アジャイル開発に適した品質チェック項目の作成手法

谷崎 浩一[1]　田上 諭[1]　森 龍二[1]　蛭田 恭章[1]　森崎 修司[2]

概要：過去のソフトウェア開発プロジェクトで検出された欠陥情報を分析し，アジャイル開発に適した品質チェック項目を作成する手法を提案する．提案手法はアジャイル開発のプラクティスとステークホルダに着目し，プラクティスと対応付く形で品質チェック項目を作成できる．商用ソフトウェア製品の開発プロジェクトで検出された過去の欠陥情報を用いたケーススタディを行い，分析者が欠陥情報から品質チェック項目を作成できること，作成された品質チェック項目を用いることで欠陥の早期検出が可能なことを確認した．品質チェック項目を利用することで，既知の欠陥の 75%が早期に検出可能であったという結果が得られた．欠陥を検出可能な品質チェック項目の分析により，アジャイル開発で見逃しやすい欠陥の検出に有用であることが確認できた．

1. はじめに

ウォータフォール型開発は，工程ごとに次工程に移行するための品質チェックの基準が設けられ，後戻りのないように上流工程から品質を作り込んでいく．一方，アジャイル開発などのイテレーティブな開発手法では，要件定義・設計・開発・テストが繰り返し実施されるため工程の概念が希薄で，従来は工程ごとに実施されていた品質チェックが暗黙的となりやすく，欠陥の発見が遅れることがある．アジャイル開発により出荷後の欠陥数を 13%削減できたという報告 [1] や自己組織化をはじめアジャイル開発でのプラクティスが品質に良い影響を与えたという報告 [2] があるが，文献 [3] や [4] で指摘されているように，アジャイル開発での品質保証はウォータフォール型開発の品質保証の方法とは異なる．ウォータフォール型の工程ごとの品質チェックとは異なる観点で，アジャイル開発に適した品質チェックを検討する必要がある．

これまでにもアジャイル開発とウォータフォール型開発を比較した研究はあるが，アジャイル開発のメリットの一つである欠陥の早期検出を目的としてアジャイル開発に適した品質チェック項目を検討した研究はない．たとえば，アジャイル開発での品質保証活動をウォータフォールモデルでの品質保証活動と対応付けた上で，アジャイル開発のプラクティスに付加する形で品質保証活動を提案している [5] が，欠陥の早期発見に関する言及はない．また，アジャイル開発において，検出された欠陥を ODC(Orthogonal Defect Classification)[6] により分析し，欠陥の分類ごとの出現頻度の割合を調査した研究 [7] はあるが，欠陥の早期検出に言及はない．文献 [8] では，アジャイル開発の代表的なプラクティスの一つであるペアプログラミングとウォータフォール型開発で実施されるピアレビューを工数と品質の観点で比較しているが，欠陥の早期発見に関する比較はない．

そこで，本論文では， 個々のアジャイル開発プロジェクトにおいて，開発・テストの着手前の早期段階から欠陥を検出できるようになることを目的とし，アジャイル開発に適した品質チェック項目の作成手法を提案する．特に，ソフトウェア開発に関わるステークホルダの意見を取り込み，多様な視点からさまざまな品質チェック項目を議論するために，実際にアジャイル開発で検出できた欠陥のうち，より早期に検出が可能であった欠陥を対象とする．具体的には，アジャイル開発のプラクティスとステークホルダを列挙し，どのプラクティスでどのようなステークホルダの視点からチェックが必要かを検討し，品質チェック項目を作成する．品質チェック項目を作成するにあたり，過去に検出された不具合を分析する．商用ソフトウェア製品の開発プロジェクトを対象としたケーススタディにおいて，過去の不具合の情報から品質チェック項目を設定できるかどうか，また，品質チェック項目を利用することで欠陥の早期検出ができるかどうかを検討する．

以降，2. で提案手法を説明し，3. で商用ソフトウェア製品の開発プロジェクトを対象としたケーススタディを示し，4. で考察する．5. で関連研究を述べ，6. でまとめる．

[1] 株式会社ベリサーブ
VeriServe Corporation
[2] 名古屋大学大学院情報科学研究科
Graduate School of Information Science Nagoya University

2. 提案手法

2.1 手法の要件

実運用を考慮して筆者らが議論した結果から，提案手法の要件を以下の 7 個にまとめた．

R1 品質チェック項目を構築するにあたり入力とする情報は，具体的なものであること
品質チェック項目を複数の分析者で洗練できるよう，具体例を通じて分析できることが望ましい．

R2 複数の独立した品質チェック項目としてリスト化できること
リストにしておき，項目間の依存関係を小さくすることで取捨選択や改善しやすくする．

R3 品質チェック項目を使うと，使わない場合と比較して早期に問題を検出し，是正できること
チームメンバが品質チェック項目を確認することにより，問題に気付きやすくなることを目指す．また，問題に気付くだけでなく，是正，修正も早期に実施できることが望ましい．

R4 対象プロジェクトに応じて，品質チェック項目を取捨選択 (テーラリング) できること
提案手法により作成した品質チェック項目を他のプロダクトにおいて流用する際に，必要のないものを削除することや、必要に応じて部分的に変更することができる．

R5 プロダクトに対するさまざまなステークホルダからの多様な視点から分析できること
品質チェック項目は多岐にわたると考えられるため，多様な視点からチェック項目を挙げられる必要がある．

