QC 七つ道具を利用した DevOps プラクティスの導入による開発とテストの生産性改善

Development and Testing Productivity Improvement through DevOps Practice Implementation and 7 tools of QC

楽天株式会社　エコシステムサービス部
デリバリーアンドクオリティグループ
内藤　史郎 1)　黎 園園 2)　○荻野 恒太郎 3)
Shiro Naito 1)　YuanYuan Li2)　○Kotaro Ogino 3)

Abstract　Productivity improvement of development and testing, through the introduction of right Agile and DevOps practices for resolving root causes in actual project issues, is a key factor for success of software development. In this empirical paper, we report a case study of productivity improvement. Firstly, we conducted a root cause analysis using 7 tools of QC followed by introducing DevOps practices, including pipeline and test automation. As a result, the incident on integration testing environment was reduced by 66% and bug fixing days were reduced from 10 to 2 days.

1.　はじめに

生産性改善のためアジャイルや自動化、DevOps などのソフトウェア開発プラクティスを導入するチームが増えている[1]。特にスクラムはトヨタ生産方式（TPS）[2]や QC サークル[3]を起源に持ち[4][5]、そのプラクティスの日次ミーティング、ふりかえり、イテレーション計画ミーティング、イテレーションは、日本でのプラクティス適用率も高い[1]。また近年では、自動化やクラウドの普及と共に、品質保証や運用チームにもアジャイル関連のプラクティスの適用範囲が広がっており[6][7][8]、それらの活動は DevOps として知られている[9][10][11]。

アジャイルや DevOps を導入するチームが増えている一方で、導入にはアジャイル開発ならではの問題点もある。アジャイル開発は、ウォーターフォールの品質保証の考えとの親和性が低く、チームの条件に合わせて判断することやトレーニングの必要性が指摘されている[12]。また、アジャイルのような海外で体系化されたモデルではなく、日本の経験にもとづいたソフトウェアプロセス改善のアプローチの重要性も指摘されている[13]。さらに、少しずつ素早い開発を繰り返す DevOps では継続的なプロセスの改善が重要となる一方、やみくもな自動化をしてしまうとプロセスの生産性がかえって低下する可能性もある。

本経験論文では、QC 七つ道具を活用し DevOps プラクティスをプロジェクトに導入した事例を紹介する。対象プロジェクトの生産性の問題の根本原因を QC 七つ道具によって分析し、クライアントの統合テスト環境での障害に関するプロセス中のムダを特定した。それらのムダを解決するため DevOps プラクティスをプロジェクトに導入し、開発とテストの生産性を改善した。

楽天株式会社 エコシステムサービス部
Ecosystem Services Department, Rakuten, Inc.

東京都世田谷区玉川 1-4-1　Tel: 050-5581-6910　e-mail:shiro.naito@rakuten.com
1-4-1, Tamagawa, Setagaya, Tokyo Japan
1)楽天株式会社 エコシステムサービス部 デリバリーアンドクオリティグループ　ヴァイスマネージャー
Vice Manager, Delivery and Quality Group, Ecosystem Services Department, Rakuten Inc
2)楽天株式会社 エコシステムサービス部 デリバリーアンドクオリティグループ　アシスタントマネージャー
Assistant Manager, Delivery and Quality Group, Ecosystem Services Department, Rakuten Inc
3)楽天株式会社 エコシステムサービス部 アイデンティティプラットフォームグループ　マネージャー
Manager, Identity Platform Group, Ecosystem Services Department, Rakuten Inc

【キーワード：】DevOps、QC 七つ道具、アジャイル、テスト自動化、パイプライン、メトリクス

2. 対象のプロジェクト

2.1 プロジェクトの背景

本経験論文の対象プロジェクトのプロダクトは内製のウェブサービスであり、開発、テスト、運用のすべてのプロセスが社内で完結している。このウェブサービスはクライアントを通しエンドユーザーに利用されており、いったんこのシステムが停止するとすべてのビジネスが利用できなくなるため影響範囲が大きく、高い信頼性と可用性が求められる。

