

QA エンジニアへのインタビューによる欠陥検出方法の調査

谷崎 浩一

株式会社ベリサーブ

kouichi.tanizaki@veriserve.co.jp

吉川 努

株式会社ベリサーブ

tsutomu.yoshikawa@veriserve.co.jp

蛭田 恭章

株式会社ベリサーブ

yasuaki.hiruta@veriserve.co.jp

鈴木 梨沙

株式会社ベリサーブ

risa.suzuki@veriserve.co.jp

東島 恵美子

株式会社ベリサーブ

emiko.higashijima@veriserve.co.jp

森崎 修司

名古屋大学

s.morisaki.jp@ieee.org

要旨

ソフトウェア開発において、QA エンジニアは開発者が見逃しやすい欠陥を検出する重要な役割を担っている。しかし、QA エンジニアがテストを行う際にどのような方法で欠陥を検出しているのか、具体的なノウハウに関する研究は十分に行われていない。本研究ではQA エンジニアへのインタビューを通じて欠陥検出時の操作を収集し、それらを分析することで欠陥のトリガーと、それを起こす具体的な方法を明らかにした。少ないテストケースで欠陥を検出できるQA エンジニアは欠陥のトリガーとそれを起こす方法を多く知っており、それにより多くの欠陥を検出できると考えられる。

1. はじめに

ソフトウェア開発プロジェクトにおいて、QA(Quality Assurance)を専門の役割とするQA エンジニアがプロジェクトチームにアサインされることが多くある。QA エンジニアがテストを行うことで、開発者には見つけられない欠陥を見つけることができる。QA エンジニアの欠陥検出方法を明らかにし活用できるようにすることは重要だが、QA エンジニアが欠陥検出のために行っている具体的なノウハウについての議論は十分ではない。

QA エンジニアの経験をベースとするテスト技法としてエラー推測が知られる。エラー推測ではQA エンジニアの経験や知識に基づき、エラー、欠陥、故障の発生を予測してテストを行う[1]。エラー推測による効率的に欠陥を検出するテストケースを作成するためには、実際に起こり得るエラー、欠陥、故障を推測する必要がある。その推測はQA エンジニア個人の経験に依存しやすいものとなっており、有効なテストを作成できるかどうかは経験に大きく依存する。エラー推測を実施するためのアプローチとして、フォールト攻撃が知られる。フォールト攻撃ではエ

ラー、欠陥、故障のリストを利用してテストを行うが、そのリストをどのような構造で作成するとQA エンジニアのノウハウを適切に活用できるか明らかでない。

本研究では、QA エンジニアがテストにおいて欠陥検出時に実施した操作に着目し、テストにおける欠陥検出のノウハウを明らかにすることを目的とする。欠陥はソフトウェアが問題のある状態(ある入力を受け付けたときに正しい出力を出せない状態)になることで発生する。欠陥を狙って検出するには、問題のある状態を知り、その状態を起こす方法を知っている必要があると考えられる。本研究では、問題のある状態を「欠陥のトリガー」、それを意図的に発生させる方法を「欠陥のトリガーを起こす方法」と呼ぶ。欠陥のトリガーとそれを起こす方法に着目してQA エンジニアのノウハウを可視化する。

2. 関連研究

QA エンジニアのテスト設計の思考を整理してテスト設計の手法を構築した研究として、テスト対象の弱い部分・欠陥の存在が懸念される部分である「脆弱性」と、テスト対象の外部から与える厳しい条件である「悪条件」を考慮したテスト設計手法が提案されている[2]。QA エンジニアは脆弱性と悪条件を認識することで欠陥を検出しやすいテストケースを作成していると考えられる。脆弱性と悪条件の関係は、本研究で着目する欠陥のトリガーとそれを起こす方法の関係に類似している。しかしながら、脆弱性と悪条件の対応関係について言及のない部分があり、欠陥検出のための具体的な方法が不明確である。

河野らは、ソフトウェア開発において欠陥が発生する要因となる曖昧な外部仕様記述に着目し、テスト項目を設計する方法を提案している[3]。200 個のテスト項目事例を分析した結果から 27 個の後工程欠陥要因の観点を明らかにし、それぞれの観点に対応するテスト項目設計の考慮点を示している。本研究で着目する欠陥のトリガ

一とそれを起こす方法の整理方法として参考になる。

Web サービスやモバイルアプリケーション開発におけるテスト観点の整備に関する研究として、河野らはテスト観点を整理するための枠組みを定義したうえで、枠組みを活用して既存のテストスイートを分析し約 700 個の標準テスト観点を導出している[4]。枠組みに基づいて実際のテストスイートを分析し着眼点を導出するアプローチは、本研究における QA エンジニアが実施した操作の分析に通じ、分析方法の参考になる。

