開発者と品質保証担当者の協働によるアジャイル開発とシフトレフト

～ QA to AQ の実現方法の提案と生成 AI を活用した実験 ～

研 究 員：原田 聡（コニカミノルタ株式会社）
芳沢 圭一（株式会社オージス総研）
大泉 博紀（NTT コミュニケーションズ株式会社）
酒井 雄太（株式会社 Wells System Design）
金子 敬子（三菱電機ソフトウエア株式会社）
主 査 ：永田 敦（サイボウズ株式会社）
副 主 査：萩野 恒太朗（株式会社カカクコム）
アドバイザー：山口 鉄平（株式会社 LayerX）

概要

アジャイル開発の最大の特長は，変化への追随性や適応性，開発の迅速性である．これは，メンバー間の密な連携と「協働（コラボ）」によってもたらされるという側面が大きい．ただ現実的には，開発者（Dev）と品質保証担当者（QA 担当者）とプロダクトオーナー（PO）のコミュニケーションのしにくさなどの原因によりコラボがうまく機能しないことも多い．そこで我々はその現状を確認すべく，コラボの実験を行った．その結果，コラボを阻害するいくつかの問題点を見出すことができた．次に，この問題を解決する手段のひとつとして，生成 AI を活用することを考えた．そして再度実験を行ったところ，コラボが効果的に行えることが分かった．さらにそのコラボは，シフトレフトなど様々な品質向上の効果をもたらすことを確認するに至った．

1. 序論

アジャイルソフトウェア開発宣言[1]が定められてから 20 数年が経過した昨今，デジタル化対応や DX（Digital Transformation）への取り組みの加速も相まって，我が国においてもアジャイル開発は一般化してきている．これは，アジャイル開発がもつ市場の変動や技術進化への追随性や適応性，開発の迅速性といった特長が，現代社会におけるソフトウェア開発のニーズにマッチしたものであるためといえる．

一方でアジャイル開発は，従来の開発とは異なった手法をとることも多いため，実施するには変革が要求される．またアジャイル開発は，組織文化やマインドセットの変化も必要とされる．そのために，適切なトレーニングが必要なだけでなく，組織論的なサポートや開発現場の文化的な適応も必要となる．このようなことから，実際の現場においては，アジャイル開発の理念を完全に実現するための障壁も多い．例えば，従来の方法論やプロセスに対する強い固執のある組織においては，アジャイルの導入を困難にしていることも珍しくない．

このように，アジャイル開発の「理想」と「現場」の間にはギャップが存在することがある．しかしこれはアジャイル開発自体の欠陥ということではなく，それを実施する方法や組織文化の調整に関する問題だと考えることができる．そこで私たちは，アジャイル開発の理想に対して現場にいくつか存在するギャップに着目することにした．

このギャップのうち私たちが最も問題視したのは，「自己組織的」チームを用いるというアジャイル開発の理念がうまく実現されていないことである．アジャイル開発の根底には，Dev，QA 担当者，PO が密に連携し，互いに「協働（コラボ）」することによって，各々が持つ強みを発揮しあいながら開発を進めていくという考えがある（図 1）．

