

手動テストを効果的に実施する

コードカバレッジを活用したテストケーステスト・探索テストの実施

はじめに

ソフトウェアの品質を改善するためには、ソフトウェアテストを行うことが一般的です。

開発プロジェクトでは単体テスト、結合テストやシステムテストなど、ソフトウェアの品質を確保・改善するために多くの種類のテストが実施されています。テストの手段にも多くの種類があり、単体テストをテスティングフレームワーク上で実施する、Eclipse や IntelliJ, NetBeans などの統合開発環境にてブレークポイントを設定しメモリ上のオブジェクトの値を直接確認する、膨大な数のテストシナリオやテストケースを作成し手動によりテストを行うなど、開発プロジェクトごとにさまざまな手段でテストが行われています。

本書では、開発プロジェクトにて実施されている「手動で行うテスト」に着目し、効果的に手動テストを行いソフトウェアの品質を改善させる方法を記載します。なお、ここでいう「手動テスト」とは、実際にソフトウェアを動作させ意図した処理が行われることをソフトウェアの画面(UI)を操作して行うテストのことを意味しています。

手動テストは特別なテストプログラムを必要とせずに簡単に実施できるため実施のハードルが低く、多くの開発者や品質保証部門の方が実施する機会の多いテストです。手動テストを今よりも効果的に実施するための 1 つのアイデアとして本書の内容が参考となれば幸いです。また、弊社取り扱い製品である Parasoft Jtest を用いた、効果的に手動テストを行うための具体的な手順についても記載します。Jtest の体験版をダウンロードしていただければ、すぐに本書に記載の効果的なテストが実施できます。体験版のご利用を合わせてご検討ください。

テストケーステストと探索的テスト

本書では、手動テストを効果的に実施する具体的なテスト手法として以下の 2 つのテストを取り上げます。

- **テストケーステスト**
 - テストの計画やテストケースなどのテストドキュメントを整備し、ドキュメントに従ってテストを行う手法
- **探索的テスト**
 - テスト対象のソフトウェアを学習し、テスト実施者の経験や知見、ソフトウェアの動作から得られたフィードバックをもとにテストを行う手法

コードカバレッジを利用し、テストケーステストと探索的テストを効果的に行う

コードカバレッジは、ソフトウェアテストによってソースコードが実行された割合を数値で表したもので、「手動テストにより、全体のソースコード行数のうちの 50%がカバーできた」というように、コードカバレッジはコードの全体量と計測したコード量から計算され、パーセントで表されます。

コードカバレッジは、開発プロジェクトにおけるテストの質を計測する指標です。テストにてどれくらいのコードがカバーできて

いるかを知ることができます。テストの過不足を判断する値として利用されます。今回はソフトウェアの実行時に計測するコードカバレッジ（アプリケーションカバレッジ）を用いて、テストケーステストと探索的テストを効果的に実施する方法をご紹介します。

テストケーステスト

テストケーステストは開発プロジェクトにてよく行われているテストであり、あらかじめ作成したテスト計画に沿ってテストを行う一般的なテスト手法です。テストケーステストは以下の流れで行われます。



図：テストケーステストの流れ

テストケーステストでは、開発プロジェクトの開始時または上流工程においてテスト計画を策定し、決められたプロセスでテストが行われます。

最近では TDD（テスト駆動開発）のように、XP（エクストリームプログラミング）のテストファーストを取り入れた開発も行われています。テストファーストでは、ソフトウェアに必要な機能について最初にテストを書き、テストが動作する最低限な実装を行い、その後さらに完成度を高めていくという流れで開発が行われます。テストファーストにおいても、テストドキュメントに相当するテストケースが作成され、そのケースに沿って実装およびテストが行われるため、テストケーステストの一種であるといえます。

テストケーステストの利点や懸念点には以下があります。

- 利点
 - 認知されているテスト手法であるため、テスト手法自体の学習コストが安い。（何をどうテストするかを決め、そのまま通りにテストを行うというわかりやすいプロセスである）
- 懸念点
 - テストを行うためにはテストドキュメントを作成する必要があり、作成にコストがかかる。
 - テストドキュメントの作成を開発の初期段階に終えることが難しい。ソフトウェアの要件や仕様が初期段階で確定しないケースが多く、入力値の範囲や上限が確定していない場合は、境界値を設定することができないため、テストケースが確定しない。
 - テストケースの抜けや漏れを検出することが難しい。テストの実施を完璧に実施できたとしてもテストケース自体に欠陥があれば、問題が発見されないままソフトウェアがリリースされてしまう。

テストケーステスト × コードカバレッジ

テストケーステストには

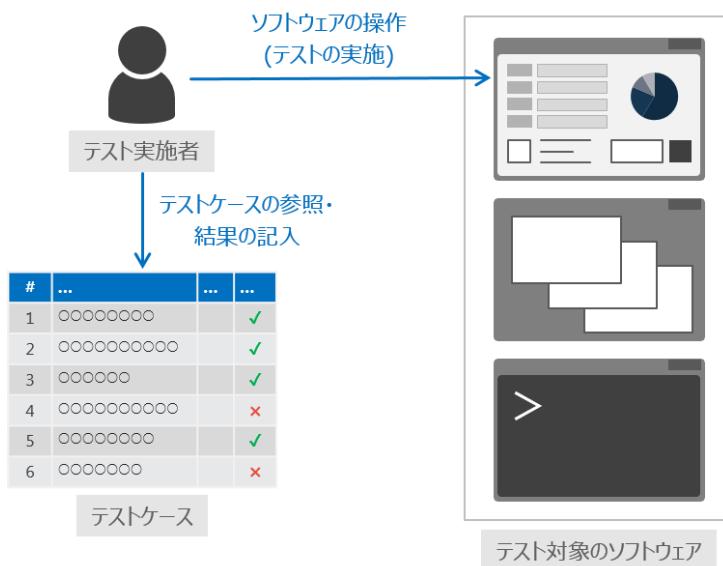
 テストケースの抜けや漏れを検出することが難しい。テストの実施を完璧に実施できたとしてもテストケース自体に欠陥があれば、問題が発見されないままソフトウェアがリリースされてしまう。