Deak は、4 つのソフトウェア開発企業における観察調査から 9 つのテストセッション戦略(テスト設計)と 13 個のテスト実行技術があることを特定している[3]。例えば、テストセッション戦略として、ソフトウェアの潜在的な弱点を見つける弱点戦略やある機能を複数の異なる方法でテストする代替方法のテストが挙げられている。しかしながら、これらの戦略や技術を基に具体的なテストケースをどのように作成すればよいかという本研究が目的とする具体的なテストケース作成方法は言及していない。

Madalina は、産業界からの要望が求人に現れると仮定し、求人広告を調べている[6]。調査は、特定のツールの利用スキル、対人スキル、テスト設計スキル、チームワークスキルを調べている。具体的なテストケース作成に必要なスキルに関して、仕様書等のドキュメントからテスト実行が可能な部分を抽出するスキルが挙げられている。しかしながら、抽出に必要な具体的な知識やテストケース作成が効率的になるスキルや要因には言及がない。

Basili らは、欠陥検出の効率性をコードリーディング、機能テスト、構造テストの 3 つの方法で比較している[7]。しかしながら、これらの方法でどのような知識を活用するかやエラーを推測するかは言及していない。

Iivonen らは、優秀なテスターの特性を複数の企業を対象としたケーススタディにより調査している[8]。具体的には、経験、内省、モチベーション、人格の 4 つを挙げている。また、ドメイン知識と技術スキルが重要であることを報告し、特に、テストケース設計やテスト計画の汎用的なスキルよりも製品知識や顧客の業務プロセスへの理解が重視されていることを報告している。しかしながら、具体的な製品知識や業務プロセスがどのようにエラー推測や効率的なテストケース作成につながるかは言及していない。

3. 前提・対象・方法

QA エンジニアが検出した欠陥を対象とし、どんな操作によってその欠陥を検出できたかをインタビューし、イン

タビュー結果を分析して欠陥のトリガーとそれを起こす方法を明らかにした。

3.1. 前提

ISTQB が公開する「ソフトウェアテストで使う標準用語集」の日本語版では、「エラー」は「間違った結果を生み出す人間の行為」、**「欠陥」**は「作業成果物に存在する、要件または仕様を満たさない、または意図した使用を妨げる不備または欠点」、**「故障」**は「コンポーネントやシステムが、指定した範囲内でその要件を満たさない事象のひとつ」とされている[9]。本研究では ISTQB の定義の「欠陥」と「故障」を同一視し、欠陥と記載する。

本研究の全体像を図 1 に示す。まず、QA エンジニアへのインタビューにより、テストにおいて欠陥検出の際に行った操作を収集する。次に、収集した操作を分析し、その操作が狙っていたと考えられる欠陥のトリガーを定義する。さらに、欠陥のトリガーごとに、それを起こすための具体的な方法を定義する。こうして得られた欠陥のトリガーとそれを起こす方法の一覧は、欠陥検出に活用可能な知見であり、本研究の主要な成果である。本研究ではこの成果に対し考察を加え、欠陥のトリガーを定義した意義、実務における活用の可能性、今後の拡張性について議論する。

3.2. 対象

インタビューの対象者はあるソフトウェア開発プロジェクトにてテストの業務に従事した QA エンジニア 4 名で、QA・テストに関する業務経験は 1~3 年だった。対象のプロジェクトは、開発工数が 3~4 人月程度の商用製品の開発プロジェクトである。

当該 QA エンジニアがシステムテスト工程で検出した欠陥を次の 2 つに分類し、後者の欠陥 B' を対象としてインタビューを行った。

- 欠陥 B: テストケースの事前条件・手順・期待結果に従ってテスト実行し、テスト対象が期待結果と異なる動作をしたことで検出された欠陥。
- 欠陥 B': 期待結果に記載されていない動作に対して、テスト実行者が期待通りではないと判断して検出された欠陥。

図 2 に欠陥 B と欠陥 B' の例を示す。欠陥 B' はテストケース以外で QA エンジニアの気付きによって検出されたものであり、QA エンジニアの欠陥検出方法を明らかにする上で有用であると考え、インタビューの対象とした。