R6 具体的なプラクティスに対応付けてチェックできること
品質チェック項目の内容を開発の中で実際に確認する場合には，どのようなタイミングで実施するかが明らかになっている必要がある．アジャイル開発ではプラクティスが開発メンバにとっての共通理解になるため，プラクティスに対応付けて品質チェック項目が定義されていることが望ましい．

R7 提案手法の適用において，特殊なスキルが必要にならないこと
高度なスキルがなくても，手順に従えば問題の早期発見ができるような品質チェック項目を作成できること

2.2 概要と前提

提案手法の概要を図 1 に示す．提案手法では，分析者が具体的な議論をしやすいように過去に記録された欠陥の記録を使う (**R1** より)．図 1 において，リポジトリは分析対象プロジェクト r において検出された欠陥 D_r を蓄積したものである．リポジトリには，n 件の欠陥が記録されているものとする．つまり，$D = \{D_1, D_2, \cdots, D_n\}$ である．図 1 中の分析者は，対象プロジェクトの欠陥 D を分析し品質チェック項目リスト Q を作成する．品質チェック項目リストは m 件のチェック項目から成る (**R2** より)．つまり，$Q = \{Q_1, Q_2, \cdots, Q_m\}$ である．別のプロジェクト r' において，そのプロジェクト用に品質チェック項目を利用する際には，図 1 中の品質チェック項目の利用者が Q からチェック項目を取捨選択し，$Q_{r'}$ を作成する．つまり，$Q_{r'} \subseteq Q$ である (**R4** より)．

分析者は，$D_i (1 \leq i \leq n)$ から品質チェック項目を作成する際に，ステークホルダ別に品質チェック項目を作成する (**R5** より)．品質チェック項目は，具体的なプラクティス P_j を想定して，作成する (**R6** より)．品質チェック項目の作成には，ステークホルダごとの視点を設定することで，より具体的な項目となるようにする．具体的には，まず，品質チェックを行う際に各ステークホルダ S_k が考慮する視点を定める．アジャイル開発では，開発チームをはじめ，ステークホルダに複数の役割が存在する場合があり，その場合はステークホルダの役割ごとに視点を設定する．次にステークホルダの視点を考慮してチェック項目を検討する．検討したチェック項目は，実践するプラクティス P_j を想定したものとした上で，品質チェック項目として定義する．

2.3 想定するステークホルダとプラクティス

想定するステークホルダは，対象のプロジェクトで開発する製品に関連するステークホルダとする．さまざまな立場における品質チェックの視点を列挙できるように，個々の人物は対象とせず，ステークホルダの役割に着目する．ステークホルダの役割の具体例として，製品を利用する顧客，製品の開発に関わるプロダクトオーナや開発チームなどが挙げられる．

ステークホルダの視点の概念モデルを，図 2 に示す．図 2 は，アジャイル開発の一つの手法であるスクラムにおけるステークホルダ (顧客，プロダクトオーナ，スクラムマスタ，開発チーム) と同一のステークホルダの役割 (開発チームの中でのデザイナの役割，開発者の役割，テスタの役割) を示している．顧客にとっては，提供される製品や機能が顧客自身の課題を解決し価値を提供してくれるものであるかどうかが重要なため，顧客の視点での品質チェックでは「課題を解決できるか」という視点が必要である．プロダクトオーナは製品の価値を最大化するために何を開発するかに責任を持つことから，プロダクトオーナの視点での品質チェックでは「作る価値があるか」という視点が必要である．開発チームは，プロダクトオーナから示された開発項目や仕様をどのように実現するかに責任を持つこ

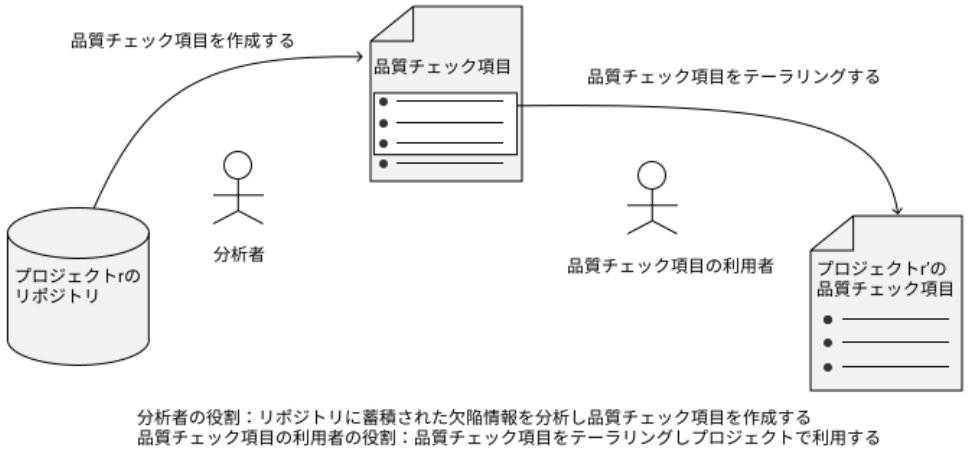

図 1 提案手法の概要
Fig. 1 Overview of proposed approach.