 という懸念点があります。

 テストケースに抜けや漏れがあることを検出するためにコードカバレッジを利用することで、従来のテストケーステストをより効果的に行うことができます。

従来のテストケーステスト

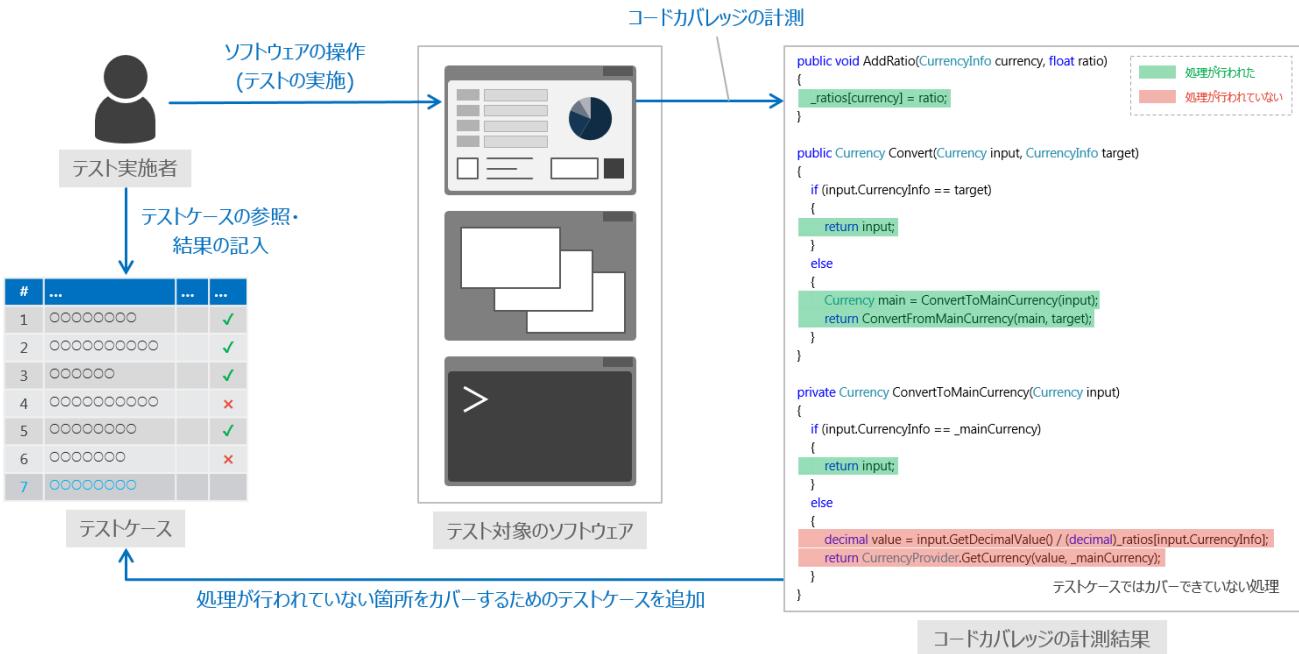
 テストケーステストでは、テスト実施者がテストケースをもとに開発ソフトウェアを操作しテスト結果を記録します。



図：テストケーステストの流れ

テストケーステスト × コードカバレッジ

 テストケーステストにコードカバレッジを利用した場合は、コードカバレッジの計測によりテストで**処理が行われていない箇所**が明らかになります。処理が行われていない箇所がテストの範囲内であれば、テストケースの抜け漏れとして該当の処理を網羅するテストケースを追加し、テストを改善します。



図：テストケーステストにコードカバレッジを利用した場合の流れ

なお、テストケースの抜けや漏れに加えてソースコード上に実装漏れが存在する場合は、コードカバレッジでは問題が検出できません。そのため、コードカバレッジの利用以外にもコードレビューを行うなどの活動が必要となることに注意してください。

Jtest のコードカバレッジ計測を用いたテストケーステストの実施

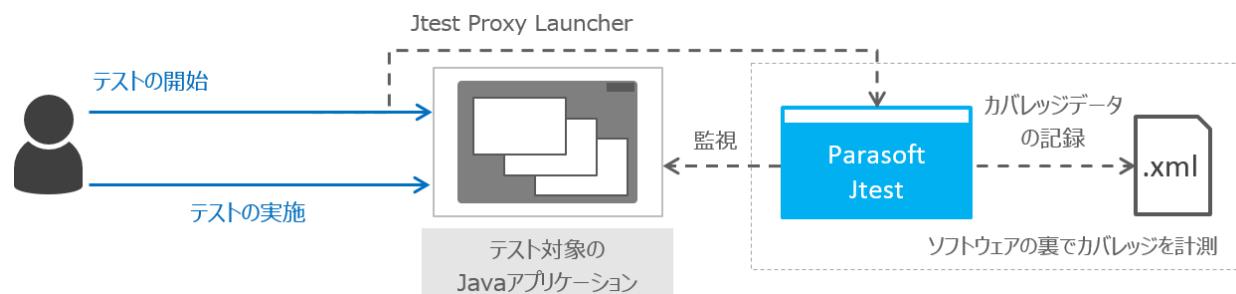
弊社取り扱いの Parasoft Jtest を用いることで、バッチやクライアントアプリケーション、Web アプリケーションのコードカバレッジを計測することができます。（C 言語に対応した C++test や C#、VB .NET に対応した dotTEST も取り扱っています。）