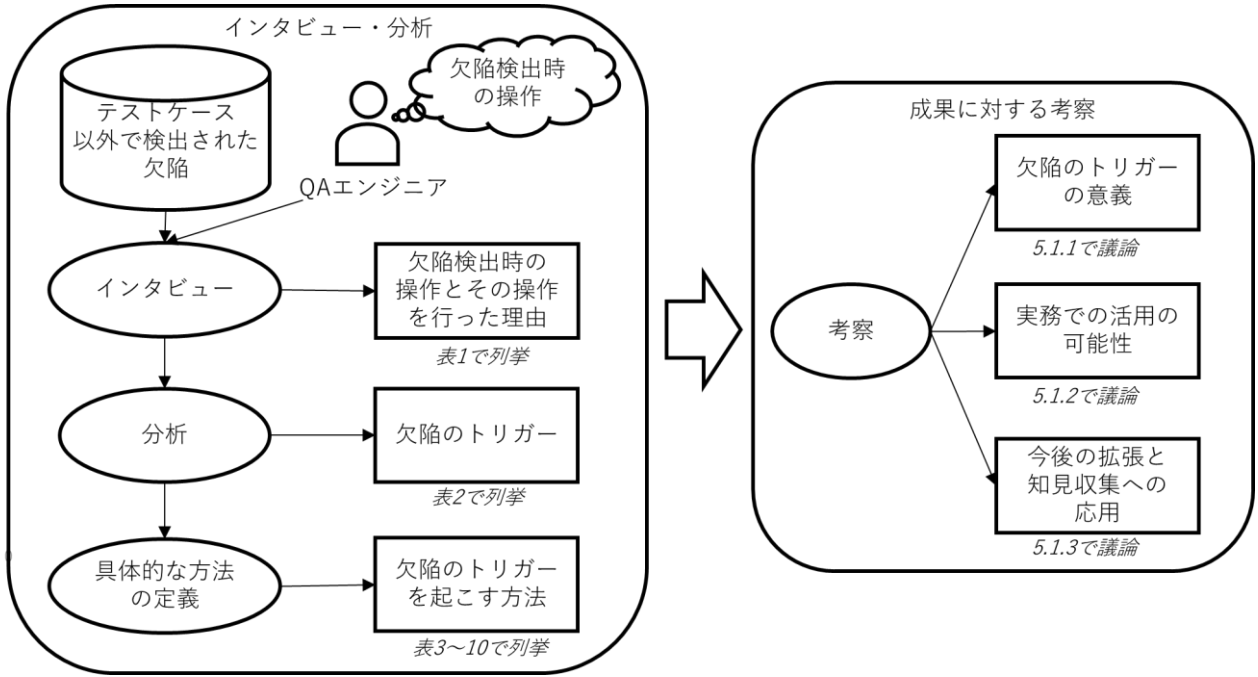


図1 本研究の全体像

3.3. 方法

QA・テストに関する業務経験が7年のQAエンジニア1名がインタビューを実施した。インタビューは対面で、インタビュアーとインタビューイが1対1で対話する形式とした。

■ECサイトのテストケースの例

事前条件	手順	期待結果
カートが空	購入ボタンをクリックする	「カートが空です」とメッセージが表示される
...

■テストケースで検出される欠陥Bの例

「Empty.」というメッセージが表示され、期待結果のメッセージと異なる

■テストケース以外で検出される欠陥B'の例

ボタンをダブルクリックしたらカートが空のままで購入確認ページに遷移した

図2 テストケースで検出される欠陥とテストケース以外で検出される欠陥の例

インタビューおよびインタビュー結果の分析の手順は以下の通りである。

1. 対象者が検出した欠陥に対して、その欠陥を検出した際の操作とその操作を行った理由をインタビュアーがインタビューした。
2. 筆者らがインタビュー結果を確認し、狙いがなくたまたま行っただけの操作はノウハウを含まないと考え、以降の分析手順に含めないようにインタビュー結果から除外した。
3. 欠陥を検出した際の操作を対象として、同じ欠陥のトリガーを狙ったと考えられる操作を筆者らの議論に基づきグルーピングし、欠陥のトリガーの名称を定義した。
4. 欠陥のトリガーごとに、それを起こす方法を筆者らの議論に基づき定義した。

4. 結果

手順1の結果、表1に示した36件の操作とその操作を行った理由を得た。これらの操作には、QAエンジニア自身が思いついて行ったもの(ID1-1~1-18)、上司からの指示があり意識的に行ったもの(ID1-19~1-26)、テストの準備や実施の中でたまたま行ったもの(ID1-27~1-36)が含まれていた。