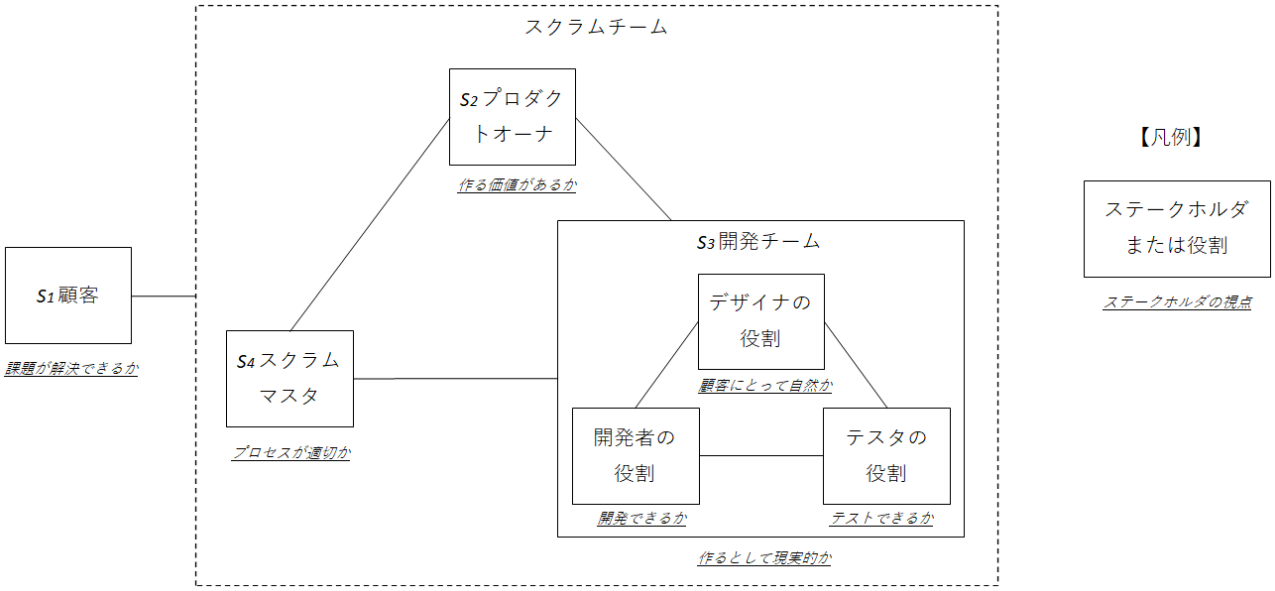

図 2 ステークホルダの視点の概念モデル
Fig. 2 Conceptual model for perspective of stakeholders.

とから，開発チームの視点での品質チェックでは「作るとして現実的か」という視点が必要である．開発チームの中のデザイナの視点では，顧客が迷わず製品を利用できるようなユーザインタフェースの制作に責任を持つことから，「顧客にとって自然か」という視点が必要である．開発チームの中の開発者の視点では，実際に製品や機能を開発することに責任を持つことから，「開発できるか」という視点が必要である．開発チームの中のテスタの視点では，開発された製品や機能をテストすることに責任を持つことから，「テストできるか」という視点が必要である．スクラムマスタは開発チームのプロセスに責任を持つことから，スクラムマスタの視点での品質チェックでは「プロセスが適切か」という視点が必要である．

提案手法は開発・テストに着手する前段階での欠陥の早期検出を目的とするため，想定するプラクティスは，開発・テストの着手に必要な成果物を生み出すプラクティスとする．具体的には，バックログアイテム作成，バックログリファインメントや受け入れテストのテスト設計などが，想定するプラクティスである．これらのプラクティスの成果物である，ユーザストーリーやソフトウェア要件，それらを記述したプロダクトバックログアイテム，テストケースなどが品質チェックの対象となる．

2.4 手順

提案手法の具体的な手順は，以下のとおりである．

2.4.1 品質チェック項目の作成

(1) 分析対象となる欠陥 D を蓄積したプロジェクト r の選択
プロジェクト r は複数選んでもよい．

(2) 対象プロジェクト r で利用しているプラクティスの列挙
対象プロジェクト r のプラクティス P を列挙する．プロジェクトの情報のみから列挙が難しい場合は，インターネット等で公開されているプラクティスの一覧を参照し，プロジェクトで実施しているプラクティスを選択する．

(3) 対象プロジェクト r のステークホルダ S の列挙
ステークホルダは，開発チームのように，複数人いても構わない．

(4) ステークホルダ S_k の視点の列挙
ステークホルダ S_k ごとに品質チェックを行う際の視点を列挙する．ステークホルダに複数の役割がある場合は，役割ごとに視点を列挙する．

(5) 早期発見できる可能性のあった欠陥 D' の選択
欠陥 D_{ri} を報告するドキュメント (バグレポート) を確認し，各ステークホルダの視点から，実際に検出した時点よりも早期に欠陥 D_{ri} が発見できなかったかどうかを確かめる．より早期に発見できた可能性がある場合には，D' とする．$i = 1, \cdots, n$ まで繰り返す．このとき，早期に発見できた可能性があるか否かという観点に加え，早期に発見することで開発の手戻りを防ぐメリットが大きいか否かという観点で D' を抽出してもよい．