2.2 開発から本番運用までのプロセス

図.1 に開発から本番運用までのプロセスを示す。このプロジェクトには開発、テストと DevOps チームが含まれており、それぞれ 10～ 20 名ほどの規模である。開発チームはプログラムの実装後、開発環境でデバッグおよびテストを行う。次に変更は QA 環境にデプロイされ、テストチームによってテストされる。次に変更はクライアント統合テスト環境にデプロイされ、クライアントとの統合テストを実施する。すべてのテストに成功したら、本番環境へとリリースされる。

これらの開発からリリースまでのプロセスは独立しておらず、「ブレンド」して最短で 1 日で実施される。つまり XP モデル[12][14]に相当する。XP モデルの場合、プロセスがブレンドして実施されるため、ウォーターフォールモデルに比べプロセス間の影響に対し敏感である。

2.3 対象プロジェクトの問題

対象プロジェクトでは、クライアントの統合テスト実施環境で、週に中央値で 3 件の障害が発生するという問題が発生していた。クライアント統合テスト環境でのバグが増えると、対象の機能開発で手戻りが発生する上に、並行して進められているその他の機能開発にも遅延等の影響が出る。テストチームがテストを実施する QA 環境でもバグが見つかっており、結果として QA 環境とクライアント統合テスト環境では 1 週間の中央値で合計 60 件のバグが見つかっていた。

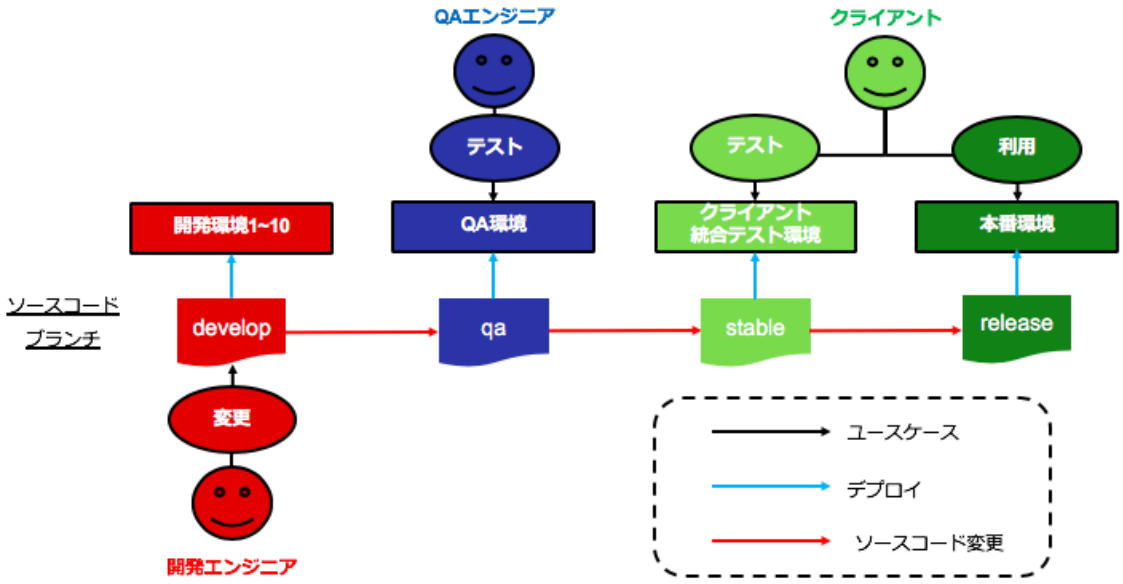

図 1 開発から本番運用までのプロセス

3. QC 七つ道具を使用した根本原因分析

クライアント統合テスト環境での障害の根本原因を特定するため、QC 七つ道具[15]を使用してプロセスの品質メトリクスを分析した。まず、障害の根本原因が「開発」、「テスト」と「開発とテスト」の各プロセスのどこにあるか特性要因図を使って仮説を立てた。次にヒストグラムや散布図などの QC 七つ道具を使用し、それらの仮説が正しいか検証した。

根本原因分析で QC 七つ道具を使った理由は、対象プロジェクトのプロセスが開発・テスト・リリースと定型的な反復開発だからである。定型的な反復開発の場合、品質メトリクスが時系列で手に入るため、ヒストグラムや散布図などの QC 七つ道具をより有効に使用できる。時系列で品質メトリクスが手に入る点は、定型的な製造ラインがあるトヨタ生産方式(TPS)[2]とも類似している。