表 1 欠陥検出時の操作とその操作を行った理由

ID	欠陥検出時の操作	その操作を行った理由	分析対象
1-1	通常では押せないボタンを他の方法で押せるようにした	できないことを無理やりできる手順をした	対象
1-2	通信のやり取りを見ていた	中身を見たほうが良いと思った	対象
1-3	テストケース実施終了後の動作も気になってやってみた	テストケースの間も気になった	対象
1-4	テストで使った登録を削除した	同上	対象
1-5	ボタンを押してみた	やれることをやってみた	対象
1-6	いろいろな値にしてみた	同上	対象
1-7	ブルダウンを押してみた	同上	対象
1-8	二重起動を試してみた	同上	対象
1-9	設定を変更してみた	同上	対象
1-10	重複登録を行ってみた	同上	対象
1-11	怪しそうな負荷を与えてみた	同上	対象
1-12	画面の移り変わりを素早くしてみた	同上	対象
1-13	見てない画面がないかを確認してみた	同上	対象
1-14	ペアでテストをしてみた	テストのやり方を変えてみた	対象
1-15	画面の確認を行うテストで音出力の確認を行ってみた	同上	対象
1-16	同じシステムで二人で違う動作を行うという探索的テストを行っていた	同上	対象
1-17	複数のエラーが同時に発生した場合に複数のエラーメッセージが表示されないかを確認していた	ユーザの気持ちになってみた(エラーが複数出るのはユーザにとってまどろっこしいと感じた)	対象
1-18	普段のテストではやらないカメラの映り方にしてみた	ユーザの気持ちになってみた(カメラにアップで映るのが恥ずかしいと感じた)	対象
1-19	直接入力できないところはコピー&ペーストで入力した	上司から言われて意識した	対象
1-20	”&”を入力した	同上	対象
1-21	テスト設計者が書ききれていないと考えられることを実施した	同上	対象
1-22	競合後の動作確認をした	同上	対象
1-23	ショートカットキーによる操作をした	同上	対象
1-24	「未入力」にした	同上	対象
1-25	0 値を入力した	同上	対象
1-26	無効値を入力した	同上	対象
1-27	テスト開始の前に、機能の確認を行った	テストに必要な準備のために行った	対象外
1-28	テスト用のデータを作成していた(たまたま不具合が起こる条件に該当するデータだった)	同上	対象外
1-29	繰り返しのテスト中に終了操作を行った	繰り返し操作を行うテストの中で、やらなければいけない操作だった	対象外
1-30	設定変更を繰り返していた	同上	対象外
1-31	たまたま↓キーを押した	特に理由はなく実施した	対象外
1-32	たまたま不具合の起こる設定にしていた	同上	対象外
1-33	たまたま放置していた	同上	対象外
1-34	時間のかかるテストの時に、テストとは関係のない画面を見ていた	同上	対象外
1-35	たまたま不具合の起こるデータを使っていた	同上	対象外
1-36	たまたまある状態(ある画面である処理が実施されている状態)中に、違うテストをしていた	同上	対象外

手順 2 では ID1-27～1-36 を、欠陥検出を狙った操作ではないと判断し、分析の対象外とした。ID1-27, 1-28 はテストに必要な準備のために実施されたものであり、欠陥

表 2 欠陥のトリガーと欠陥検出時の操作

ID	欠陥のトリガー	欠陥検出時の操作
2-1	想定外の操作を受け付けた	通常では押せないボタンを他の方法で押せるようにした (ID1-1), ショートカットキーによる操作をした (ID1-23)
2-2	想定外の入力値, 入力データを受け付けた	いろいろな値にしてみた (ID1-6), ” & ” を入力した (ID1-20), 「未入力」にした (ID1-24), 0 値を入力した (ID1-25), 無効値を入力した (ID1-26), 直接入力できないところはコピー & ペーストで入力した (ID1-19)
2-3	データの不整合, 競合が発生した	二重起動を試してみた (ID1-8), 重複登録を行ってみた (ID1-10), 同じシステムで二人で違う動作を行うという探索的テストを行っていた (ID1-16), 画面の移り変わりを素早くしてみた (ID1-12)
2-4	高負荷, リソース不足に陥った	怪しそうな負荷を与えてみた (ID1-11)
2-5	処理の進行に伴う状態の変化やデータの蓄積が起こった	テストケース実施終了後の動作も気になってやってみた (ID1-3), 通信のやり取りを見ていた (ID1-2), 競合後の動作確認をした (ID1-22)
2-6	使用頻度が低い機能や設定がテストされていない	ボタンを押してみた (ID1-5), プルダウンを押してみた (ID1-7), 設定を変更してみた (ID1-9), 見てない画面がないかを確認してみた (ID1-13)
2-7	仕様通りであってもユーザにとって望ましくない動作をする	複数のエラーが同時に発生した場合に複数のエラーメッセージが表示されないかを確認していた (ID1-17), 普段のテストではやらないカメラの映り方してみた (ID1-18)

検出を狙った操作ではない。ID1-29, 1-30 はテストケースで指定された操作を実施した結果, 偶然欠陥の検出に繋がったものであり, QA エンジニア自身が欠陥検出を狙って行ったものではない。ID1-31～1-36 はテストの実施中に偶然実施した操作や, 偶然使用した設定やデータによって欠陥が検出されたものであり, いずれも意図的な欠陥検出のための操作ではない。

表 1 に示した操作の ID1-1～1-26 を手順 3 でグルーピングした結果, 表 2 に示す 7 個の欠陥のトリガーを定義した。ID1-4, 1-14, 1-15, 1-21 の 4 つの操作は特定の欠陥のトリガーを狙ったものではないと判断し, 欠陥のトリガーの定義には利用しなかった。

手順 4 で定義した欠陥のトリガーを起こす方法の概要を表 3 に, 具体的な方法を表 4～表 10 に示す。表 4～表 10 の「欠陥のトリガーを起こす方法の具体例」は, 表 1 の欠陥検出時の操作の中で該当するものに加え, 筆者らの議論で挙げたものを記載した。