(6) 早期発見できる可能性のあった欠陥 D' の分析
D' に含まれる各欠陥について，対象プロジェクト r におけるプラクティス P_j において，早期に検出するための品質チェック項目 Q_w を定義する．Q_w を定義する際，同一の欠陥のみに適用可能な具体的な表現にはせず，ステークホルダの視点を参考に抽象的な表現とする．抽象的な表現とすることで，別のプロジェクト r' においても利用しやすくなる．

2.4.2 品質チェック項目を利用した欠陥検出

(1) 品質チェック項目 Q_w を利用するプロジェクト r' の選択
プロジェクト r' は複数選んでもよい．プロジェクト r を r' としてもよい．

(2) 品質チェック項目 Q_w のテーラリング
対象プロジェクト r' で利用しているプラクティス P'_j とステークホルダ S'_k を考慮し，品質チェック項目 Q_w から項目をテーラリング（取捨選択）する．テーラリングした品質チェック項目は Q'_w とする．

(3) 品質チェック項目 Q'_w を用いた欠陥の検出
対象のプロジェクト r' でプラクティス P'_j を実施した際，P'_j の成果物に対して品質チェック項目 Q'_w を利用したチェックを行い，欠陥の検出を試みる．

3. ケーススタディ

3.1 目的と方法

提案手法の実現性と効果を実証的に評価するため，後述する対象プロジェクト r の過去の欠陥情報をもとに，品質チェック項目を作成できるか，品質チェック項目を用いて欠陥の早期検出ができるか試行する．

対象プロジェクト r のリポジトリから，受け入れテスト以降に検出された過去の欠陥 D を抽出する．D をある時点 t で分け，古い欠陥のリストを品質チェック項目作成用の欠陥 $D_f = \{D_1, D_2, \cdots, D_n\}$，新しい欠陥のリストを品質チェック項目の適用先の欠陥 D_l とする．t の時点で提案手法を利用して D_f から品質チェック項目を作成し，それ以降の開発で品質チェック項目を利用していたと仮定し，D_l の欠陥を早期検出できたかどうか確認する．D_l の欠陥を早期検出できることが確認できれば，品質チェック項目により欠陥の早期検出の効果が得られたものとする．

3.2 対象

ケーススタディとして，ある企業でアジャイル開発を実践するソフトウェア開発プロジェクトに提案手法を適用する．プロジェクトの概要を，**表 1** に示す．対象プロジェクトは開発期間の途中で開発手法が変わった．2017 年までは特定の手法は用いないイテレーティブな開発を行い，半年に 1 回程度のペースで製品をリリースしていた．2018 年以降はスクラムを取り入れた開発手法を実践し，1 週間のスプリントを繰り返し，1〜3 か月程度で製品をリリースするようになった．開発期間全体を通して一つの BTS(Bug Tracking System) でバグレポートの管理を行っている．BTS に蓄積されたバグレポートのうち受け入れテスト以降に検出された 197 件をケーススタディで利用した．t は 2017 年末とし，対象プロジェクトの開発手法が変わる前後で D を分けることとした．2018 年以降はアジャイル開発をより強く意識した開発手法を取っており，品質チェック項目を用いた欠陥の早期検出の効果を確認するのにふさわしいと考えた．D_f は 120 件，D_l は 77 件であった．

対象プロジェクトのプロダクトオーナが試行した．

3.3 結果

3.3.1 品質チェック項目の作成

対象プロジェクトで実施していたプラクティスを確認し，P_1 バックログアイテム作成，P_2 バックログリファイ

表 1 対象プロジェクトの概要
Table 1 Overview of the project.

概要	テスト業務用のアプリケーション
ケーススタディの対象とした開発期間	約 3.75 年（2015 年 12 月〜2019 年 8 月）
開発規模	約 130,000 行 (Java/JavaScript)
開発手法	アジャイル開発
人数	3〜6 名
ケーススタディに利用した不具合の収集フェーズ	受け入れテスト以降

表 2 品質チェック項目リスト
Table 2 Quality Checklists.

プラクティス	ステークホルダ	ステークホルダの視点	品質チェック項目
P_1 バックログアイテム作成	S_1 顧客	課題が解決できるか	Q_1 顧客が解決したい課題が定義されているか
	S_2 プロダクトオーナ	作る価値があるか	Q_2 誰のためのものかが定義されているか Q_3 作りたいものが定義されているか Q_4 なぜ必要かが定義されているか
	S_3 開発チーム	作るとして現実的か	Q_5 仕様同士の関係性は適切か
P_2 バックログリファインメント	S_3 開発チーム	顧客にとって自然か	Q_6 実際にエンドユーザが使っていくことができるか Q_7 デザインや操作感に統一性があるか
		開発できるか	Q_8 仕様が複雑すぎないか Q_9 受け入れ基準が適切に定義されているか Q_{10} 技術的な制約が考慮されているか
		テストできるか	Q_{11} 実行すべきテストケース・テストシナリオを特定できるか
P_3 受け入れテストのテスト設計	S_3 開発チーム	テストできるか	Q_{12} 受け入れ基準を満たすことを確認できるテストケースになっているか
P_4 自動化されたリグレッションテスト	S_3 開発チーム	テストできるか	Q_{13} テストコードが正しく実装されているか

