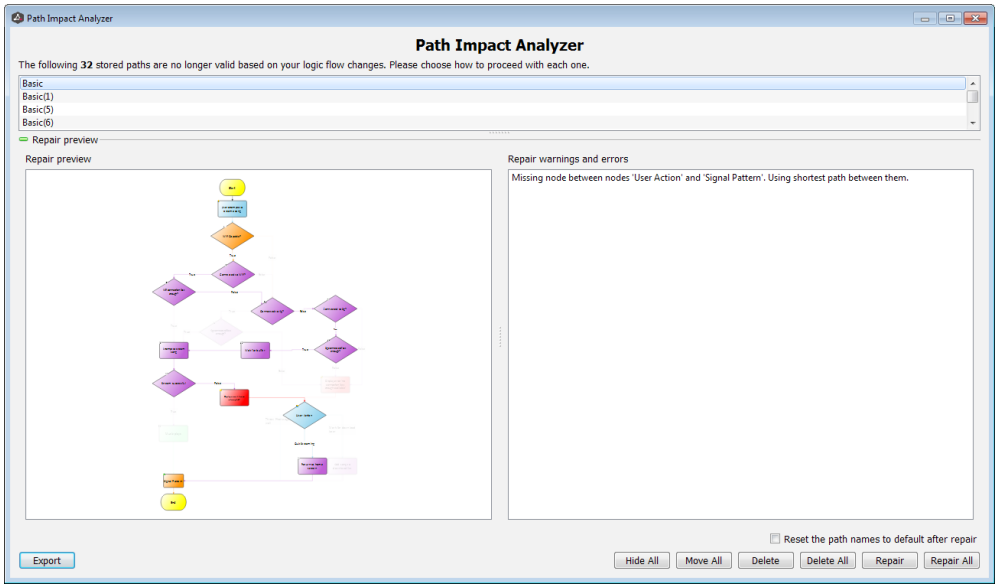
動的なフローチャートに新しい論理が追加されると、コンポーネント内やコンポーネント間のパスへのインパクトを自動で特定し計算されます。自動依存性分析を使用して、マスター・フローの何が影響を受けるか、低いレベルのフローでは変更がどのように影響するかを特定できます。

CA Agile Requirements Designer を使用すると、実際に変更を行う前にこの分析を実行して予期しない結果を回避できます。「インパクト表示」機能はまた、変更がシステムのテストと開発の複雑性にどのように影響するかを特定するために役立ちます。これによって予期しない再作業が発生する前に、変更の比較的な価値を分析できます。これは静的な設計技法では不可能なことです。

影響を受けたパスはその後、パス作成時に使用したのと同じ分析を使用して、自動で更新できます。CA Agile Requirements Designer のパス・インパクト分析機能は、破損したテストや無効なテストを削除したり修復して自動テストの失敗や過剰なテストを回避し、変更を検証するために必要な新しいテストを自動作成できます。つまり、テスト担当者は変更を検証するために必要なテストを、必要な数だけいくつでも実行できます。

図 D

CA Agile Requirements Designer のパス・インパクト・アナライザ分析によって、モデルから削除されたある論理が 32 個のパスを無効にしたことが特定されました。[すべて修復] をクリックすることで、パスは自動的に更新され、冗長なテストがあれば削除されます。



このアプローチは実施した作業の価値を最大化し、また、最新のテストとデータの回帰パックを迅速に提供して手作業による保守が引き起こすボトルネックを回避し、1 回のスプリントで厳密なテストを行うことを可能にします。技術系の利害関係者と非技術系の利害関係者の間の緊密なコラボレーションを促進し、考えを統一できます。たとえば BA がフローチャート・モデルに変更を行うと、その変更は自動的にテストケースに反映されます。

