テスト技術者資格制度

Foundation Level Extension シラバス
アジャイルテスト担当者

Version 2014.J01

International Software Testing Qualifications Board

Foundation Level Extension Agile Tester Working Group：Rex Black (Chair), Bertrand Cornanguer (Vice Chair), Gerry Coleman (Learning Objectives Lead), Debra Friedenberg (Exam Lead), Alon Linetzki (Business Outcomes and Marketing Lead), Tauhida Parveen (Editor), and Leo van der Aalst (Development Lead).

著者：Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, and Stephan Weber.

内部レビューア：Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael, and Erik van Veenendaal; 2013-2014.

改訂履歴

◆ ISTQB®

バージョン	日付	摘要
シラバス v0.1	2013 年 7 月 26 日	スタンドアロンセクション
シラバス v0.2	2013 年 9 月 16 日	バージョン v0.1 についての WG レビューコメントの反映
シラバス v0.3	2013 年 10 月 20 日	バージョン v0.2 についての WG レビューコメントの反映
シラバス v0.7	2013 年 12 月 16 日	バージョン v0.2 についてのアルファレビューコメントの反映
シラバス v0.71	2013 年 12 月 20 日	v0.7 に対するワーキンググループによる更新
シラバス v0.9	2014 年 1 月 30 日	ベータバージョン
シラバス 2014	2014 年 5 月 31 日	GA バージョン
シラバス 2014	2014 年 9 月 30 日	些細な誤字の修正

◆ JSTQB®

バージョン	日付	摘要
Version 2014.J01	2018 年 3 月 5 日	ISTQB Certified Tester Foundation Level Extension Syllabus Version 2014 の日本語翻訳版

目次

謝辞

このドキュメントは、International Software Testing Qualifications Board Foundation Level Working Group のチームメンバにより執筆された。

Agile Extension チームは、レビューチームおよび各国委員会のメンバによる提案と意見に感謝をしたい。

Foundation Level Agile Extension シラバスの完成時において、Agile Extension ワーキンググループのメンバは以下のとおりである。
Rex Black (Chair), Bertrand Cornanguer (Vice Chair), Gerry Coleman (Learning Objectives Lead), Debra Friedenberg (Exam Lead), Alon Linetzki (Business Outcomes and Marketing Lead), Tauhida Parveen (Editor), Leo van der Aalst (Development Lead)

著者:
Rex Black, Anders Claesson, Gerry Coleman, Bertrand Cornanguer, Istvan Forgacs, Alon Linetzki, Tilo Linz, Leo van der Aalst, Marie Walsh, and Stephan Weber.

内部レビューア:
Mette Bruhn-Pedersen, Christopher Clements, Alessandro Collino, Debra Friedenberg, Kari Kakkonen, Beata Karpinska, Sammy Kolluru, Jennifer Leger, Thomas Mueller, Tuula Pääkkönen, Meile Posthuma, Gabor Puhalla, Lloyd Roden, Marko Rytkönen, Monika Stoecklein-Olsen, Robert Treffny, Chris Van Bael, Erik van Veenendaal

本チームは、Foundation Agile Extension シラバスについてのレビュー、コメントおよび投票に参加した、各国委員会とアジャイル専門家コミュニティの次のメンバに感謝したい。
Dani Almog, Richard Berns, Stephen Bird, Monika Bögge, Afeng Chai, Josephine Crawford, Tibor Csöndes, Huba Demeter, Arnaud Foucal, Cyril Fumery, Kobi Halperin, Inga Hansen, Hanne Hinz, Jidong Hu, Phill Isles, Shirley Itah, Martin Klonk, Kjell Lauren, Igal Levi, Rik Marselis, Johan Meivert, Armin Metzger, Peter Morgan, Ninna Morin, Ingvar Nordstrom, Chris O'Dea, Klaus Olsen, Ismo Paukamainen, Nathalie Phung, Helmut Pichler, Salvatore Reale, Stuart Reid, Hans Rombouts, Petri Säilynoja, Soile Sainio, Lars-Erik Sandberg, Dakar Shalom, Jian Shen, Marco Sogliani, Lucjan Stapp, Yaron Tsubery, Sabine Uhde, Stephanie Ulrich, Tommi Välimäki, Jurian Van de Laar, Marnix Van den Ent, António Vieira Melo, Wenye Xu, Ester Zabar, Wenqiang Zheng, Peter Zimmerer, Stevan Zivanovic, and Terry Zuo.