ンメント，P_3 受け入れテストのテスト設計，P_4 自動化されたリグレッションテストを列挙した．ステークホルダとして S_1 顧客，S_2 プロダクトオーナ，S_3 開発チームを抽出した．開発チームにはデザイナ，開発者，テスタという 3 つの役割が存在した．D_f から，早期検出可能な欠陥 (D'_f) を抽出した結果，40 件であった．欠陥を抽出する際，早期検出可能か否かという観点に加え，早期検出することで開発の手戻りを防ぐメリットが大きいか否かという観点でプロダクトオーナが欠陥を抽出した．開発の手戻りを防ぐメリットの大きさは，プロダクトオーナの主観およびプロジェクト r の経験に基づき，修正にかかる変更規模と確認テストの件数を想定した．プロダクトオーナが欠陥を分析し作成した品質チェック項目を，**表 2** に示す．13 件の品質チェック項目が作成された．

品質チェック項目の作成の具体例を 1 件示す．「メニューに細分化された項目が一列に並んでおり，エンドユーザがどのメニューをどの順番で使ったらよいか分からない」という欠陥を分析の対象とした．この欠陥がプロジェクトでどのように対処されたか確認したところ，メニューを階層化することで対処されていた．欠陥の内容と対処結果をもとに，どのプラクティスにおいてどのステークホルダの視点から品質チェックを行っていれば検出できたかを検討した．検討の結果，バックログリファインメントの際に「顧客にとって自然か」という視点で，バックログアイテムおよび付随するユーザインタフェースのデザインをチェックしていれば，この欠陥を検出できた可能性があると考えた．「顧客にとって自然か」という視点を，前述の欠陥の検出につながるように具体化し，「Q_6 実際にエンドユーザが使っていくことができるか」という品質チェック項目を作成した．

3.3.2 品質チェック項目を利用した欠陥検出

対象プロジェクト r は t 以降の開発においても，ステークホルダ S が変わらなかったこと，同じプラクティス P を実施していたことから，品質チェック項目はそのまま利用した．品質チェック項目を用いて D_l の欠陥 77 件をチェックしたところ，58 件は早期検出できることが分かった．

一例として，「複数の要素を選択して切り取り・貼り付けした際，選択された要素のうち最初の要素しか切り取られない」という欠陥があった．選択された要素すべてが切り取られるのが期待する動作だった．もともと単数の要素のみ対象だった操作を，複数の要素を対象に拡張した際に埋め込まれた欠陥である．バックログアイテムには，複数の要素に対しても単数の要素と同様の操作を可能とすることのみが示されていた．具体的な動作が定義されないまま開発に着手したことが欠陥の埋め込みの要因だった．「Q_9 受け入れ基準が適切に定義されているか」という品質チェッ

ク項目で当該バックログアイテムをチェックしていれば，P_2 バックログリファインメントの段階で検出できたと考えられる．

58 件の欠陥がどの品質チェック項目で検出可能だったか集計した結果を，**表 3** に示す．「Q_{12} 受け入れ基準を満たすことを確認できるテストケースになっているか」で検出可能だった欠陥が 26 件で最も多く，「Q_9 受け入れ基準が適切に定義されているか」が 12 件で次に多かった．「Q_{10} 技術的な制約が考慮されているか」は 3 件，「Q_5 仕様同士の関係性は適切か」は 2 件，「Q_8 仕様が複雑すぎないか」は 1 件で少なかった．プロダクトオーナの視点による「Q_2 誰のためのものかが定義されているか」，「Q_3 作りたいものが定義されているか」，「Q_4 なぜ必要かが定義されているか」，およびデザイナの視点による「Q_7 デザインや操作感に統一性があるか」という品質チェック項目から検出可能な欠陥は未然に防止されていたと考えられる．

プラクティスごとの欠陥数では「P_3 受け入れテストのテスト設計」で検出可能だった欠陥が 26 件で最も多く，「P_2 バックログリファインメント」で検出可能だった欠陥が 21 件で次に多かった．「P_1 バックログアイテム作成」で検出可能だった欠陥は 2 件であり少なかった．

品質チェック項目で検出が難しかった欠陥 19 件を分類した．結果を**表 4** に示す．最も多かったのはレアな条件に起因する欠陥で 9 件だった．次に多かったのは事前の想定が困難な入力データに起因する欠陥で 5 件だった．

4. 考察

3.1 に述べた通り，品質チェック項目の作成の観点および，品質チェック項目を用いた欠陥の早期検出の観点から考察する．

4.1 品質チェック項目の作成

試行者による 40 件の欠陥の分析の結果，13 件の品質チェック項目が作成されたが，この件数の差は，品質チェック項目が具体的な欠陥情報よりも抽象的な表現になったことに起因する．欠陥の情報を抽象化することで，同一の欠陥以外にも適用可能な品質チェック項目を作成できる．複数の欠陥から同じ品質チェック項目が作成されたことで，品質チェック項目の件数は欠陥の件数より少なくなった．

複数の欠陥から同じ品質チェック項目が作成されたことから，分析に使用する欠陥の数を減らせる可能性がある．分析対象の欠陥の数を減らすことで，短時間で品質チェック項目を作成できる．分析対象の欠陥の抽出方法は試行者の主観的な判断への依存が大きい．欠陥の修正にかかった工数や再テストの件数など，開発の手戻りの大きさを表す指標を考慮することで，分析対象の欠陥を客観的に抽出できると考えられる．