セクション 6

誤解 4 : フォーマルなモデル化はエキスパート以外には難しすぎる

この意見は、フォーマルなモデル化が実際にきわめて熟達した技術者のみが行える、高度な論理仕様であった時代から続いている考えのようです。[8] このようなモデル化技法は粒度が非常にきめ細かく、要件への可観測性の構築や欠陥検出の機能も非常に優れています。したがって、操作中に欠陥が見つかることが許容されない航空宇宙業界や軍事業界では、こうした方法を使用する必要があります。

しかしこのような方法は、上記の特殊な業界を除くほとんどの企業が採用している反復型開発プラクティスとは相いれません。このような方法は一般的なビジネス関係者にとっては高度すぎるのです。[9] 幸い BA など非技術系の利害関係者にとってなじみがある形式で、フォーマルなモデル化を導入することが可能です。

BA は VISIO のようなフローチャート・ツールや BPMN などの形式を幅広く使用しますが、フローチャートは計算によって正確性を確保でき、適切なツールを使用すればループや非線形制約のような高度な論理も含めることができます。その結果、フローチャートはテスト対象のシステムの論理を正確に反映できます。この正確性によってテスト担当者は上述のように要件を反映したテストを効率的に導出できます。

非技術系の利害関係者にとっては、実行可能なテストケースとテスト・データ、期待される結果を導出するために必要な粒度を備えたモデルを作成することが難しいのは確かです。しかし、フローチャート・モデルの価値はコラボレーションにあります。コラボレーションを行うことで、技術系と非技術系の両方の利害関係者が同じ考えで、役割のニーズに応じて異なるレベルの抽象化を使用して作業できます。

BA は CA Agile Requirements Designer にインポートされた設計を使用して、なじみのある形式やツールでより高いレベルのフローで作業できます。BPM、記述された文書や要件はすべてインポート可能です。BA が元の要件に変更を加える場合は、差異分析を使用して変更を特定し、フォーマルなフローチャートに反映させ、それにしたがってテスト・アセットを更新することができます。要件からテストへの変換作業の繰り返しを避けるために、テスト担当者は BA が実行した設計作業を活用します。

「Bloor Market Report:Test Case Generation」[10] の中で、Philip Howard 氏は多数のテストケース生成技法の強みと弱みを分析し、フローチャート･モデル化を支持しました。その理由はそれがフォーマル･アプローチのメリットを提供しながら、非技術系の利害関係者にとっても利用しやすいためです。フローチャートを使用すると、ペアワイズ・アプローチが提供するテストケースの最適化が可能になり、また、要件とテストケースの間に原因結果論理と追跡可能性がもたらされます。したがってフローチャートから直接、最適化したテストを導出でき、変更が起きた場合は自動保守が可能です。Howard 氏はフローチャート・モデルは「ビジネス・ユーザとのコラボレーションに非常に適している（少々適しているのではなく）」代替アプローチのすべてのメリットを提供するため、結局のところ、実際に実行が最も容易なアプローチであると結論づけています。

多くの組織にとっては、欠陥の観察と検出においてちょっとした妥協をすることで、ビジネスと IT のコラボレーションが可能になります。「本格的な」仕様技法は多くの企業が使用するアジャイル開発には適しません。しかしそれはフォーマルなモデル化を組み込むすべての技法が非技術系の利害関係者にとって難しすぎるということではありません。

セクション 7 :

誤解 5 : モデル化はアジャイルではない / アジャイルの枠組みに合わない

これについてはすでに説明した点が多く含まれます。モデルベースのテストはシフトレフトを可能にし、フローチャートの設計作業がテストの設計、データ・プロビジョニング、テストの保守など時間のかかるタスクの多くを自動化することは、強調するに値します。これによって以前の作業を活用でき、テストのより多くの部分が自動化されるため、スプリント内で厳密なテストが可能です。

フローチャート・モデルはまた、ビジネスおよび技術的なイニシアチブを緊密に整合させます。これはアジャイルのコンテキストではきわめて重要なことです。テスト担当者と設計者が一か所の参照先を使用して作業し、要件から抽出したテスト・アセットを自動で更新して、変更点をすべて反映できます。開発とテストは設計の直線的な段階をこのように 1 つにまとめることで多くの並行作業を行え、テスト担当者が設計をテスト・アセットに変換する必要があるときに発生する欠陥や時間の無駄を回避できます。スプリントはスプリントであって、テストが強制的に終了され継続的に後回しにされるような、単なる小規模なウォーターフォールとは異なります。