3.1　特性要因図による根本原因分析

特性要因図[15]を用いて問題の根本原因を分析した。プロセスやチーム間での受け渡しが発生するタスクに根本原因があると仮定し、1.開発の生産性、2.テストの生産性、3.開発とテストの生産性を要因として設定した。図.2 に特性要因図を示す。

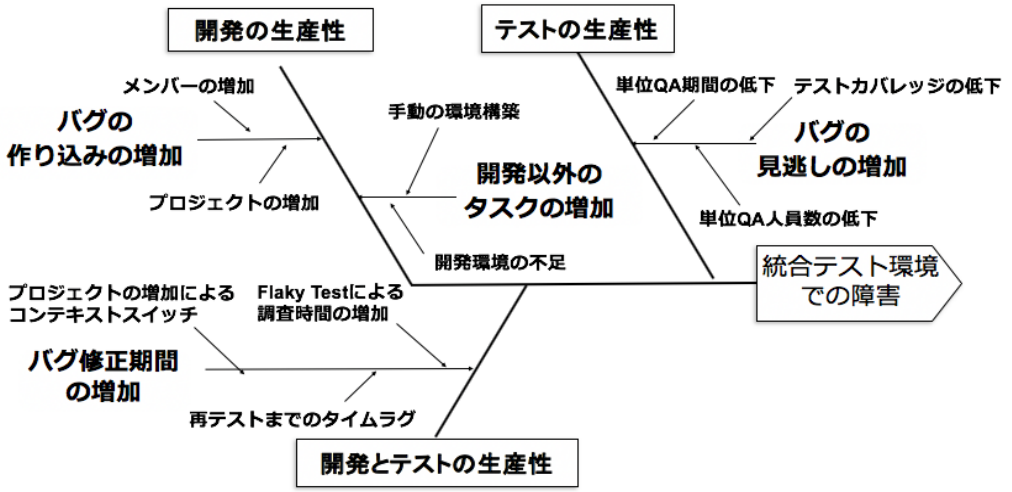

図 2 特性要因図による生産性分析

開発の生産性の問題の原因は開発のバグの作り込み、つまり不良のムダの増加である。プロジェクト数や新規メンバーの数が増えると、作り込まれるバグも単純に増えることが想定される。また、手動での開発環境の構築に数時間かかったり、開発環境が不足したりすると、QA 環境にソースコードをマージする前に十分なデバッグを開発者は実施できない。

テストの生産性の問題の原因は、テストでのバグの見逃しの増加である。テストカバレッジが不十分だと、バグの見逃しの数が増加する。加えて、単位 QA 期間や単位 QA 人員数が低下するとテスト文書の作成をしなくなるなど、テスト自体の品質にムラが発生することがある。

開発とテストの生産性の問題の原因はバグ修正期間、つまり開発・テストチーム間の手待ちのムダの増加である。テスト実行のたびに結果が変わる Flaky Test は、テスト失敗の原因調査の時間を増加させる。さらに、確認テストを実行する際のテスト環境の再セットアップなども、バグ修正期間に影響を与える。

2.1 のとおり、XP モデルでは各プロセスの問題が周囲のプロセスに対し敏感に影響する。たとえば、開発プロセスでのバグの増加はテストプロセスで発見されるバグの増加を引き起こす。テストプロセスでのバグの増加は、バグ修正開始までの手待ちのムダの増加につながる。

3.2　ヒストグラムによる開発の生産性の分析

前項で述べた開発の生産性の問題を検証するため、QC 七つ道具によって分析した。分析結果では、特に開発環境の不足の問題が顕著に示された。過去 6 ヶ月間における週次の開発・テスト環境の予約数の推移を図.3 に、また、開発・テスト環境予約数の頻度を図.4 のヒストグラムに示す。