Jtest を用いてテストケーステストの際にコードカバレッジを計測する手順をご紹介します。

1. アプリケーションをビルド

テスト対象のアプリケーションをビルドします。

2. アプリケーションを起動（カバレッジ計測も起動）し、テストケーステストを実施



図：カバレッジ計測の手順 2

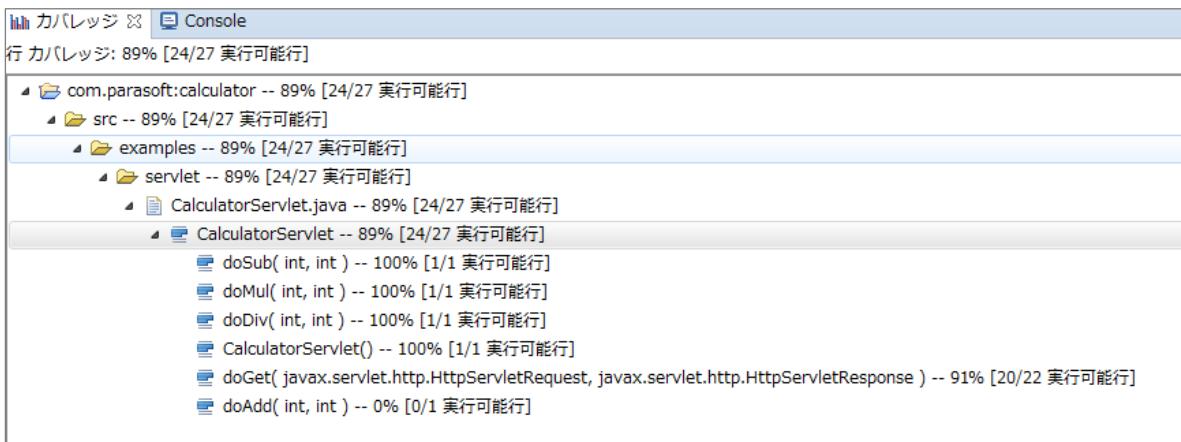
テストを行う際は、アプリケーションを起動する際に、Jtest のカバレッジ計測機能を同時に起動し、テストケースに従ってソフトウェアを操作し、テストを行います。この間、Jtest は開発ソフトウェアのどのコードが実行されたのかを監視し、カバレッジデータとして記録します。

3. 計測したカバレッジデータを統合開発環境にインポート



図：カバレッジ計測の手順 3

テストが完了したら Jtest が記録したカバレッジデータを Eclipse 等の統合開発環境へインポートし、結果を確認します。コード上にどの処理が行われたか／行われなかったかを表示しますので、処理が行われなかった箇所を確認し、テストケースに抜けや漏れがないかを確認します。



- 統合開発環境上にはテストにて計測したカバレッジデータの概要（実行可能行とテストにて実行された行）が表示されます。

```

17  protected void doGet(HttpServletRequest request, HttpServletResponse response)
18  throws ServletException, IOException
19  {
20      PrintWriter out = response.getWriter();
21
22      String sP1 = request.getParameter("p1");
23      int iP1 = Integer.parseInt(sP1);
24
25      String sP2 = request.getParameter("p2");
26      int iP2 = Integer.parseInt(sP2);
27
28      String sProcess = request.getParameter("process");
29
30      int result = 0;
31      if (PROCESS_ADD.equals(sProcess)) {
32          result = doAdd(iP1, iP2);
33      } else if (PROCESS_SUB.equals(sProcess)) {
34          result = doSub(iP1, iP2);
35      } else if (PROCESS_MUL.equals(sProcess)) {
36          result = doMul(iP1, iP2);
37      } else if (PROCESS_DIV.equals(sProcess)) {
38          result = doDiv(iP1, iP2);
39      }
40
41      response.setContentType("text/html");
42
43      out.println("<html><body>");
44
45      out.println("<h2>Calculator</h2>");
46      out.println("Result: " + result);
47
48      // back button
49      out.println("<FORM><INPUT Type=\"button\" VALUE=\"Back\" onClick=\"history.go(-1);return true;\"></FORM>");
50
51      out.println("</body></html>");
52  }

```

- また、コードエディター上には処理が行われた行は緑／行われなかつた行は赤でハイライトして表示されます。

図：カバレッジの計測結果を Eclipse にインポートした例

4. 処理が行われていない箇所の確認、テストケースの更新

Jtest のカバレッジ計測の結果から、処理が行われていない箇所が存在する場合は、その処理がテストの範囲内なのかどうかを検討します。以下にカバレッジの計測結果からテストケースの追加に至る簡単な例を記載します。