表 3 欠陥のトリガーを起こす方法の概要

ID	欠陥のトリガー	欠陥のトリガーを起こす方法の概要
3-1	想定外の操作を受け付けた	制限がかかっているところに対して, 他の方法で操作を試みる
3-2	想定外の入力値, 入力データを受け付けた	通常では想定しにくい異常値や本来入らないはずの値を入れる
3-3	データの不整合, 競合が発生した	同時に同じデータやリソースにアクセスしたり, 同じデータに対して複数カ所から更新をかけたりする
3-4	高負荷, リソース不足に陥った	負荷がかかるようなデータを使ったり, いろいろな処理を同時に動かしたりする
3-5	処理の進行に伴う状態の変化やデータの蓄積が起こった	処理の進行中に止めたり, 処理が終わった後まで確認したりする
3-6	使用頻度が低い機能や設定がテストされていない	実装やテストを行う際に使用頻度が低い機能や設定を使う
3-7	仕様通りであってもユーザにとって望ましくない動作をする	ユーザが利用する状況を想定して, 利用した際の気持ちや想像しながらテストする

表 4 「想定外の操作を受け付けた」(ID 3-1)を起こす具体的な方法

ID	欠陥のトリガーを起こす方法の具体例	具体的な方法
4-1	通常では押せないボタンやメニューを他の方法で押そうとする	<ol style="list-style-type: none"> 仕様や実際のテスト対象の画面から、押せないボタンやメニュー(特定の条件で非活性や非表示になるボタンやメニュー, ダイアログなどの裏側に隠れて押せないボタンやメニュー)を探す 該当のボタンやメニューが押せない状態を作ってから、以下のような通常と異なる方法で押すことを試みる <ul style="list-style-type: none"> マウスクリックでなくキー操作で押す, Tab キーでフォーカス移動する ショートカットキーで、ボタンやメニュー押下時と同等の操作をする
4-2	想定外と思われるイベントを起こす	<ol style="list-style-type: none"> ソフトウェアの状態と発生し得るイベントを識別する 各状態で、通常の利用では発生しないイベントを起こす。ボタン押下やケーブル抜き差しなどの物理的なイベントは多くの状態で起こすことができることも考慮する。

表 5 「想定外の入力値, 入力データを受け付けた」(ID 3-2)を起こす具体的な方法

ID	欠陥のトリガーを起こす方法の具体例	具体的な方法
5-1	通常では想定しにくい異常値を入力する	<ol style="list-style-type: none"> 数字や文字やデータを入力できる欄を探す 以下のような値を入力する <ul style="list-style-type: none"> 0 値, 無効値を入力する ”&”などシステムにとって特別な意味を持つ記号を入力 古い形式のデータを入力する 未入力にする <p>■変数の型に対する異常値を入力する</p> <ul style="list-style-type: none"> 数値の場合 → 全角文字, 負の値, 小数, とても大きい(または小さい)値 日付の場合 → 形式違い, 全角文字, 日付として存在しない数値, 過去や未来 文字の場合 → HTML や JS や SQL などのコード, システムにとって特別な意味を持つ記号, 文字数が多い値 <p>■変数の意味を考慮した異常値を入力する</p> <ul style="list-style-type: none"> ファイルやフォルダ名 → すでに存在するのと同じ名前(大文字・小文字の違い含む), パスの区切り文字を含む値 ID・パスワード → 存在する ID(大文字・小文字の違い含む)
5-2	直接入力できないところはコピー&ペーストで入力する	<ol style="list-style-type: none"> 仕様や実際のテスト対象の画面から、入力できない入力欄(特定の条件で非活性や非表示になる入力欄)を探す 該当の入力欄が入力できない状態を作ってから、以下のような通常とは異なる方法で入力することを試みる <ul style="list-style-type: none"> 他のところから値をコピー&ペーストして入力

表 6 「データの不整合, 競合が発生した」(ID 3-3)を起こす具体的な方法

ID	欠陥のトリガーを起こす方法の具体例	具体的な方法
6-1	二重起動を行う	<p>1. PC やモバイルのアプリケーションであれば二重起動する。ログインして利用するアプリケーションであれば、複数端末で起動する。</p> <p>Web アプリケーションであれば、二重(複数タブ, 複数ウインドウ, 複数ブラウザ, 複数端末など)でアクセスする</p> <p>2. 二重起動した各画面で、同一のデータやリソースに対して同じ処理を行ったり、矛盾する処理を起こしたりする(同一データを各画面で削除する, 片方でのみ削除してもう片方はデータが残っている前提で他の操作をする, など)</p>
6-2	データを重複登録する	<p>1. データを登録する機能を探す</p> <p>2. 通常は 1 個しか登録しないデータを複数登録したり、同一と判定されるようなデータを登録したりする(同じ名前のデータ, 同じ内容のデータ, 同じ ID のデータ)</p>
6-3	画面の移り変わりを素早くする	<p>1. テスト対象を動作させ、画面遷移に時間がかかっている部分(画面が開ききる前に、他の操作を行える時間的な余裕がある部分)を探す</p> <p>2. その画面が開ききる前に、画面を閉じたり別の画面に遷移する操作を行ったり、またすぐにその画面に戻ったりする</p>
6-4	同じシステムで二人で違う動作を行う	<ul style="list-style-type: none"> データの更新が起こる部分を探し、二人で異なる動作を行う(片方はデータを編集, 片方はそのデータを削除など) データに影響を与える設定を探し、二人で異なる設定を行いテストする(片方は設定 ON・片方は設定 OFF など)