4.2 品質チェック項目を利用した欠陥検出

表 3 より，P_1 バックログアイテム作成と P_2 バックログリファインメントで検出可能な欠陥は合計 23 件だった．対象のプロジェクトではこれらのプラクティスはスプリントでの開発着手前に行っていた．全体の検出可能な欠陥 58 件のうち 40%は開発着手前に検出可能だったと言える．品質チェック項目を利用することで，開発やテストに着手する前の早い段階で欠陥を検出でき，品質を高めることが可能となる．

「Q_9 受け入れ基準が適切に定義されているか」，「Q_{12} 受け入れ基準を満たすことを確認できるテストケースになっているか」で検出可能な欠陥が多かった．これらは，ウォータフォール開発では要件定義や要件定義書のレビュー，テスト設計やテストケースのレビューといった活動で検出できるが，アジャイル開発ではそれらの活動が明確に定義されておらず，漏れやすいと考えられる．品質チェック項目を利用することで，アジャイル開発で見逃しやすい欠陥を検出できる．

「Q_{13} テストコードが正しく実装されているか」で検出可能な欠陥 9 件はリグレッションによるものだった．対象プロジェクトでは，開発手法を変更した直後に，開発とリリースのスピードを上げるためにリグレッションテストのテストコードを充実させるスプリントを設けたが，開発リソースの制約から期間を区切って実施したため，テストコードが不十分な部分が存在したと考えられる．一度開発が完了した部分に対して，後からテストコードだけを実装するスプリントを設けるのは，リリース可能な製品機能が増えずプロジェクトの成果をチーム外に示しにくいため長い期間を取ることが難しい．TDD(Test Driven Development) や TFD(Test First Development) を実践し，テストコードが常に正しく実装されている状態とすることが重要となる．品質チェック項目を用いテストコードの正しさをチェックしながら開発を進めることで，製品の品質の担保につながる．

「Q_5 仕様同士の関係性は適切か」，「Q_6 実際にエンドユーザが使っていくことができるか」，「Q_8 仕様が複雑すぎないか」，「Q_{10} 技術的な制約が考慮されているか」で検出可能な欠陥が少なかったのは，以下の理由から欠陥を未然に防止できていたためと考えられる．

- 対象のプロジェクトは t 以降は 3〜4 名で毎週ミーティングを行い，バックログの内容についてチームメンバ全員が共有し，既存機能の仕様との整合性や，実装困難な複雑さがないかの議論をできていた．
- 開発期間中にプロダクトオーナや主要な開発者の入れ替わりがなく，既存機能の仕様や，実装に利用しているライブラリなどの技術的な制約を把握できていた．
- 製品を数回リリースしエンドユーザからフィードバッ

表 3　品質チェック項目で検出可能な欠陥数
Table 3　Number of the defects detected with the quality checklists.

プラクティス	品質チェック項目	欠陥数	小計
P_1 バックログアイテム作成	Q_5 仕様書同士の関係性は適切か	2	2
P_2 バックログリファインメント	Q_6 実際にエンドユーザが使っていくことができるか	5	
	Q_8 仕様が複雑すぎないか	1	
	Q_9 受け入れ基準が適切に定義されているか	12	
	Q_{10} 技術的な制約が考慮されているか	3	21
P_3 受け入れテストのテスト設計	Q_{12} 受け入れ基準を満たすことを確認できるテストケースになっているか	26	26
P_4 自動化されたリグレッションテスト	Q_{13} テストコードが正しく実装されているか	9	9

表 4　品質チェック項目で検出できなかった欠陥数
Table 4　Number of the defects that cannot be detected.

理由	欠陥数
レアな条件に起因する欠陥	9
想定困難な入力データに起因する欠陥	5
実装の形式に起因する欠陥	4
特定のユーザ環境に起因する欠陥	1

クを得られ，エンドユーザの使い方を想定できていた．

表 2，表 3 より，欠陥の検出につながらなかった品質チェック項目が存在した．「Q_2 誰のためのものかが定義されているか」，「Q_3 作りたいものが定義されているか」，「Q_4 なぜ必要かが定義されているか」が欠陥検出につながらなかった理由として，プロダクトオーナと開発チームによる密なコミュニケーションが考えられる．毎週のミーティングでバックログアイテムの内容をプロダクトオーナから開発チームに説明し，誰のために何をなぜ開発するかをプロダクトオーナと開発チームが合意のうえで開発に着手していた．これらの品質チェック項目に該当する欠陥は未然に防止されていたと考えられる．

「Q_7 デザインや操作感に統一性があるか」が欠陥検出につながらなかった理由として，t 以降はそれ以前の製品に対する機能追加や改修を加えていく開発フェーズとなっており，t 以前に開発されたユーザインタフェースのデザインや操作感を踏襲して開発が行われたことが考えられる．

同一のプロジェクトであっても，過去の欠陥から作成した品質チェック項目のすべてで新たな欠陥が検出できるとは限らないため，プロジェクトの状況に応じて品質チェック項目のテーラリングを行うことが重要である．