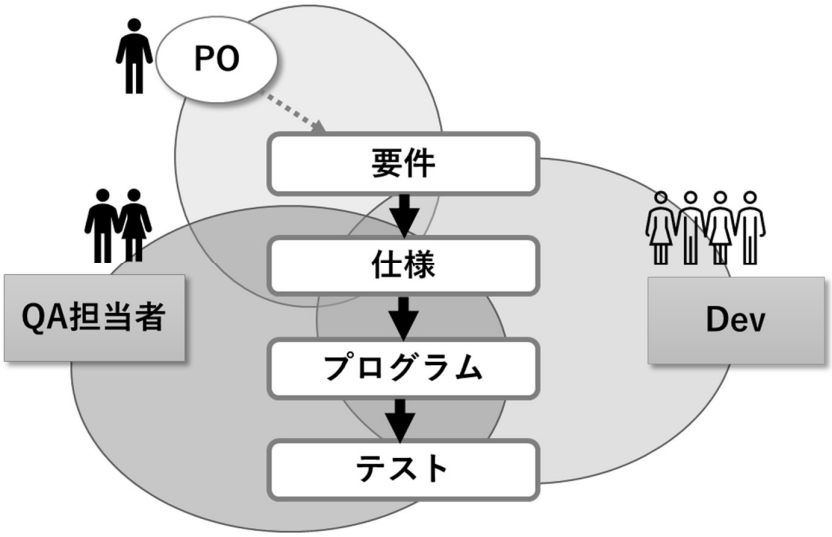

図 1 アジャイル開発における「協働（コラボ)」の概念

しかし，これらを行う中でコラボがうまく機能せず，アジャイル開発の適応性や迅速性を完全に生かし切ることを阻害する多くの「障壁」に直面しているのも現実である（図 2)．そこで，本研究では，この課題を解決することを目的とすることにした．

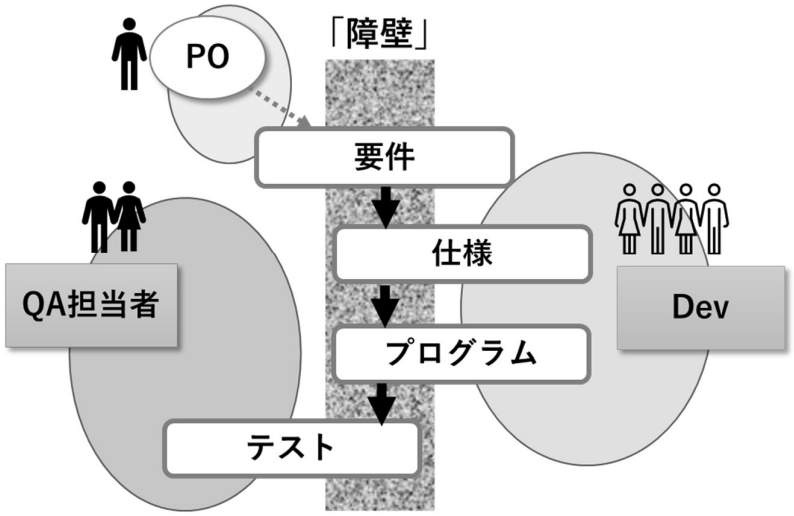

図 2 「協働（コラボ)」を阻害する「障壁」のイメージ

本論文の構成としては，第 2 章で，まずいくつかの事例を交えながら，この現状を課題として整理している.そのうえで,第 3 章でこの課題を解決するための提案内容を示した．次に第 4 章で関連研究について記述したあと，第 5 章では，現状の把握と課題解決のために本研究で行った実験の方法とその結果について記載した．第 6 章で本研究での議論内容（考察）について整理し，第 7 章で結論を述べている．最後に第 8 章では，今後の展望についても触れている．

2. 課題

2.1 具体例(1) チームが柔軟でないアジャイル開発

筆者の一人が経験したベンチャー企業におけるモバイルアプリ開発プロジェクトにおいて，アジャイル手法が導入されていることが報告されている．このプロジェクトでは，開発チームが機能開発を完了すると，その後に QA 担当者が評価を行うという一連の流れが確立されていた．このプロセスは，開発と評価の間に時間的なズレを生じさせ，プロジェクトの全体的なスケジュールを延長する原因となっていた．加えて，評価過程で見つかる問題点や不具合は，既に次の開発フェーズに移行している開発チームにとって，作業の中断や手戻りを引き起こしていた．

これは，開発チームと QA 担当者間の連携不足が，開発サイクルの遅延を引き起こしており，アジャイル手法の持つ柔軟性を活かせていない．特に，開発後の段階で発生する不具合の修正は，既に他のタスクに集中している開発チームにとって大きな負担となり，効率的な作業進行を妨げる要因となっている．

2.2 具体例(2) 開発時の品質埋め込みができないアジャイル開発

別の筆者が経験した大手企業のウェブアプリ開発プロジェクトでも，アジャイル手法が導入されていることが報告されている．このプロジェクトでは，開発チームで品質を担保する体制が取られている．しかし，開発終盤に見つかる不具合が多く，リリースの遅れを引き起こしていた．

この問題は，開発チームに品質の専門家がいないことが原因として挙げられる．これは，アジャイル開発の効率性と柔軟性を妨げていると考えられる．アジャイル開発では，チーム内でのスキルや知識の共有，および連携が重要である．