図 3 時系列での開発・テスト環境の予約件数

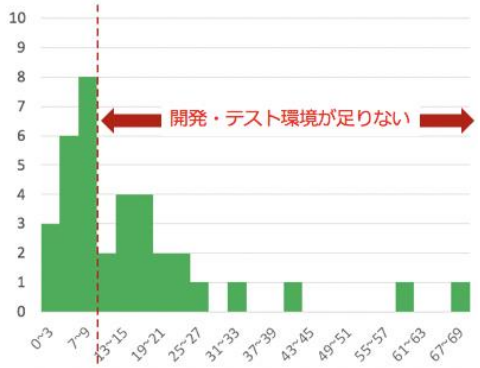

図 4 開発・テスト環境の予約件数のヒストグラム

対象プロジェクトでは、実在する開発・テスト環境数 10 を上回る環境の予約件数があり、開発・テスト環境数が不十分な場合があった。開発・テスト環境が不足すると、開発者が環境を利用できるようになるまで手待ちのムダが発生する。最大予約件数は 69 件で、予約件数のヒストグラムは右スソ引き型[15]を示している。予約件数の分布はばらつきが大きいため、単純に開発・テスト環境数を増やすのではなく、プロジェクトの状況に応じて柔軟に数を変更できる仕組みを導入する必要があることを開発・DevOps チーム間で合意した。

3.3 散布図によるテストの生産性の分析

テストの生産性に係る原因であるテストカバレッジの低下を検証した。QA 環境で発見された不具合とクライアント統合テスト環境で発生した障害数の関係を図.5 の散布図に示す。QA 環境での再帰テストでの検出バグ数が増えると統合テスト環境での障害件数も増えており、正の相関関係を示している。本来十分なテストカバレッジがあれば、負の相関関係になるはずである。これにより、QA 環境での再帰テストのカバレッジが不足していることがわかった。

また、過去 6 ヶ月間の QA 環境でのバグ検出率の推移を図.6 に示す。バグ検出率は低下の傾向を示している。これらの分析結果より、テストカバレッジの不足が原因で QA 環境での再帰テストでは品質のムラを抑えきれていないことがわかった。

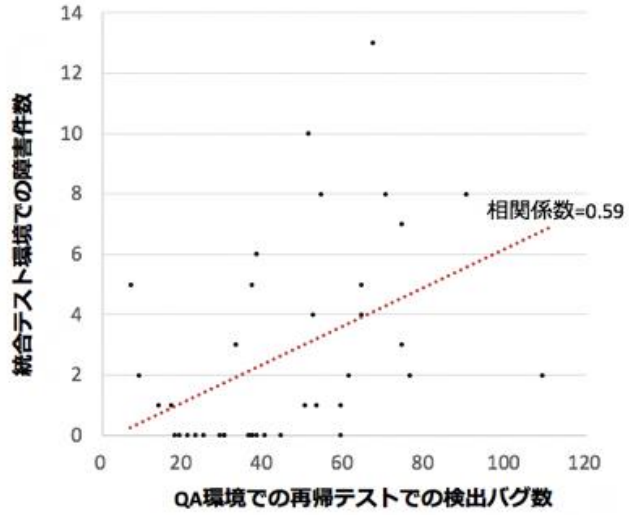

図 5 検出バグ数と障害件数の散布図

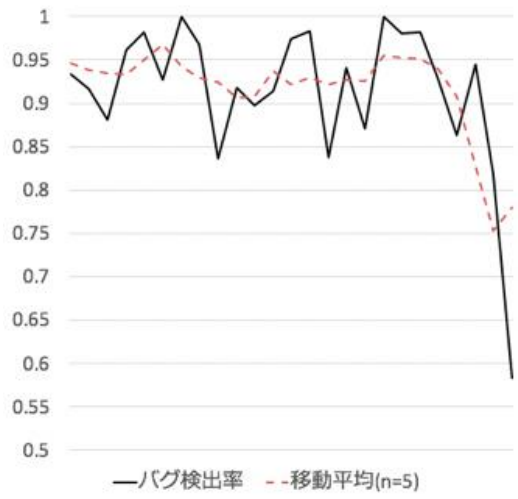

図 6 バグ検出率の推移

3.4 箱ひげ図による開発とテスト間の生産性の分析

開発とテストの生産性の問題の原因を検証するために、QA 環境でバグが検出され修正されるまでにかかる時間を図.7 の箱ひげ図に示す。QA 環境で発見されたバグのうち、25%程度は 2 日以内に修正されているが中央値では 10 日かかっており、手待ち時間が長かった。ときに 24 日以上かかることもあり、手待ちのムダが適切に対処されていない認識を開発・テストチーム間で共有した。