表 7 「高負荷, リソース不足に陥った」(ID 3-4)を起こす具体的な方法

ID	欠陥のトリガーを起こす方法の具体例	具体的な方法
7-1	怪しそうな負荷を与える	<ul style="list-style-type: none"> 処理量を数値で指定する部分では、大きい値にする(印刷枚数の設定など) 文字を入力する部分では、長めの文字を入力したり、入力する量を多くしたりする ファイルを指定する部分では、ファイルを大量に指定したり、サイズが大きいファイルを指定したりする データを作る部分では、データを大量に作ったり、サイズが大きいデータを作ったりする 他の機能や裏で動く処理があれば同時に動かすことで、メモリや CPU やネットワークに負荷をかける

表 8 「処理の進行に伴う状態の変化やデータの蓄積が起こった」(ID 3-5)を起こす具体的な方法

ID	欠陥のトリガーを起こす方法の具体例	具体的な方法
8-1	通信のやり取りを見ながら操作する	<p>通信のやり取りを見て、やり取りの途中でネットワークを切ったり処理をキャンセルしたりするなどの割り込みを起こす(「処理中」「通信中」という一つの状態を、処理内容ごとに分けて異なる状態と捉えてテストする)</p>
8-2	テストケース実施終了後の動作も続けて確認する	<p>テストケースを最後まで実施した後、そこで作ったデータや設定を使う機能を探して、動かす(録画して録画データが保存されたことを確認したら、その録画データを再生したり外部に書き出したりするなど)</p>
8-3	競合後の動作確認をする	<p>同じデータへの操作の競合が発生する部分を探し、複数名で同じデータに対して操作を行うなど競合を起こしてから、そのデータを使った操作を行う</p>

表 9 「使用頻度が低い機能や設定がテストされていない」(ID 3-6)を起こす具体的な方法

ID	欠陥のトリガーを起こす方法の具体例	具体的な方法
9-1	使用頻度の低い要素を触る	・テストの実施中に触ったことがない要素(ボタンやプルダウンなど)や触る頻度が低い要素があるか、仕様やマニュアルと照らして探し、該当の要素を触る
9-2	使用頻度の低い設定にする	・テストの実施中に変更したことがない設定や変更する頻度が低い設定があるか、仕様やマニュアルと照らして探し、該当の設定を使ったり変更したりする
9-3	見えない画面がないか確認する	・テストの実施中に表示していない画面や表示する頻度が低い画面があるか、仕様やマニュアルと照らし合わせて確認し、該当の画面を表示する

表 10 「仕様通りであってもユーザにとって望ましくない動作をする」(ID 3-7)を起こす具体的な方法

ID	欠陥のトリガーを起こす方法の具体例	具体的な方法
10-1	ユーザが利用する際の気持ち进行を想像する	<ul style="list-style-type: none"> ・エラーが表示される部分を探し、表示されるエラーが適切か確認する(表示内容、表示回数・頻度など) ・ユーザが置かれた状況を想像した上で、ユーザがどのような行動を取るか、テスト対象の挙動をどう感じるかを想像する ・マニュアルに沿ってテスト対象を動かし、ユーザがそのシステムを使う目的が達成されるか確認する

5. 考察

5.1. 欠陥のトリガーの意義・活用・拡張

本節では、QA エンジニアの欠陥検出時の操作を欠陥のトリガーとそれを起こす方法という枠組みで分析したことによって得られた知見について考察する。具体的には、①欠陥のトリガーの意義、②実務での活用の可能性、③今後の拡張と知見収集への応用という3つの観点から、欠陥のトリガーを定義したことの意義と活用や拡張の可能性について議論する。

5.1.1. 欠陥のトリガーの意義

本研究では QA エンジニアが欠陥検出時に行った操作を分析し、欠陥のトリガーとそれを起こす方法を明らかにした。これにより、欠陥検出時の操作が何を狙ったものを明示的に理解できるようになった。表 4~10 に示したように欠陥のトリガーを起こす具体的な方法を定義したことで、実務や教育において再現性のある知識として利用できる。