<pre> @WebServlet("/Calculator") public class CalculatorServlet extends HttpServlet { private static final long serialVersionUID = 1L; protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException, IOException { PrintWriter out = response.getWriter(); String sP1 = request.getParameter("p1"); int iP1 = Integer.parseInt(sP1); String sP2 = request.getParameter("p2"); int iP2 = Integer.parseInt(sP2); String sProcess = request.getParameter("process"); int result = 0; if (PROCESS_ADD.equals(sProcess)) { result = doAdd(iP1, iP2); } else if (PROCESS_SUB.equals(sProcess)) { result = doSub(iP1, iP2); } else if (PROCESS_MUL.equals(sProcess)) { result = doMul(iP1, iP2); } else if (PROCESS_DIV.equals(sProcess)) { result = doDiv(iP1, iP2); } } } </pre>	■ 処理が行われた ■ 処理が行われていない
--	---

このアプリケーションでは、指定された数値の四則演算を行います。

未処理の箇所について、開発者に確認したところ、演算を行う機能について、除算が行われる場合に、この行の処理が行われるようです。

除算を行うテストケースの有無を追加し、ケースが無ければ追加します。
この例では除算においても加算を行うコードが記述されており、コードのコピー & ペーストによりバグが埋め込まれているため、テストが漏れたままだとバグがそのままリリースされてしまいます。

図：未処理箇所およびテストケース追加の例

5. 更新されたテストケースを用いて再度テストケーステストを行う

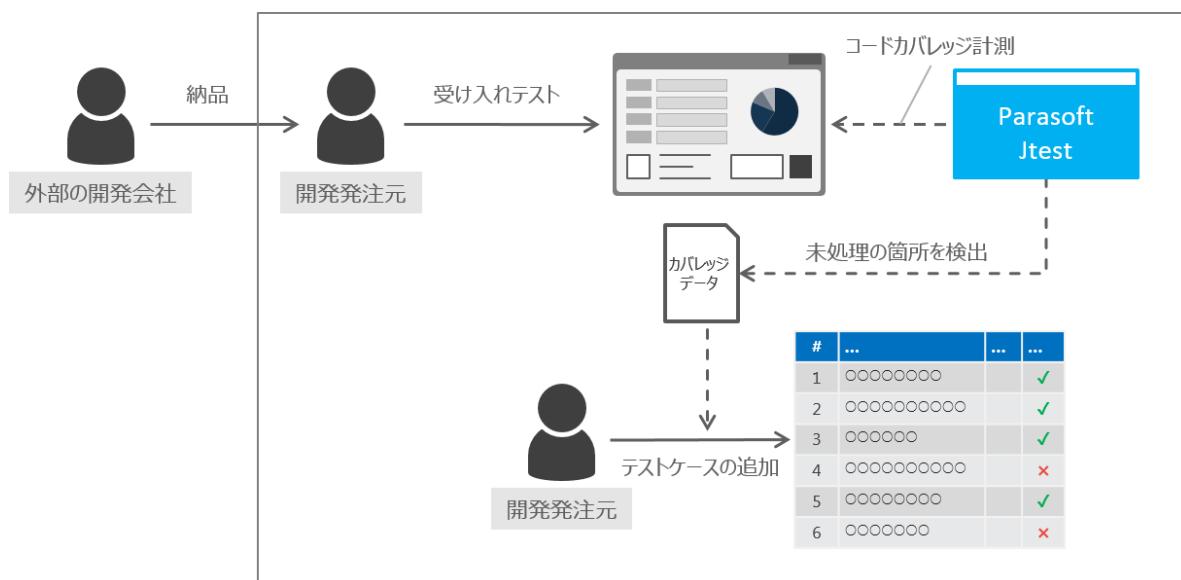
テストケースに漏れが見つかった場合は、再度テストケーステストを行います。テスト範囲の機能をコードレベルで網羅的にテストできるまで繰り返します。テストの実施結果を残すと同時に、コードカバレッジの計測結果のうち、未処理である箇所について報告できるようソースコードファイルごとのカバレッジの値などを記録しておきます。

改善事例

Jtest のコードカバレッジ計測を利用し、テストケーステストが改善された例を 2 つ紹介します。

例 1：テストシナリオの抜け漏れを検知し、受け入れテストを改善

ソフトウェア開発を外部へ委託し、納入時に受け入れテストを実施している開発プロジェクトにおいて、コードカバレッジを利用することで、受け入れテストの抜け漏れを検出しテストが改善した実績があります。