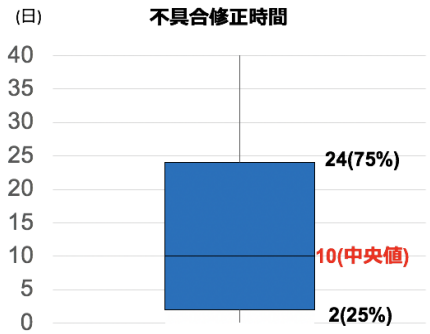

図 7 不具合修正時間

4. 解決方法

4.1 解決方法概要

QC 七つ道具で発見した根本原因を解決するため、DevOps のプラクティスを導入した。表.1 に根本原因と対策を示す。パイプラインの自動化と、受け入れテストの自動化により、図.8 に示すように開発からリリースまでの一連のプロセスを自動化した。

DevOps の技術的な側面である継続的デリバリー[9][10]の重要なプラクティスであるパイプラインの自動化では、ソフトウェアをバージョン管理ツールから取り出しリリースするまでのプロセスを自動化する[11]。この自動化により、ソフトウェアを変更するたびに必要だった環境構築やデプロイなどの煩雑なプロセスが、素早く、繰り返し実行可能なものになる[11]。

受け入れテストの自動化は継続的デリバリーのもう一つの重要なプラクティスである[11]。受け入れテストを自動化することでソースコード変更の直後に回帰テストを実行し、問題があった場合それを開発チームに迅速にフィードバックすることができる[11]。これは TPS のニンベンのある自働化 [2] と共通した考え方である。もしパイプラインだけを自動化すると、予期しない問題の発見が遅れ、プロセス全体の生産性が下がる恐れがある。

表 1 QC 七つ道具で発見された根本原因と対策

根本原因	DevOps プラクティス	対策
1. 開発/テスト環境が不十分	パイプライン自動化	スケーラブルな開発/QA 環境
2. 回帰テストのカバレッジが不十分	受け入れテスト自動化	テストカバレッジ向上
3. DEV/QA 間の手待ち時間が長い	受け入れテスト自動化	自動テストの日次実行

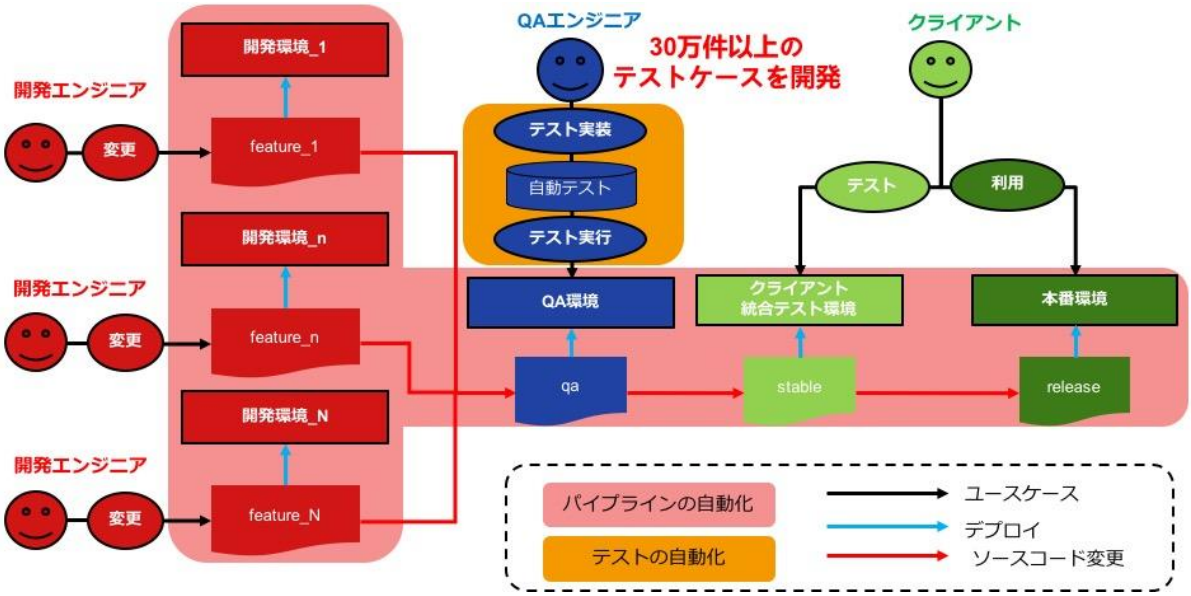

図 8 改善後の開発から本番運用までのプロセス

4.2 パイプラインの自動化（スケーラブルな開発/QA 環境）

開発とテスト環境が不十分な問題を解決するため、まず Infrastructure as Code[16]を実践した。環境構築手順を Docker[17]で自動化し、数分で新たな環境を構築できるようにした。次に Kubernetes[18]を使い環境差分の設定を切り分けることで、保守性を担保しつつ、必要に応じて環境をスケーラブルに用意できるようにした。また、Jenkins [19] と統合し開発環境から本番環境までのプロセスを自動化することで、このプロセスを素早く、繰り返し実行可能なものにした。