3. 解決方法の提案

本論文では，アジャイル開発の理想に対して現場に存在するギャップを減らす方法の一つとして，Dev と QA 担当者と PO のコラボを提案する．具体的には，ペアワークやモブワークにより共にコードを見たり書いたりすることで，コーディング段階での品質確保と開発終盤での手戻り削減を目指す．さらには，従来多く作成されていたドキュメントを削減したり，チーム間の連携を強化したりする効果を期待する．

特に，テストケースが仕様の役割を果たすため，仕様書を削減することが期待できる．これらのテストケースはソフトウェアの要件や期待される振る舞いを具体的に表現し，従来型の詳細な仕様書の代わりとなり得る．

ただし，このアプローチを成功させるためには，QA 担当者と PO がコードに慣れるための支援が必要である．例えば，テストコードの定義を自然言語で記述して理解しやすい形にする，ペアワークを円滑に行うための方法を模索する，テスト容易性を高めるソフトウェアアーキテクチャの採用，テスト対象の適切な選定，変更を記録する構成管理ツールの活用などが考えられる．これらの対策を通じて，アジャイル開発の潜在的な利点を最大限に引き出すことができるだろう．

4. 関連研究

本研究に関連する研究として，アジャイルプロセスで効率的かつ効果的に品質保証を行うための有用なアジャイル品質保証のパターン集“QA(Quality Assurance) to AQ(Agile Quality)”[2]がある．“QA to AQ”という名称には「伝統的な品質保証からアジャイル品質へと変わっていこう」「昔ながらの品質保証の考え方から脱却し，アジャイルプロセスに適合する形でよりアジャイルな方法で品質保証を進めよう」といったメッセージが込められている．[3]

このパターン集に含まれるパターンのひとつ“Pair with a Quality Advocate（以下，

QA リーダーとペアリングパターン）”は，開発者と他のアジャイルチームメンバーをペアにして QA を行い，品質関連のタスクを完了させ，QA の知識と品質の視点を広めるアプローチである．

具体的な実践方法としては，QA 担当者が開発者とペアを組み，一緒に仕事をしながらユニットテストの設計・作成・結合テストを共同で行う．また QA リーダーとのペアリングは，アジャイルチーム全体に品質作業を分散させることにより，QA を含む One チームの考え方を根付かせることにも寄与する．また,Pair with a Quality Advocate において Joseph W. Yoder は，「QA リーダーとペアリングのパターンを用いることで，開発者が自ら優れたテストシナリオを思いつくようになった」と述べている．本研究においても，開発メンバーと QA メンバーのペアワークを促進するアプローチをとっており，「QA リーダーとペアリングパターン」で示された効果を本研究でも期待できるといえる．

また，「QA リーダーとペアリングパターン」においては，パターンの考え方までは示されているが，パターンの具体的な手法までは示されていない．QA 担当者と開発者が協働するのは良いことだということは多くの QA 担当者や開発者にとって既知のことである．しかし，実際に QA 担当者と開発者が協働に向けた行動を起こして協働を実現させるのは，パターンの理解よりも難しいといえる．

研究においては，「QA リーダーとペアリングパターン」からもう一歩踏み込み，開発者と QA 担当者の具体的な協働手法まで検討し実験検証まで行っている．本研究は「QA リーダーとペアリングパターン」の実現方法を示している点がトピックといえる．

5. 実験方法と結果

5.1 実験内容

解決方法の提案で述べた Dev と QA 担当者と PO のコラボを実証するために，以下のシナリオで実験を行った．

対象物　:ToDoList アプリ.アプリ作成までは実施せず,モック作成とモデル作成まで.
フェーズ：結合テスト．テスト作成～実施～改善 （モデル）を参加者全員で繰り返す.
参加者　：Dev，QA 担当者，PO
テストシナリオの定義：自然言語

5.2 実験結果

5.2.1 実験第 1 回目

振る舞い駆動開発(以下 BDD)[4]で一般的に利用されている Gherkin 記法[5]を採用した．QA 担当者と PO がテストシナリオを Dev へ自然言語で伝えていき，Dev がそれを Gherkin 記法で記述していく．仕様が自然言語で記述されるため，QA 担当者や PO でも何がテストされているのかが容易に理解することが出来る．しかし，シナリオの前提条件や期待値など Gherkin 記法によるテスト仕様を記述していく過程で問題が発生した．