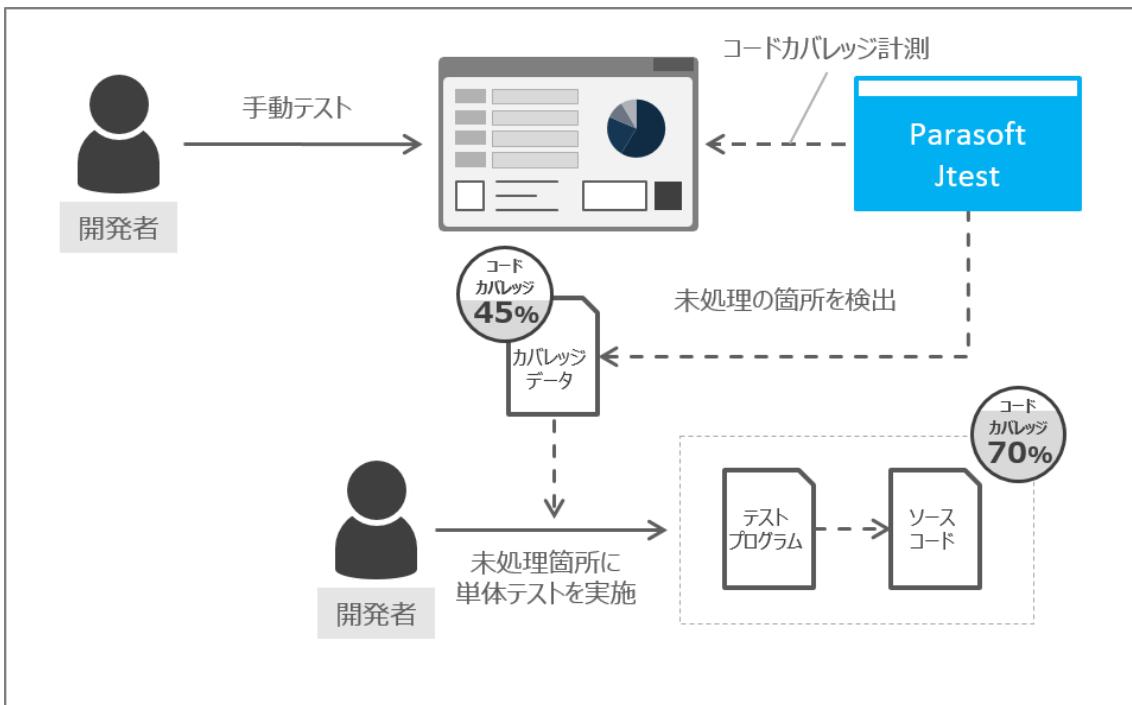


図：改善されたテストケーステスト 1

この例では、要件や仕様をもとに作成したテストケースにてテストを行い、コードカバレッジの計測結果からテストケースの不足箇所を特定し、テストケースへ反映、再度テストを実施しテストの網羅性を改善させています。

例 2：カバレッジの目標水準の達成

カバレッジの達成目標が設定されている開発プロジェクトにおいて、すべてのコードに対して単体テストを実施する代わりに、手動のテストケーステストにコードカバレッジ計測を利用してすることで、効率的にテストが実施できた例があります。



図：改善されたテストケーステスト 2

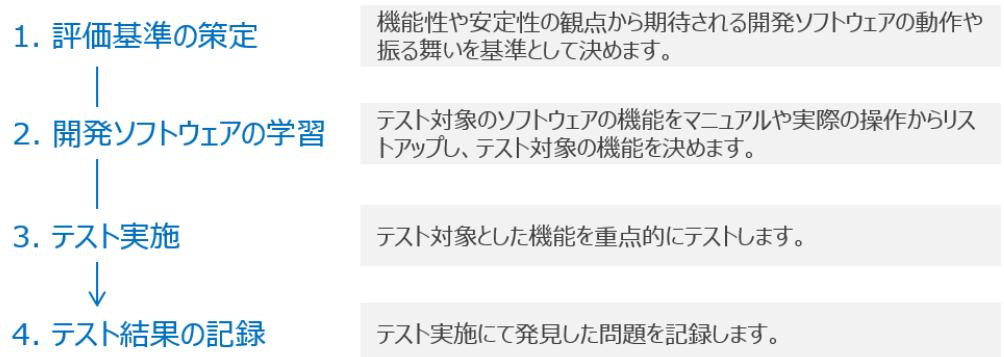
この例では、手動テスト+コードカバレッジ計測にてテストをまず行い、おおまかに開発ソフトウェア全体をカバーし、コードカバレッジの結果からテストが実施されていないコードを見つけています。テストが実施されていないコードについては、追加の手動テストや単体テストを行うことでコードカバレッジの目標水準を達成しています。

このようにテストケーステストにおけるコードカバレッジでは、ソフトウェアの"テストされていない部分"を発見するために大いに役に立ちます。

探索的テスト

探索的テストは 1983 年に Cem Kaner が「テスト手順書などのテストドキュメントには依存せず、開発ソフトウェアの分析結果やフィードバックを活用する手法」として提言しました。テストケーステストとはアプローチが異なり、探索的テストではテスト設計とテスト実施をテスト実施者が頭の中で同時にいながらテストを行います。

探索的テストは特に決まったプロセスがあるわけではありませんが、例として以下ののような流れで行うことができます。



図：探索的テストの流れ

探索的テストでは、他のシステムと通信する処理や例外の発生する処理、初期値と設定値により動作が切り替わる処理などを含む機能（要は“弱そうな機能”）をテスト対象の機能とします。また、開発者へ複雑な処理を含む機能をヒアリングすることで、テストの対象を決定する場合もあります。

このように、探索的テストでは開発ソフトウェアの特性を理解した上で問題が検出されそうな箇所を重点的にテストします。テストドキュメントは評価基準や調査した機能一覧など最低限のものに留め、テスト結果も検出した問題のみを記録します。ドキュメントの作成にかかるコストをテスト実施に充当してテストを実施します。

テストケーステストと探索的テストにおける問題のタイプと検出能力を表したデータを以下に記載します。

タイプ	探索的テスト	テストケーステスト	探索的テスト / テストケーステスト(%)
ドキュメンテーション (Documentation)	8	4	200%
GUI	70	49	143%
矛盾点 (Inconsistency)	5	3	167%
機能欠如 (Missing function)	98	96	102%
パフォーマンス (Performance)	39	41	95%
技術的バグ (Technical defect)	54	66	95%
ユーザビリティ (Usability)	19	5	380%
間違った振る舞い (Wrong behavior)	263	239	110%