パイプラインを自動化するとソフトウェアと環境のバージョンを一元的に管理できるようになり、バージョン間の差分で問題が発生した場合の調査時間を軽減できるメリットもある。また、パイプラインを自動化すると開発チームの誰でも環境を構築できるようになり、属人性を排除できる。

4.3 受け入れテストの自動化

(1) テストカバレッジ向上

回帰テストのカバレッジが不十分である問題を解決するため、End-To-End のシナリオ 30 万件のテストケースを受け入れテストとして実装した。画面テスト、API の機能テスト、入力値のチェック等のテストが含まれる。受け入れテストを自動化すると、ソースコードを変更した後すぐに回帰テストを実行できるため、ソースコードの不良のムダを迅速にフィードバックできる。

３０万件と大量のテストを簡便に実装、保守するために、Selenium[20]と Cucumber[21]を使ってテスト自動化フレームワークを開発した。振る舞い駆動開発（BDD）[22]で記述されたテストスクリプトを、テスト観点ごとに管理する。データ駆動テストを採用し、複数のテストシナリオで共通のテストステップを、異なるデータタイプのテストで再利用できるようにした。

(2) 自動テストの定期（日次）実行

最新のソースコードを毎日テスト専用の QA 環境にデプロイし、３０万件の回帰テストを日次で実行するようにした。テストが失敗した場合、その原因は前日のソースコード変更に限定されるため、バグ修正のための調査時間、つまり手待ちのムダが短縮されることが期待される[7][23]。また、長期間テストを実行しないと自動テストはすぐ期待通りに動かなくなってしまう。しかし、毎日テストを実行し 100%の成功率を担保することで、自動テストが劣化することを防ぐことができる。

日次でテストを実行するためには、テスト結果が不安定な Flaky テストを最小化する必要がある[23]。そのため、テストシナリオで使うテストデータの処理方法を工夫した。すべてのテストデータを前処理と後処理で削除、作成するようにし、前回のテスト実行時のテストデータが原因でテストが失敗することを防ぐ。テスト環境に関する変数はテストスクリプトの中でハードコードせず、テスト実行者が入力変数として与えられるようにした。これにより自動テストのポータビリティが向上し、スケーラブルに構築された開発、QA どの環境でもテストを実行可能になる。

5. 結果と考察

5.1 結果

パイプラインの自動化により、開発チームは必要に応じスケーラブルな開発環境を用意し、開発、デバッグを実施できるようになった。その結果、開発環境で作り込まれるバグが減少し、QA 環境とクライアント統合テスト環境で見つかったバグの 1 週間の総数が中央値で 60 件から 22 件へと改善した。つまり、開発環境で作り込まれる不良のムダを削減することができた。

受け入れテストの自動化により、毎日、十分なカバレッジの再帰テストをクライアント統合テスト環境へのデプロイ前に実行できるようになったため、55%まで落ちていた QA 環境でのバグの検出率が 94%まで改善した。

これらの改善活動は複合的に作用し、開発・テストチーム間で生じていた手待ちのムダであるバグ修正日数も改善した。図.9(a)に示すように、導入前は中央値で 10 日かかっていたバグ修正日数が、導入後は 2 日まで改善した。これは、

1. パイプラインの自動化により、開発チームの生産性と成果物の品質が安定したこと
2. 自動テストを日次で実行することにより、バグの原因特定にかかる時間を短縮したこと

が主な要因と考えられる。

図 9.(b)に示すように、統合テスト環境の障害件数も、中央値で導入前の 3 件から導入後は 1 件まで改善した。開発側の生産性を改善することで不良のムダを軽減するとともに、テスト側の生産性を改善し手待ちのムダを減らすことで、障害件数が削減された。