本ドキュメントは、2014 年 5 月 31 日に開催された ISTQB®の総会で正式に承認され、発行された。

日本語訳については、Japan Software Qualifications testing Board メンバおよび以下の日本語翻訳ワーキンググループメンバにより行われた。

日本語翻訳ワーキンググループメンバ:
浅黄 友隆(ヒューマンクレスト)
上村 真季(ウェブレッジ)
須原 秀敏(ベリサーブ)
早川 隆治(JIEC)、
森 龍二(ベリサーブ)
山﨑 崇(ベリサーブ)

本書は、「4.参考文献」章にある日本語翻訳文献の慣習を参考に翻訳している。

0. 本シラバスの紹介

0.1 本書の目的

本シラバスは、アジャイルテスト担当者向けの国際ソフトウェアテスト資格 Foundation Level のベースとなる。ISTQB®は、本シラバスを次の趣旨で提供する。

- 各国の委員会に対し、各国語への翻訳および教育機関の認定の目的で提供する。各国の委員会は、本シラバスを各言語の必要性に合わせて調整し、出版事情に合わせてリファレンスを修正することができる
- 試験委員会に対し、各シラバスの学習目的に合わせ、各国語で試験問題を作成する目的で提供する
- 教育機関に対し、コースウェアを作成し、適切な教育方法を確定できるようにする目的で提供する
- 受験志願者に対し、試験準備の目的で提供する（研修コースの一部として、または独立した形態で実施）
- 国際的なソフトウェアおよびシステムエンジニアリングのコミュニティに対し、ソフトウェアやシステムをテストする技能の向上を目的とする他、書籍や記事を執筆する際の参考として提供する

ISTQB®では、事前に書面による申請があった場合に限り、第三者がこのシラバスを先に定めた以外の目的での使用を許諾することがある。

0.2 概要

『Foundation Level シラバス日本語版アジャイルテスト担当者概要』[ISTQB_FA_OVIEW]には、次の情報を記載している。

- シラバスのビジネス成果
- シラバスの概要
- 各シラバスの関係
- 知識レベルの説明（K レベル）
- 付録

0.3 試験のための学習の目的

学習の目的はビジネス成果を支援し、Foundation Level アジャイルテスト担当者認定を取得するための試験問題作成を行うために使用する。基本的に本シラバスのすべての箇所は K1 レベルで試験対象となる。つまり、受験志願者は、用語や概念について認識し、記憶し、想起することになる。関連する章の最初には、K1、K2、K3 レベルにおける特定の学習の目的を記載する。

1. アジャイルソフトウェア開発 - 150 分

用語

アジャイルマニフェスト、アジャイルソフトウェア開発、インクリメンタル開発モデル、イテレーティブ開発モデル、ソフトウェアライフサイクル、テスト自動化、テストベース、テスト駆動開発、テストオラクル、ユーザストーリー

アジャイルソフトウェア開発の学習の目的

1.1 アジャイルソフトウェア開発の基本

FA-1.1.1　(K1)アジャイルマニフェストに基づくアジャイルソフトウェア開発の基本概念を想起する。
FA-1.1.2　(K2)チーム全体アプローチの利点を理解する。
FA-1.1.3　(K2)早期かつ頻繁なフィードバックの利点を理解する。

1.2 アジャイルアプローチの特徴

FA-1.2.1　(K1)アジャイルソフトウェア開発アプローチを想起する。
FA-1.2.2　(K3)開発者およびビジネス代表者と協調してテスト可能なユーザストーリーを記述する。
FA-1.2.3　(K2)アジャイルプロジェクトにおけるプロセス改善のメカニズムとしてふりかえりを使用する方法を理解する。
FA-1.2.4　(K2)継続的インテグレーションの効用と目的を理解する。
FA-1.2.5　(K1)イテレーション計画とリリース計画の違いおよびテスト担当者がそれぞれの活動で価値を提供する方法について学習する。