経験の少ない QA エンジニアだと十分なエラー推測ができない可能性がある。これに対し本研究では、欠陥が発生しやすい典型的な欠陥のトリガーを整理し、それを

起こす具体的な方法を提示したことで、経験の浅い QA エンジニアでも欠陥を検出しやすくなることが期待される。

複数名の QA エンジニアへのインタビュー調査に基づき多様な視点の欠陥のトリガーを導出したことで、幅広い視点での欠陥検出に活用可能な知識を構築することができた。本研究で整理した欠陥のトリガーは、「想定外の操作」や「データの不整合」といった開発技術の視点に加え、「使用頻度が低い機能や設定がテストされていない」といったテストの視点、「仕様通りであってもユーザにとって望ましくない動作をする」といったユーザの視点も考慮したものとなっており、幅広い視点で欠陥検出が可能となる。

5.1.2. 実務での活用の可能性

本研究で定義した欠陥のトリガーとそれを起こす方法は、テストケースの設計や探索的テストの実施に活用できる。テストケースの設計では、仕様や要件を基にテストの内容を検討する際の着眼点として活用できる。探索的テストでは事前にテストケースを定義せずに、QA エンジニアがテスト実行とテスト内容の検討を並行して行う。探索的テストではテストチャーターと呼ばれるテストの目的を記述したドキュメントをあらかじめ用意することがある。本研究で整理した欠陥のトリガーをテストチャーターの素

材として活用することで、着眼点の漏れを防ぎ、より効果的な欠陥検出が可能になると考えられる。探索的テストの実践的な進め方として、旅行者によるツアーになぞらえてテストする方法が知られている[10]。例えば「知的ツアー」はソフトウェアに対してできるだけ厳しい条件を選んでテストを進めるものであり、そのようなツアーにおけるテスト内容の検討時に、本研究で定義した欠陥のトリガーとそれを起こす具体的な方法は活用可能と考えられる。欠陥検出の方法を具体的に定義したことで、これらの場面に加えて QA エンジニアの教育にも活用できる可能性がある。

QA エンジニアが持つ知識に応じて、狙いやすい欠陥のトリガーは異なると考えられる。QA エンジニアの知識と狙いやすい欠陥のトリガーの関係を表 11 に整理した。例えば、ユーザの操作や入力値をソフトウェアがどのように受け取って処理を行っているかという、実装に関する知識があれば、「想定外の操作を受け付けた」や「想定外の入力値、入力データを受け付けた」という欠陥のトリガーを狙いやすい。テストの実施状況に関する知識があれば使用頻度が低い機能を把握でき、ユーザに関する知識があればユーザにとって望ましくない動作に気づきやすい。QA エンジニアが持つ知識に応じて狙う欠陥のトリガーを定めることで、効果的な欠陥検出が可能になると考えられる。

5.1.3. 今後の拡張と知見収集への応用

欠陥のトリガーごとに具体的な方法を定義したことで、QA エンジニアへのインタビューでは得られなかった方法を充実させることができた。例えば、以下の具体的な方法は表 1 のインタビュー結果には存在しなかったが、筆者ら

表 11 QA エンジニアの知識と狙いやすい欠陥のトリガー

ID	知識	狙いやすい欠陥のトリガー
11-1	実装に関する知識	<ul style="list-style-type: none"> ・想定外の操作を受け付けた ・想定外の入力値、入力データを受け付けた ・データの不整合、競合が発生した ・高負荷、リソース不足に陥った ・処理の進行に伴う状態の変化やデータの蓄積が起こった
11-2	テストの実施状況に関する知識	<ul style="list-style-type: none"> ・使用頻度が低い機能や設定がテストされていない
11-3	ユーザに関する知識	<ul style="list-style-type: none"> ・仕様通りであってもユーザにとって望ましくない動作をする

の経験から導出できたものである。

- 表 4 の ID4-2:「1.ソフトウェアの状態と発生し得るイベントを識別する 2.各状態で、通常の利用では発生しないイベントを起こす」
- 表 5 の ID5-1:「変数の意味を考慮した異常値を入力する」
- 表 6 の ID6-4:「データの更新が起こる部分を探し、二人で異なる動作を行う」、「データに影響を与える設定を探し、二人で異なる設定を行いテストする」

欠陥のトリガーとそれを起こす方法を分けて整理することで、欠陥検出の知見を体系的に整理でき拡充しやすくなると考えられる。

今後さらなるインタビューの実施により、欠陥のトリガーとそれを起こす方法を充実させることで、経験が少ない QA エンジニアや開発者でもより多くの欠陥の検出が可能になる。なるべく少ないテストケースで欠陥を検出できる熟練の QA エンジニアは、欠陥のトリガーとそれを起こす具体的な方法のバリエーションを暗黙的に把握していると考えられる。そのような熟練者へのインタビューを通じて欠陥のトリガーとそれを起こす方法を拡充すれば、より有用な知見となる。