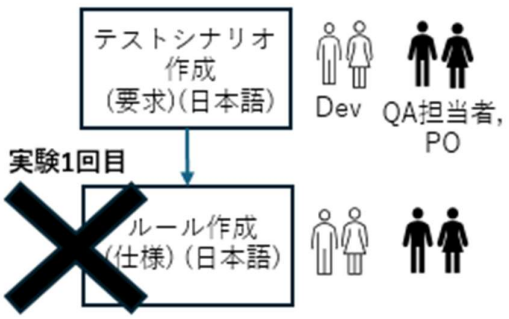

図 3 実験第 1 回目のフロー

各テストのシナリオ出しまでは,Dev と QA 担当者と PO で確認しながらスムーズに実施できたが，シナリオの前提条件やアクション，期待値を全て網羅してアウトプットする必要があったため，そこで多大な時間を費やしてしまった．また，Dev が Gherkin 記法に不慣れであったため，テストがパスできず，その原因解明にも時間を要してしまった．その

際は，QA 担当者と PO が時間を持て余してしまった．

その結果，実験第 1 回目では，テストを通すためのテストコード作成作業に注力してしまい，参加者はテストの内容に集中することができなかった．また，QA 担当者と PO は Dev の状況を理解し，協働することが困難であった．

5.2.2 実験第 2 回目

第 1 回目の実験の反省点(QA 担当者と PO が時間を持て余してしまった)を活かし,第 2 回目の実験では，BDD の実例マッピング手法[6]を参考に Python でテストケースの作成まで実施した．議論の中で各シナリオの共通理解が深まり，矛盾点に早く気付くようになった．これにより，Dev と QA 担当者と PO の協働が効果的に出来るようになった．しかし,Dev がどのようにテストを実現するか，テストコードを作成していく過程にて困難が生じた．

テストコード作成の間，QA 担当者と PO を待たせているというプレッシャー，終始見られていることによる緊張から，構文やメソッドを調べるための時間が十分に取れず，エラーが多発し，トライアンドエラーを多く繰り返してしまった．最終的には，テストコードを全てパスすることが出来ず，持ち帰り調査・修正したいと Dev 側から申し出ることとなってしまった．

結果として,参加者がテストの実施作業(何を確認すべきかを考える)に注力できたものの,QA 担当者と PO の待ち時間が解消されなかったこと，また Dev としては QA 担当者と PO から一挙手一投足を見られているように感じられ心理的負荷が高いことから，テストを継続することが出来なかった．

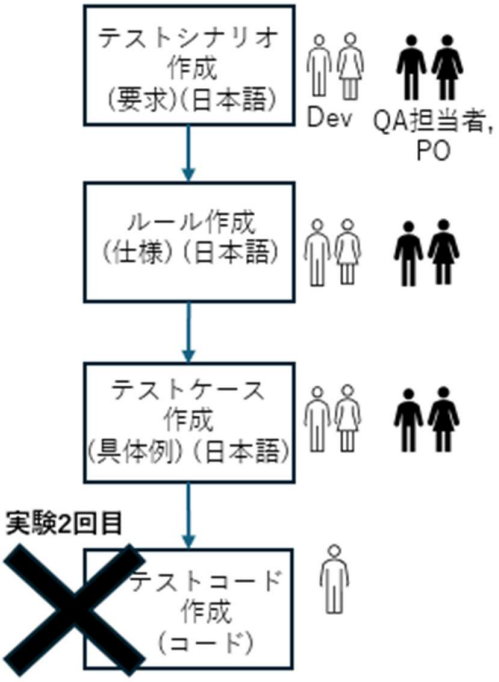

図 4 実験第 2 回目のフロー

5.2.3 実験第 3 回目

第 2 回目の実験の反省(QA 担当者と PO の待ち発生が多い点，Dev のストレスが大きい点)を生かし，Dev のアシスタントとして生成 AI を活用した．アプローチは第 2 回目と同様とし，Python で記述した．

エラー解消のシーンで，生成 AI にコード案を提示してもらうことにした．これによって，Dev の調査時間，考えこむ時間を削減し，エラー解決のヒントをタイムリーに得ることができた．

これによってまず，Dev のプレッシャーが激減し，本手法を継続することに手応えを感じることができた．

また，テストコード作成速度が飛躍的に改善し，テストがスピーディーに実施できるようになったことで，QA 担当者がテスト内容に集中することができ，高速かつ反復的にフィードバックを返すことが可能となった．