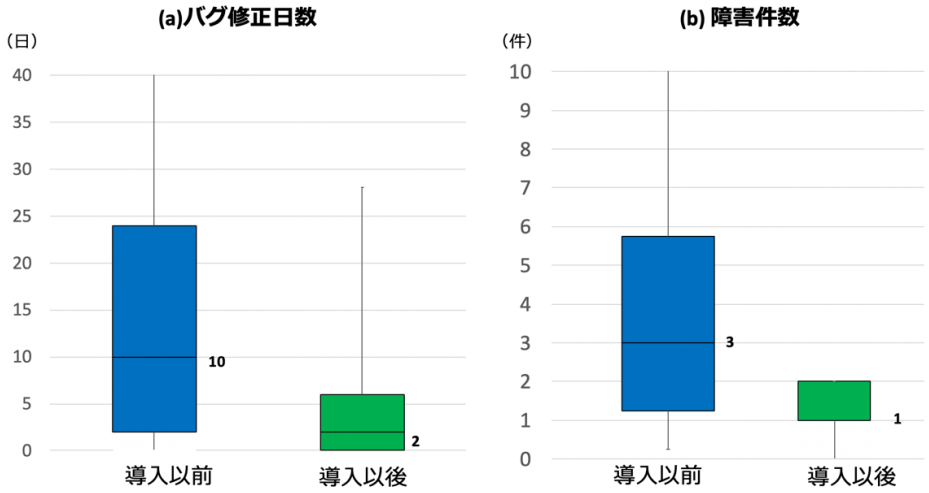

図 9 DevOps プラクティス導入前後のバグ修正日数と障害件数の比較

5.2 考察

5.2.1 QC 七つ道具についての考察

本事例では、QC 七つ道具を活用することでチームが抱えるプロセスのムダを発見した。アジャイルの導入においてはチームの条件に合わせて判断することの必要性が指摘されている[12]が、QC 七つ道具のような定量的な分析手法は、それを補助できる。

加えて、日本の経験にもとづいたソフトウェアプロセス改善のアプローチの重要性が指摘されている[13]が、QC 七つ道具を利用した QC 活動はもともと日本でのソフトウェア品質改善の黎明期に使われていた手法でもある[13]。特に開発からリリースまでのプロセス全体を素早く繰り返し可能にすることを目的とする DevOps は、ウォーターフォールよりも QC 七つ道具との親和性が高い。

また、アジャイル開発では測定値のばらつきが大きい問題が指摘されている[12]が、プロセス品質メトリクスが時系列で手に入るという長所もある。そのため、グラフや散布図といった QC 七つ道具を使い、バグ検出率や不具合修正日数などの時系列のメトリクスを効果的に分析できる。

本事例ではバグ修正日数を分析し、開発とテストの２つのチームにまたがるプロセスの手待ちのムダを見つけた。同様のアプローチは、例えば開発と運用チームにまたがる障害復旧時間や、すべてのチームにまたがるリリースのリードタイムにも適用可能だろう。

5.2.2 DevOps プラクティスについての考察

今回の事例では、DevOps プラクティス導入前に課題であった生産性の改善を達成できた。この要因は、パイプラインと受け入れテストを構成するソフトウェアに要求される品質特性を適切に設計、実装したためと考えられる。DevOps プラクティスにおける自動化は、バージョン管理ツールからリリースまでのプロセスのソフトウェア化に他ならない。やみくもな自動化をせず、スケーラビリティ、保守性、属人性の排除、ポータビリティといった品質を適切に設計、実装することが重要だ。

この考え方に則ると、DevOps 導入前に実施した QC 七つ道具による分析は、開発するソフトウェアに対する品質要求分析として考えることもできる。パイプラインや受け入れテストを構成するソフトウェアが実現すべきプロセスの品質を知り、品質要求として関係者で合意することで、新しいプラクティスの導入中・導入後の効果を可視化し、チームにフィードバックできる。

6. まとめ

開発とテストの生産性を改善するため、QC 七つ道具を用いたプロセスの分析と DevOps プラクティスの導入を実施した。QC 七つ道具を利用し、チーム間のプロセスの手待ちのムダと不良のムダを特定した。これらのムダを削減するため、パイプラインと受け入れテストを自動化した。これらの施策の結果、バグの修正日数は中央値で 10 日から 2 日に、統合テスト環境の障害件数は中央値で 3 件から 1 件に改善した。