引用：高橋寿一, 『知識ゼロから学ぶソフトウェアテスト【改訂版】』, 翔泳社, 2013

探索的テストはテストケーステストと比較し、ユーザビリティに関する問題の検出率が4倍の結果が出ています。このように、探索的テストでは、テストケーステストでは検出しにくい問題を多く検出できる可能性を秘めています。

探索的テストの利点および懸念点は以下があります。

- 利点
 - 仕様や要件に依存せずにソフトウェアの問題を検出できる。テストケーステストでは仕様や要件をもとにテストケースを作成しますが、探索的テストでは特定の機能の問題検出に集中してテストを行うため、仕様や要件に記載されていないような問題についても検出することができます。
 - テストケーステストと比較して、テスト計画やテスト設計、仕様変更などに伴うテストドキュメントのメンテナンスコストが低い。
- 懸念点
 - テスト実施者に知識や経験が要求される。
 - パフォーマンスやセキュリティのテストには向いていない。（これらのテストにはテストケーステストや探索的テスト以外のアプローチが必要です。）
 - テストのエビデンスが残りにくい。

探索的テスト × コードカバレッジ

探索的テストには

テストのエビデンスが残りにくい。

という懸念点があります。

探索的テストでは、テストを実施しながらテスト設計を同時にを行うため、テストドキュメントが残りにくく、テストを行った機能と操作を残すことが難しいケースが多く存在します。テスト対象の機能を評価基準として記載する場合もありますが、問題が検出されない限りは具体的な操作について記録することは稀です。

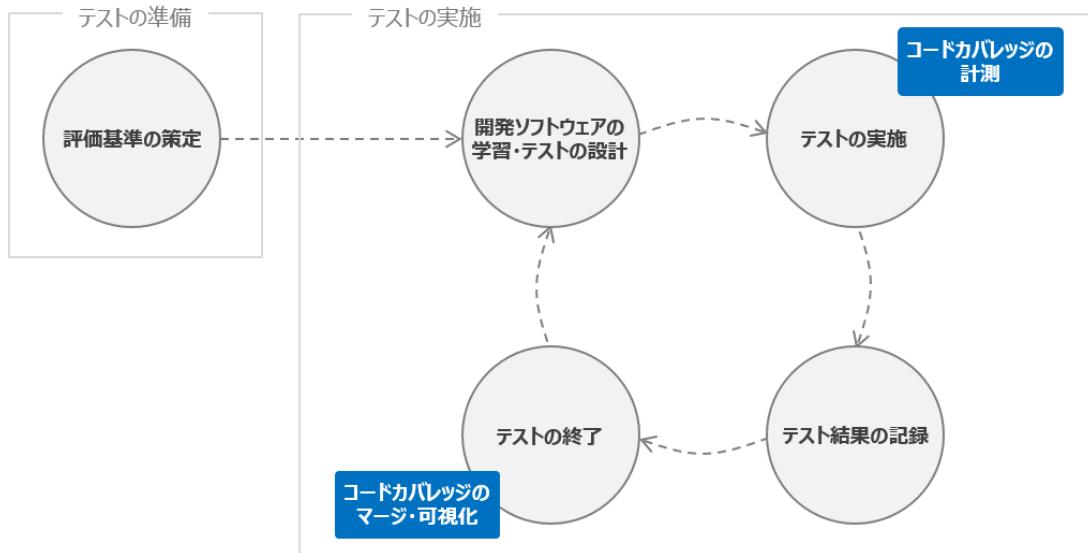
コードカバレッジを利用し探索的テストの記録を残す仕組みを構築し、探索的テストを効果的に実施できるようにします。コードカバレッジを利用した探索的テストには以下の2つの利点があります。

- コードカバレッジの計測により、テストで処理が行われた箇所が明らかになる。
 - テストの記録を残すことができます。
- 探索的テストにて問題を検出した場合にどの処理が実施されたかが明らかになる。
 - 探索的テストでは、テスト実施者が開発ソフトウェアを学習し、弱そうな箇所をテストします。テストケースができるような複雑なケースがテストされるため、問題を検出した場合にどの処理が行われたかを客観的なデータとして残すことで、その後の調査が効率的に行えます。

Jtest のコードカバレッジ計測を用いた探索的テストの実施例

Jtest を用いて探索的テストの際にコードカバレッジを計測する手順をご紹介します。Jtest によるコードカバレッジの計測はテストケーステストと同様のため手順の記載を省略し、代わりに複数のテスト実施者のテストにて計測したコードカバレッジを統合して管理する例を交えて説明します。

テストの一連の流れは以下の図の通りです。



図：コードカバレッジ計測を含む探索的テストの流れ

1. テストの準備

探索的テストでは、テストの評価基準を策定することからはじめます。

テストの評価基準の策定

- 実施する探索的テストのテーマ（特定機能の動作検証、ユーザビリティの確認など）を決め、さらにテストの合否基準（探索的テストの合格基準）を定める。

2. テストの実施

探索的テストでは、テストケーステストのアプローチとは異なり、テストのテーマに沿って問題のありそうな箇所をテスト対象とします。詳細なテストケースは作成せず、おおまかなテスト対象の機能や操作のみを決定し、テストの実施中に臨機応変に問題のありそうな箇所をテストします。テストケースを作成しない（どういった操作でテストを行ったかを記録しない）点をコードカバレッジの計測で補います。