これらの結果，Dev，QA 担当者，PO 間のコミュニケーションが円滑になった．また，テスト内容に QA 担当者の視点を組み込むことが出来るようになっただけでなく，仕様の不備に気づき，早期改善を実現すること(シフトレフト）が実現できた．

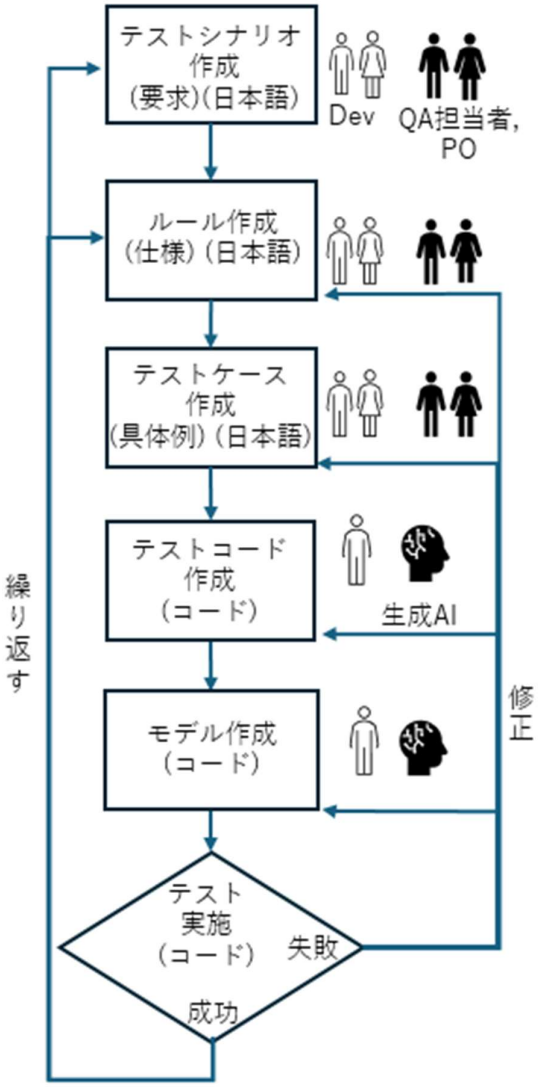

図 5 実験第 3 回目のフロー

6. 考察

実験結果をもとに，2 項で述べた打ち手に実現性があるか，また識別した課題が打ち手によって解決出来るかを考察する．

6.1 開発時に QA 担当者の視点を組み込むことができたか

(1) BDD の考え方の一例を取り入れることで，コーディング言語に関わらず，QA 担当者と PO はテストシナリオを作成することが可能となった．

(2) テストをペアプロする中で，新しいシナリオを適宜インプットしテスト自体を改善するという流れが作られた．

(3) シナリオ作成，テスト実施を何度も共同で行うことで，最初のテストモデルを変えることが出来るため，早期に認識の齟齬を統一し，後工程での修正を削減するシフトレフトを実現できた．

6.2 Dev と QA 担当者と PO の協働（コラボ）の「障壁」の軽減ができたか

(1) コーディング作業を終始他者に見られることに抵抗がある Dev でも，生成 AI のサポートを受けることで，心理的な障壁が取れ，早くミスなく作業することへのプレッシャーが取り除かれた．

(2) 共通言語(自然言語)にて会話が可能となり，Dev と QA 担当者と PO の間の障壁の軽減を実感できた．Dev，QA 担当者，PO が互いに尊重し，学びあえる存在となることができた．

7. 結論

アジャイル開発の最大の特長である変化への追随性や適応性，開発の迅速性を維持しつつ品質を担保することは，ペルソナの想定誤りやシナリオのエッジケースに如何に早い段階で気付き対処できるか，つまり，シフトレフトできるかが鍵となる．

一方，シフトレフトするためには，Dev が QA 担当者や PO とコミュニケーションをより多くとる必要がある．その際，QA 担当者と PO と Dev の情報の可視化において，BDD が有効であることは今までの知見により広く知られているところではある．しかし，実際に取り組んでみると BDD を使いながら Dev が QA 担当者および PO とコミュニケーションをとるには異なる業務コンテキストや自然言語から開発言語へのトランスレーションの課題により，円滑な運用が現実的ではなかった．