表 4 から，品質チェック項目での早期検出が難しい欠陥は，事前に想定することが難しい条件に起因するものが多い．レアケースを開発着手前に網羅的に考慮することは難しい．アジャイル開発ではこのような問題が発生した場合でも素早く修正していけばよく，品質チェック項目であらゆる欠陥を開発着手前に検出する必要はない．詳細なチェック項目を多数用意して品質チェックを行うと，開発スピードの低下につながる恐れがある．提案手法では品質チェック項目を欠陥情報よりも抽象的な表現にしたことで，詳細なチェック項目が多数作成されることを防ぐ．提案手法で作成された品質チェック項目は，開発スピードの低下を避けながら，プラクティスごとに欠陥を早期検出することに役立つ．

試行したプロダクトオーナから次の意見が得られた．

- プラクティスごとに，どのチェック項目でチェックすればよいかが整理できたため，実際の業務で使いやすいと感じた．
- チェック項目がそれほど多くなく，プロジェクトに導入しやすいと感じた．ステークホルダの視点を参考に品質チェック項目を抽象的な表現にしたことで，チェック項目が多くならなかったと思う．
- 品質チェック項目が抽象的な表現なので，実際に欠陥を検出できるかはチェック項目を利用する人の経験やスキルに依存すると思われる．チェック項目で検出可能な欠陥の具体例や，チェック項目の補足説明を用意すれば，経験の少ない人でも欠陥を検出しやすくなる．
- バックログアイテム作成の際に考慮すべき観点として，ユーザストーリーの INVEST(Independent, Negotiable, Valuable, Estimable, Small, Testable) などが知られる．欠陥情報から得られた品質チェック項目だけでなく，一般的に知られる観点を加えることで，より充実したチェック項目を作成できるだろう．

4.3　妥当性

品質チェック項目の作成は 1 名の分析者が実施したため，当該分析者の気付き以外は品質チェック項目には含まれていない．ステークホルダの視点を参考に品質チェック項目を作成する際，ステークホルダの視点は記述の抽象度が高いことから，必要十分な品質チェック項目を作成できるかどうかは，分析者の気づきへの依存が大きい．しかしながら，1 名の気付きのみで多くの欠陥の早期検出が可能だったという結果が得られており，品質チェック項目の効果は高いと考えられる．複数名で分析すれば，分析者の気付きが増え品質チェック項目が増える可能性がある．必要十分な品質チェック項目を作成するための工夫は今後の課題である．

一つのプロジェクトの欠陥情報を分割してケーススタディを実施したため，他のプロジェクトに適用可能かは確認できていない．しかしながら，対象のプロジェクトは，欠陥情報を分割した時点 t の前後で開発している製品機能が異なること，開発手法が変わっていることから，異なるプロジェクトと考えることができる．

今回のプロジェクトで獲得した品質チェック項目が他のプロジェクトにおいて活用できるかどうかは，プロジェクト間の類似度に依存する．どのような条件が満たされれば，他のプロジェクトで活用しやすいかを確かめることは今後の課題である．

5. 関連研究

アジャイル開発全体で品質面の評価を行った研究が多くある．文献 [7] では，アジャイル開発において検出された欠陥を ODC[6] により分析し，欠陥の分類ごとの出現頻度の割合を調査している．しかし，視点の整理や品質チェック項目の作成には言及していない．Ilieva らはアジャイル開発の代表的な形態である XP による開発によって，出荷後の欠陥の数が 13%削減できたことを報告している [1]．文献 [2] では，自律的な組織をはじめアジャイル開発でのプラクティスが品質に与える影響を報告している．

アジャイル開発のプラクティスごとに品質面での効果を評価した研究も多くある．TDD や TFD によって外部品質が向上するかどうかを実験を通じて調べた研究 [9][10][11][12] やケーススタディを通じて調べた研究 [13][14] がある．また，コードの保守性をはじめ内部品質に着目した研究もある [15][16]．ペアプログラミングによる品質向上を確かめた研究としては，品質が向上したことを報告する研究 [17][18] やプログラマのパーソナリティによる影響を調べた研究 [19][20]，学生の演習におけるペアプログラミングの効果 [21][22]，ピアレビューとの比較 [8]，設計品質の評価 [23] がある．リファクタリングによる品質向上を調べた研究には，[24][25] がある．これらの研究は，アジャイル開発における品質を研究している点で，本研究と共通する部分があるが，過去に検出された欠陥を活用するなどして，具体的な情報をもとに他のプロジェクトでも活用できるようプラクティスに対応付けて品質チェック項目を挙げているわけではない．

6. おわりに

本研究ではアジャイル開発に適した品質チェック項目の作成手法を提案した．ある企業のソフトウェア開発プロジェクトを対象としたケーススタディにおいて，ステークホルダの視点を考慮して過去の欠陥情報を分析し，プラクティスと対応付けて品質チェック項目を作成できることを確認した．品質チェック項目を利用すると，既知の欠陥の 75%は未然に検出でき，そのうち 40%は開発着手前に検出可能という結果を得た．提案手法で作成した品質チェック項目は，欠陥の早期検出に有用だと言える．チェック項目ごとに検出可能な欠陥数から，アジャイル開発で見逃しやすい欠陥の検出に有用であることが示唆された．

今後の課題として，品質チェック項目をプロスペクティブにプロジェクトに適用した場合の効果や副作用の検証，異なるプロジェクトに品質チェック項目を適用した場合の効果の検証と適用しやすいプロジェクトの条件の調査が必要である．代表的なプラクティスごとに汎用的に利用可能な品質チェック項目を整理できれば，多くのプロジェクトで品質チェック項目を活用しやすくなる．