今後さらに QA エンジニアにインタビューする際、本研究で定義した欠陥のトリガーを提示することで効率よくインタビューできる可能性がある。インタビューの際に、操作だけでなく狙っている欠陥のトリガーを聞き出すようにしたり、定義済みの欠陥のトリガーを提示して他の欠陥のトリガーを想起してもらったりすることで、欠陥のトリガーとそれを起こすための方法を効率的かつ構造的に収集することができると考えられる。

5.2. 研究の限界と今後の課題

本研究のインタビュー対象者は経験年数が 1～3 年の QA エンジニアであるため、熟練者の欠陥検出方法は十分に調査できていない。インタビュー結果の中には上司の指示により行っていた操作も含まれるため、熟練者の欠陥検出方法も部分的に含まれていると考えられる。しかし、熟練者の知見も含めて体系的に QA エンジニアの欠陥検出方法を明らかにするには、今後の追加調査が必要である。

本研究で明らかにした欠陥のトリガーとそれを起こす方法が、ソフトウェア開発プロジェクトにおける欠陥検出にどの程度役に立つかを検証することは今後の課題である。本研究で定義した 7 つの欠陥のトリガーは、ある商用製品の開発プロジェクトで得られたデータから定義したものであり、他のプロジェクトでは異なる欠陥のトリガーが存

在する可能性がある。今後、異なる種類のソフトウェアを対象に同様のインタビューを実施し、本研究で定義した欠陥のトリガーの汎用性の評価や、本研究に含まれなかった欠陥のトリガーを明らかにすることが必要である。

6. おわりに

本研究では QA エンジニアへのインタビューにより欠陥検出時の操作を収集し、それらを分析することで、欠陥のトリガーとそれを起こす具体的な方法を整理した。インタビューの結果から得られた 26 個の欠陥検出時の操作を分析し、「想定外の操作を受け付けた」、「想定外の入力値、入力データを受け付けた」、「データの不整合、競合が発生した」などの 7 個の欠陥のトリガーを明らかにした。筆者らの議論により欠陥のトリガーを起こす具体的な方法を定義した。

本研究で定義した欠陥のトリガーとそれを起こす方法は、テストケースの設計や探索的テストの実施に活用できる。7 個の欠陥のトリガーは開発技術の視点、テストの視点、ユーザの視点を考慮したものとなっており、これらを活用することで開発者や経験の浅い QA エンジニアでも幅広い視点で欠陥を検出できる可能性がある。

今後の課題として、定義した欠陥のトリガーとそれを起こす方法がソフトウェア開発プロジェクトにおける欠陥検出にどの程度役に立つかを検証することや、欠陥のトリガーの汎用性を評価することが必要である。熟練の QA エンジニアや異なる種類のソフトウェアを対象に同様のインタビューを実施することで、本研究で定義した欠陥のトリガーの汎用性を評価するとともに、本研究に含まれなかった欠陥のトリガーを明らかにしていく。

参考文献

- [1] International Software Testing Qualifications Board. *Certified Tester Foundation Level Syllabus v4.0.1*. Belgium, 2024.
- [2] Tanizaki, K., Hiruta, Y., Soma, T., Yamao, N., Kato, S., & Iizuka, Y. "A Method for Software Test Design Considering Weakness and Adverse Condition." *Total Quality Science*, 7(3), pp.173-189, 2021.
- [3] 河野 哲也, 西 康晴. “ソフトウェア開発における外部機能の仕様記述の曖昧さに着目したテスト項目設計法”, *品質*, 42(3), pp.118-129, 2012.
- [4] 河野 哲也, 柏倉 直樹, 前川 健二, 菊武 祐輔. “Web サービス・モバイルアプリケーション開発におけるテスト設計を支援するための標準テスト観点の整備”, *ソフトウェア・シンポジウム 2019 in 熊本*, pp.162-171, 2019.
- [5] Deak, Anca. "What characterizes a good software tester?—a survey in four Norwegian companies." *Testing Software and Systems: 26th IFIP WG 6.1 International Conference, ICTSS 2014, Madrid, Spain, September 23-25, 2014*. Proceedings 26. Springer Berlin Heidelberg.
- [6] Florea, Raluca Madalina. "The Software Tester: An Exploration of the Skills and Practice of the Role." 2023.
- [7] Basili, V. R., & Selby, R. W. “Comparing the effectiveness of software testing strategies.” *IEEE transactions on software engineering*, (12), pp.1278-1296, 1987.
- [8] Iivonen, J., Mäntylä, M. V., & Itkonen, J. (2010, September). Characteristics of high performing testers: a case study. In *Proceedings of the 2010 ACM-IEEE international symposium on empirical software engineering and measurement*, pp. 1-1.
- [9] ISTQB Glossary, V4.6.1, https://glossary.istqb.org/ja_JP/home (参照日: 2025年5月23日).
- [10] James A. Whittaker, *Exploratory Software Testing: Tips, Tricks, Tours, and Techniques to Guide Test Design*, Addison-Wesley, 2009.