そこで，生成 AI のサポートを得ることで，Dev が主として用いる開発言語を直接使用することなく，Dev，QA 担当者，PO が効果的にコミュニケーションをとることができるようになった．また，早いタイミングで細部の仕様まで確認でき，認識齟齬，仕様考慮漏れをフィードバックすることができ品質向上を確認することができた．

この研究は，アジャイル開発において品質を確保するには QA 担当者， PO を巻き込んだシフトレフトの取組が有益であること，また，その際に発生しうるコミュニケーション課題に対処するために，近年著しく進化している生成 AI を活用することでブレークスルーを起こすことができることを示唆したものである．今後，このアプローチの重要性が一層増すことが期待される．

8. 今後の展望

8.1 コーディング段階での品質確保の強化と開発終盤の手戻りの削減

コーディング段階における品質確保は，ソフトウェア開発の初期段階で問題を特定し，修正することが全体の開発効率を向上させる鍵である．本研究では，QA 視点をコーディングプロセスに早期から組み込むことの潜在的利点と，開発チームと QA チームの密接な協力が開発プロセスの終盤における手戻りを削減し，結果としてプロジェクトの効率性を高める可能性を探求した．しかし，これらのアプローチの具体的な効果はまだ実証されておらず，今後の研究で定量的に検証し，品質向上と効率性の向上に対する具体的な貢献を明らかにすることが求められる．この検証には，実際のプロジェクトでの適用例や，品質向上における QA の役割，開発フェーズごとの QA の積極的な参加が最終的な製品の品質にどのように影響するかに焦点を当てた実験が不可欠である．

8.2 テストケースによる仕様書の代替とその可能性

テストケースを従来の仕様書の代替として使用するアプローチは，開発プロセスにおける仕様の明確化と合意形成を効率的に行う新たな方法である．この方法では，テストコードが具体的な仕様の役割を果たし，同時に品質保証の基準となる．本研究では，PO，QA，Dev が協力してテストケースを通じて仕様を定義し，共有するアプローチを提案した．しかし,このアプローチが従来の仕様書を完全に置き換えることが可能かどうかについては，さらなる検証が必要である．プロジェクトの透明性と品質向上にどの程度寄与するかを定

量的に分析することが必要であろう．

8.3 QA と PO がコードに慣れるための支援

PO，QA，Dev がモブワークでテストコードを記述するという取り組みに対して，PO，QA にコードに慣れていないメンバーが多いということが課題として挙げられていた．本研究では，テストコードの定義を自然言語で記述し，かつ AI を併用する方法を取り入れた．しかし，これは一例であり，他にも様々なアプローチが考えられる．例えば，前述したテスト容易性を高めるソフトウェアアーキテクチャの検討などである．他の様々なアプローチを比較検討し，より効果的な手法を模索していくことが望まれる．

謝辞

論文作成にあたりご指導いただいた「アジャイルと品質」分科会の永田主査，荻野副主査，山口アドバイザー，および実験にご協力いただいた方々，また研究活動を支えてくれた各研究員の家族に深く感謝の意を表します．

参考文献

[1]“アジャイルソフトウェア開発宣言”，https://agilemanifesto.org/iso/ja/manifesto.html，2001

[2] Joseph W. Yoder，Rebecca Wirfs-Brock，Hironori Washizaki，“Quality Assurance to Agile Quality”，2014

[3] Hironori Washizaki，“Ultimate Agile Stories Iteration 5” QA to AQ: アジャイル品質保証のパターン集，2015

[4] 井芹洋輝，“テスト駆動開発／振る舞い駆動開発を始めるための基礎知識”，https://atmarkit.itmedia.co.jp/ait/articles/1403/05/news035_3.html#06，2014

[5] Thomas Hamilton，“Gherkin Language: Format，Syntax & Gherkin Test in Cucumber”，https://www.guru99.com/gherkin-test-cucumber.html，2024

[6] Yuya Kazama，“事例から学ぶ実例マッピングのやり方 / Example Mapping”，https://speakerdeck.com/nihonbuson/example-mapping，2020

[7] Kent Beck，和田卓人，“テスト駆動開発”，https://www.ohmsha.co.jp/book/9784274217883/，2017