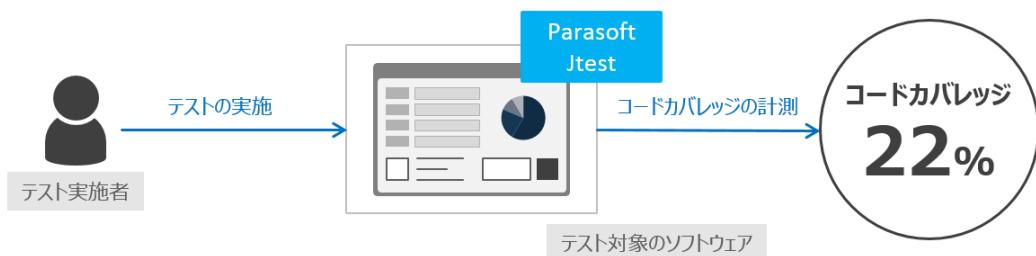
開発ソフトウェアの学習およびテストの設計

- 開発ソフトウェアの機能をリストアップします。
 - 開発ソフトウェアを実際に動かし、基本的な動作を確認します。
 - 開発ソフトウェアを実際に動かし、入力や出力、ダイアログの表示、各種マウス操作の結果、キーボード操作の結果などを確認します。
 - マニュアルやヘルプなどの文書から機能を抽出します。
- 問題を含んでいそうな機能（弱そうな機能）をリストアップします。

- データやファイルの入出力、外部システムとの接続が必要な処理、ネットワーク切断が発生した場合の復帰処理などが存在する機能を調べます。
- 開発者へヒアリングし、開発者が複雑だと考えている処理、仕様や設計の決定に時間と労力を費した機能などを洗い出します。
- 過去の類似開発ソフトウェアで発生したデグレードや、開発チームが過去に発生させた障害などから、同類の問題が発生しそうな機能がないか確認します。

テストの実施

- 開発ソフトウェアを **Jtest のコードカバレッジ計測を有効にした状態で起動**（具体的な手順はテストケーステストの手順を参照）し、学習および設計した内容に基づきテストを行います。
 - テスト実施者の判断で操作をアレンジし、問題のありそうな遷移先の機能を含めて臨機応変にテストを行います。
 - テスト実施者の操作により、Jtest がコードカバレッジを計測します。



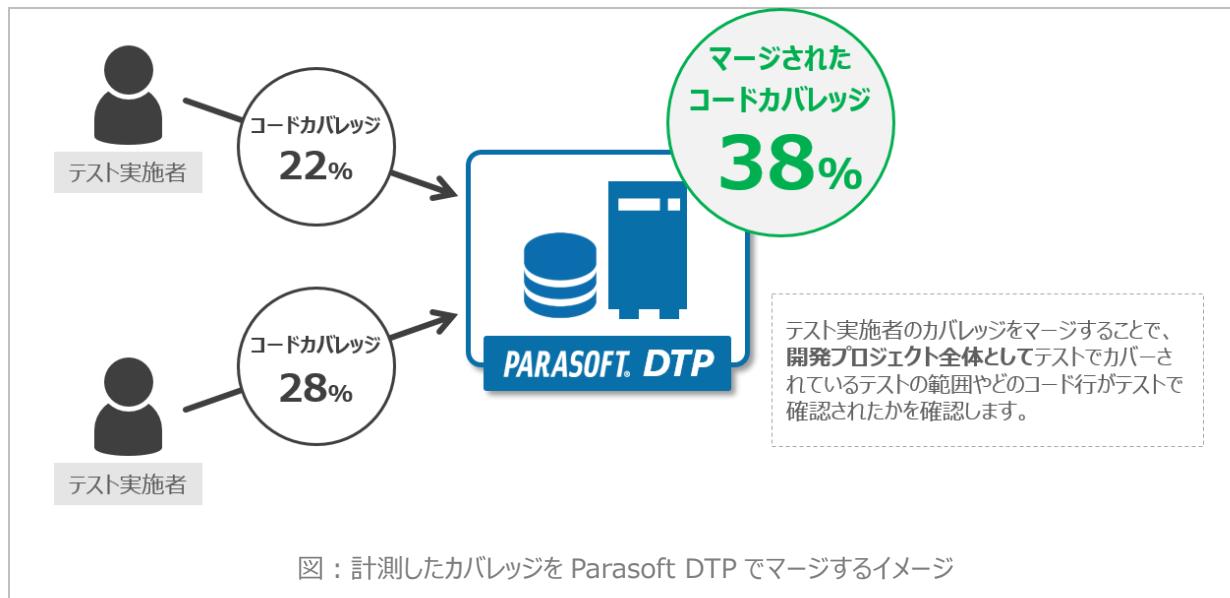
図：カバレッジ計測のイメージ

テスト結果の記録

- テスト中に問題を発見した場合は、テストした機能や操作、手順、症状などを記録し開発チームへ報告を行います。
 - コードカバレッジの計測結果を確認し、ソースコードレベルでどの処理をテストした際に問題が発生したかを確認することで、1 次調査が効率的に行える可能性があります。