今回の我々の取り組みは、アジャイル開発における継続的なプロセス改善の手法を示唆するものである。開発、テスト、リリースのプロセスを繰り返すアジャイル開発では、QC 七つ道具などの製造業の生産プロセスの改善手法を、効果的に適用できる。また DevOps の導入においては、ソフトウェア化するプロセスに要求される品質を分析することで、効果的な自動化を実施できる。

今回の取り組みは、1 つのプロジェクトのみを対象としているが、これは QC 七つ道具や DevOps プラクティスの効果をこのプロジェクトのみに限定するものではない。プロセスを繰り返すことが前提となるアジャイル開発においては、同様の手法が有効であると考えられる。今後、チーム規模や対象プロダクトの異なるプロジェクトにおいても検証されることが期待される。

7. 参考文献

[1] (独)情報処理推進機構： アジャイル型開発におけるプラクティス活用事例調査　調査概要報告書、https://www.ipa.go.jp/files/000026846.pdf (2019 年 07 月 28 日現在)
[2] 大野耐一、トヨタ生産方式 -脱規模の経営を目指して-、ダイヤモンド社、1978
[3] 松田亀松、QC のことがわかる本、日本実業出版社、1981
[4] 平鍋健児、変化を味方につけるアジャイル、SEC journal、Vol.11、No.4、p22-25、2016
[5] 児玉公信、アジャイル開発=コンカレントエンジニアリング=小集団活動=プログラムマネジメント、研究報告情報システムと社会環境（IS)、Vol.2015-IS-133、No.1、p1-8、2015
[6] 永田敦、プロダクト部門での品質保証担当の振る舞い　アジャイル開発の場合、JaSST’16 Tokyo、C5-1、http://jasst.jp/symposium/jasst16tokyo/pdf/C5-1.pdf (2019 年 07 月 28 日現在)
[7] 荻野恒太郎、15-B-11 安心なサービスの品質改善を実現するための継続的システムテスト、先進的な設計・検証技術の適用事例報告書 2015 年度版
[8] 川口恭伸、楽天でのエンタープライズアジャイルと DevOps -Dev/Test/Ops 三位一体の自動化-、情報処理学会デジタルプラクティス、Vol.7、No.3 p243-251、2016
[9] Gary Gruver、変革の軌跡【世界で戦える会社に変わる“アジャイル・DevOps”】、技術評論社、2017
[10] Nicole Forsgren、Lean と DevOps の科学[Accelerate] テクノロジーの戦略的活用が組織変革を加速する、インプレスブックス、2018
[11] Jez Humble、David Farley、継続的デリバリー 信頼できるソフトウェアリリースのためのビルド・テスト・デプロイメントの自動化、KADOKAWA/アスキー・メディアワークス、2012
[12] 誉田直美, アジャイル品質保証の動向、SQuBOK Review 2016、 Vol.1、p1-p10、2016
[13] 安達賢二、日本におけるソフトウェアプロセス改善の歴史的意義と今後の展開、SQuBOK Review 2016、Vol.1、p11-24、2016
[14] Kent Beck, Embracing Change with Extreme Programming, Computer, 32, 70-77, 1999
[15] 細谷克也、QC 七つ道具 100 問 100 答、日科技連出版社、2003
[16] Kief Morris、Infrastructure as Code: Managing Servers in the Cloud、O’Reilly Media、2016
[17] Enterprise Container Platform for High-Velocity Innovation、https://www.docker.com/ (2019 年 08 月 05 日現在)
[18] Production-Grade Container Orchestration - Kubernetes、https://kubernetes.io/ja (2019 年 08 月 05 日現在)
[19] Jenkins、https://jenkins.io/ (2019 年 08 月 07 日現在))
[20] Selenium - Web Browser Automation、https://docs.seleniumhq.org/ (2019 年 08 月 05 日現在)
[21] Cucumber、https://cucumber.io/ (2019 年 08 月 05 日現在)
[22] John Ferguson Smart、BDD in Action、 Manning Publications、2014
[23] 荻野恒太郎、”素早い”テスト実践法、日経 SYSTEMS、2017 年 8 月号、特集 2 p50-57、2017