参考文献

[1] S. Ilieva, P. Ivanov, E. Stefanova: Analyses of An Agile Methodology Implementation, In Proc. of 30th Euromicro Conference, pp. 326-333(2004)
[2] A. Ahmed, S. Ahmad, N. Ehsan, E. Mirza, S. Z. Sarwar: Agile Software Development: Impact on Productivity and Quality, In Proc. of IEEE International Conference on Management of Innovation & Technology, pp. 287-291(2010)
[3] S. Bhasin: Quality Assurance in Agile: A Study towards Achieving Excellence, Agile India, pp. 64-67(2012)
[4] E. Mnkandla, B. Dwolatzky: Defining Agile Software Quality Assurance, In Proc. of International Conference on Software Engineering Advances 2006, pp. 36(2006)
[5] M. Huo, J. Verner, L. Zhu, A. M. Babar: Software Quality and Agile Methods, In Proc. of the 28th Annual International Computer Software and Applications Conference 2004, pp. 520-525 (2004)
[6] R. Chillarege, I. S. Bhandari, J. K. Chaar, M. J. Halliday, D. S. Moebus, B. K. Ray, M. Y . Wong: Orthogonal Defect Classification - A Concept for In-Process Measurements, IEEE Transactions on Software Engineering, vol.18, no. 11, pp. 943-956 (1992)
[7] R. Karcich, J. Cangussu: A Control Approach for Agile Processes, In Proc. of the 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), pp. 123-126(2005)
[8] M.M. Muller: Two Controlled Experiments Concerning The Comparison of Pair Programming to Peer Review, Journal of Systems and Software, vol. 78, no. 2, pp. 166-179(2005)
[9] H. Erdogmus, M. Morisio, M. Torchiano: On The Effectiveness of The Testfirst Approach to Programming, IEEE Transactions on Software Engineering vol. 31, no. 3, pp.226-237(2005)
[10] A. Gupta, P. Jalote: An Experimental Evaluation of The Effectiveness and Efficiency of The Test Driven development, In Proc. of the First International Symposium on Empirical Software Engineering and Measurement, pp. 285-294(2007)
[11] B. George, L. A. Williams: A Structured Experiment of Test-Driven Development, Information and Software Technology, vol. 46, no. 5, pp. 337-342(2004)
[12] L. Huang, M. Holcombe: Empirical Investigation towards The Effectiveness of Test First programming, Information and Software Technology, vol. 51, no. 1, pp.182-194(2009)

[13] L.O. Damm, L. Lundberg: Quality Impact of Introducing Componentlevel Test Automation and Test-driven Development Software Process Improvement, vol. 4764 of Lecture Notes in Computer Science, Springer, pp. 187-199(2007)
[14] N. Nagappan, E. M.Maximilien, T. Bhat, L.Williams: Realizing Quality Improvement through Test Driven Development: Results and Experiences of Four Industrial Teams, Empirical Software Engineering, vol. 13, no. 3, pp. 289-302(2008)
[15] C. Desai, D. S. Janzen: Implications of Integrating Test-Driven Development into CS1/CS2 Curricula, In Proc. of the 40th ACM Technical Symposium on Computer Science Education, pp. 148-152(2009)
[16] C. Sanchez, L. Williams, E.M. Maximilien: On the Sustained Use of a Test-Driven Development Practice at IBM, In. Proc. of the AGILE 2007, pp. 5-14(2007)
[17] A. Begel, N. Nagappan: Pair Programming: What's in It for Me?, In. Proc. of the Second ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 120-128(2008)
[18] H. Hulkko, P. Abrahamsson: A Multiple Case Study on the Impact of Pair Programming on Product Quality, In Proc. of the 27th International Conference on Software Engineering, pp. 495-504(2005)
[19] J. Chao, G. Atli: Critical Personality Traits in Successful Pair Programming, In Proc. of AGILE'06, IEEE Computer Society(2006)
[20] P. Sfetsos, I. Stamelos, L. Angelis, I. Deligiannis: An Experimental Investigation of Personality Types Impact on Pair Effectiveness in Pair Programming, Empirical Software Engineering, vol. 14, no. 21, pp. 187-226(2009)
[21] C. McDowell, L. Werner, H. Bullock, J. Fernald: Pair Programming Improves Student Retention, Confidence, and Program Quality, Communications of ACM, vol. 49, no. 8, pp. 90-95(2006)
[22] C. McDowell, L. Werner, H. Bullock, J. Fernald: The Effects of Pair-Programming on Performance in an Introductory Programming Course, In Proc. of the 33rd SIGCSE Technical Symposium on Computer Science Education, pp. 38-42(2002)
[23] G. Canfora, A. Cimitile, F. Garcia, M. Piattini, C.A. Visaggio: Evaluating Performances of Pair Designing in Industry, Journal of Systems and Software, vol. 80, no. 8, pp. 1317-1327(2007)
[24] R. Moser, P. Abrahamsson, W. Pedrycz, A.Sillitti, G. Succi: A Case Study on the Impact of Refactoring on Quality and Productivity in An Agile Team, LNCS, vol. 5082, pp. 252-266(2008)
[25] S. Xu, V. Rajlich: Empirical Validation of Test-Driven Pair Programming in Game Development, In Proc. of the 5th IEEE/ACIS International Conference on Computer and Information Science, pp. 500-505(2006)