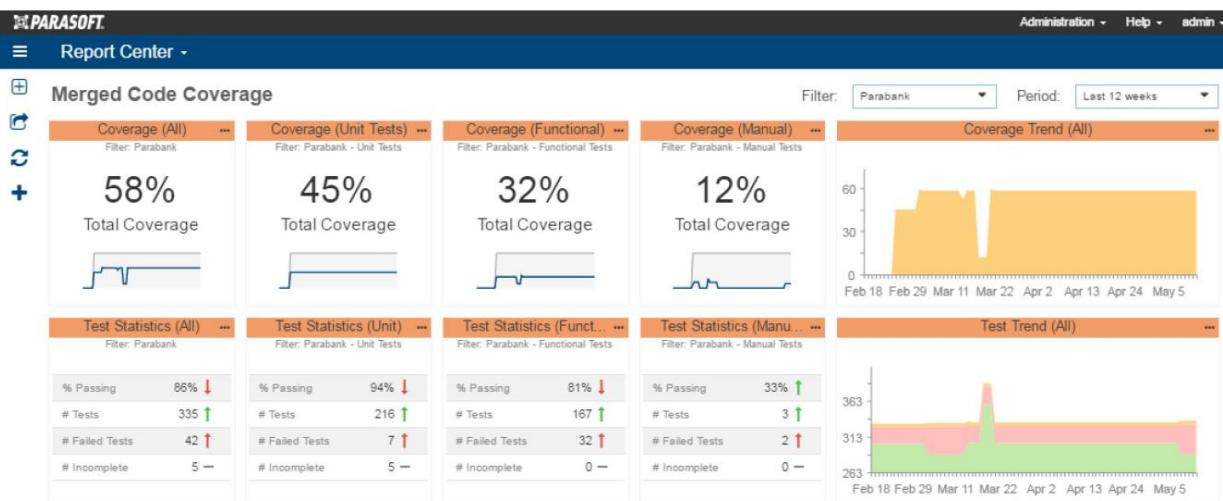
テストの終了

- テスト結果の分析やテストで学習した内容を踏まえて、再度テストを実施するための準備を行います。
- コードカバレッジの計測結果を確認し、**計測したコードカバレッジを Parasoft DTP へ登録し、他のテスト実施者のコードカバレッジとマージし管理します。**



図：計測したカバレッジを Parasoft DTP でマージするイメージ

以下は Parasoft DTP 上の可視化されたカバレッジデータの表示例です。テストにて計測された最新のコードカバレッジの情報をいつでも確認することができます。



図：Parasoft DTP のカバレッジの表示

探索的テストにはテストのエビデンスが残りにくいという懸念点があります。探索的テストのプロセスにコードカバレッジの計測を組み込むことで、テストされた箇所を明らかにし、テスト実施者がより問題の検出に注力できるようになります。

まとめ

手動テストは特別なテストプログラムを必要とせずに簡単に実施できるため実施のハードルが低く、多くの開発者や品質保証部門の方が実施する機会の多いテストです。手動テストを効果的に行なうことは、開発ソフトウェアの品質向上に大きく寄与します。今回はテストケーステストと探索的テストを取り上げ、それぞれのテスト手法の概要やテストの流れを説明とともに、懸念点を補う方法としてコードカバレッジを利用した例を紹介しました。

- テストケーステスト
 - テストケーステストにコードカバレッジを計測する仕組みを取り入れます。
 - コードカバレッジにより、テストされていない箇所が明らかになるため、テストケースの抜け漏れを発見しテストケースを改善するのに役立ちます。
 -
- 探索的テスト
 - 探索的テストにコードカバレッジを計測する仕組みを取り入れます。
 - コードカバレッジにより、探索的テストを行っている際にテストされた箇所が明らかになるため、コード行レベルでテストが行われた記録が残せるようになります。

コードカバレッジを利用し、これまで以上に手動テストを効果的に実施しましょう。

Parasoft について

Parasoft は、欠陥のないソフトウェアの効率的なデリバリーを支援するソフトウェア ソリューションの研究開発に取り組んでいます。SDLC を加速する一方で、ソフトウェアの欠陥に関するリスクに対処するため、Parasoft は Development Testing Platform および継続的テスト プラットフォームを提供しています。Parasoft のエンタープライズ向けおよび組み込み向けの開発ソリューションは、業界で最も包括的なものです。その範囲は静的解析、単体テスト、要件のトレーサビリティ、カバレッジ解析、機能テストと負荷テスト、開発 / テスト環境などに渡ります。Fortune 500 の大半の企業が、アジャイル、リーン、DevOps、コンプライアンス、セーフティ クリティカルな開発のための取り組みを進める中、最高品質のソフトウェアを一貫して効率的に生産するために、Parasoft のソリューションを頼りとしています。

技術資料および体験版

Jtest 体験版



- ご利用になれる期間は 14 日間です。
- 本書でご紹介したコードカバレッジ計測や、他の機能である静的フロー解析を含む、すべての機能を無償でご利用いただけます。
- ご評価を円滑に進めるための技術的なお問い合わせも受け付けております。

Jtest 技術資料



- 以下の技術資料をご提供しております。
 - ソフトウェアの品質を向上させるためのアイデアや事例などの技術情報
 - 弊社が過去に開催したセミナーのプレゼンテーション資料

お問い合わせ先



テクマトリックス株式会社

システムエンジニアリング事業部 ソフトウェアエンジニアリング営業部

<https://www.techmatrix.co.jp/product/jtest/>

parasoft-info@techmatrix.co.jp