モデルベースのテストに関する詳細と、その実際の効果については、テストのエキスパートである Paul Gerrard 氏による CA ウェビナー **「What Came First—the Model or the Test?」** をご覧ください。

セクション 8 :

著者について

Huw Price は 2015 年にテスト専門ベンダの Grid-Tools 社が CA DevOps ポートフォリオに合併吸収された際に、アプリケーション・デリバリ担当副社長として CA に入社しました。30 年のキャリアを通じて現代の企業が直面する課題について熟知し、テストに関する深い理解によってそれらの問題の解決方法を心得ています。

Huw Price はテスト・モデルを一新する多数の革新的な製品のリリースに携わってきました。1988 年にはデータ・アーカイブ専門会社 BitbyBit を立ち上げ、長年のパートナーとなる Paul Blundell 氏がこれに加わりました。BitbyBit が買収された後、二人はデータの移行とアプリケーション変換を専門とする会社 Move2Open を共同で設立しました。

2004 年に二人は Grid-Tools Ltd を設立し、Huw Price は大企業のテスト方法の刷新を始めました。Datamaker（現在の CA Test Data Manager）の開発を監督し、テストへのデータ中心のアプローチを他に先駆けて開発し、後に Agile Designer（現在の CA Agile Requirements Designer）の設計と開発に先見的な役割を果たしました。

ca.com/jp/でCA Technologiesにアクセスしてください。

CA Technologies（NASDAQ : CA）は、企業の変革を推進するソフトウェアを作成し、アプリケーション・エコノミーにおいて企業がビジネス・チャンスを獲得できるよう支援します。ソフトウェアはあらゆる業界であらゆるビジネスの中核を担っています。プランニングから開発、管理、セキュリティまで、CA は世界中の企業と協力し、モバイル、プライベート・クラウドやパブリック・クラウド、分散環境、メインフレーム環境にわたって、人々の生活やビジネス、コミュニケーションの方法に変化をもたらしています。詳細については **ca.com/jp** をご覧ください。

1 Paul Gerrard、「Models at Heart」、CA Technologies、2016 年、https://www.ca.com/us/collateral/white-papers/models-at-heart.register.html

2 Philip Howard、「Automated test case generation」、Bloor Research、2015 年、https://www.ca.com/us/register/forms/collateral/bloor-research-spotlight-paper-automated-test-case-generation.aspx

3 Bender RBT、「Requirements Based Testing Overview」、Bender RBT、2009 年、http://benderrbt.com/Bender-Requirements%20Based%20Testing%20Process%20Overview.pdf

4 Soren Lauesen および Otto Vintro、「Preventing Requirement Defects:An Experiment in Process Improvement」、IT University of Copenhagen、デンマーク、2001 年、http://www.itu.dk/people/slauesen/Papers/PrevDefectsREJ.pdf

5 P Mohan、A Udaya Shankar、K JayaSriDevi、「Quality Flaws:Issues and Challenges in Software Development」、Hyderabad Business School, GITAM University、2012 年、http://www.iiste.org/Journals/index.php/CEIS/article/viewFile/3533/3581

6 同上

7 Philip Howard、「Automated test case generation」、Bloor Research、2015 年、https://www.ca.com/us/register/forms/collateral/bloor-research-spotlight-paper-automated-test-case-generation.aspx

8 Llyr Wyn Jones、「A Critique of Testing」、CA Technologies、2015 年、https://www.ca.com/us/collateral/white-papers/a-critique-of-testing.register.html

9 Philip Howard、「Bloor Market Report:Test Case Generation」、Bloor Research、2014 年 12 月 11 日、http://www.bloorresearch.com/research/market-report/test-case-generation/

10 同上

CS200-210233_0816