1.1 アジャイルソフトウェア開発の基本

アジャイルプロジェクトのテスト担当者は、従来型のプロジェクトのテスト担当者とは異なる方法で作業する。テスト担当者は、アジャイルプロジェクトを支える価値と原則、そして、開発者とビジネス代表者とが一体となるチーム全体アプローチにとって、テスト担当者がいかに不可欠な要素となるかを理解しなくてはならない。アジャイルプロジェクトのメンバは、初期から頻繁に相互にコミュニケーションする。これにより、欠陥を早期に除去し、品質の高いプロダクトを開発できる。

1.1.1 アジャイルソフトウェア開発とアジャイルマニフェスト

2001 年、最も広く用いられている軽量ソフトウェア開発方法論を提唱する代表的な人々が集まり、価値と原則の共通セットについて合意した[Agilemanifesto]。それは、アジャイルソフトウェア開発マニフェストまたはアジャイルマニフェストとして知られるようになった。アジャイルマニフェストは、価値に関する次の 4 つの宣言を含む。

- プロセスやツールよりも個人と対話を
- 包括的なドキュメントよりも動くソフトウェアを
- 契約交渉よりも顧客との協調を
- 計画に従うことよりも変化への対応を

アジャイルマニフェストは、「よりも」の左側の概念にも価値はあるが、右側の概念はさらに価値があることを主張している。

個人と対話

アジャイル開発は、極めて人間中心主義である。ツールやプロセスではなく、チームが最も効率的に作業するための継続的なコミュニケーションと対話を通して、チームでソフトウェアを構築する。

動くソフトウェア

顧客の視点から見ると、動くソフトウェアは、過度に詳細なドキュメントよりも役に立ち、価値がある。また、開発チームに迅速にフィードバックするための機会を提供する。さらに、動くソフトウェアは、機能的には劣っていたとしても、開発ライフサイクルの早期から利用できるので、アジャイル開発は市場への参入時期を大幅に早めるという利点がある。このため、アジャイル開発は、急速に変化するビジネス環境で特に役に立つ。それは、問題およびソリューションが明確でない場合や、ビジネスが新しい問題領域でイノベーションを起こそうとする場合である。

顧客との協調

顧客は、自身が必要とするシステムの仕様を決めるのが困難であることが多い。顧客と直接協調することにより、顧客が必要とすることを正確に理解する可能性が高まる。顧客との契約関係は重要であるが、顧客と定期的かつ緊密に協調することにより、プロジェクトが成功する確率がさらに高まる。

変化への対応

ソフトウェアプロジェクトでは、変化は避けることのできないものである。ビジネスが運営されている環境、法規制、競合他社の状況、技術進化などの要因により、プロジェクトおよびその目標は大きく影響を受ける。開発プロセスによって、これらの要因に適応させる必要がある。このため、作業する上で変化を受け入れる柔軟性を持つことは、計画に固執するよりもはるかに重要である。

原則

アジャイルマニフェストの核となる価値(コアバリュー)は、次の 12 の原則で定義される。

- 顧客満足を最優先し、価値のあるソフトウェアを早く継続的に提供する
- 要求の変更はたとえ開発の後期であっても歓迎する。アジャイルプロセスは変化を利用してお客様の競争力を引き上げる
- 動くソフトウェアを、2～3 週間から 2～3 カ月というできるだけ短い時間間隔でリリースする
- ビジネス側の人と開発者は、プロジェクト全体を通して日々協力して働かねばならない
- 意欲に満ちた人々を集めてプロジェクトを構成する。環境と支援を与え、仕事が無事終了するまで、開発チームメンバを信頼する
- 開発チームに対する情報伝達、または開発チーム内での情報伝達の最も効率的で効果的な方法はフェイスツーフェイスの対話である
- 動くソフトウェアこそが進捗の最も重要な尺度である
- アジャイルプロセスは、持続可能な開発を促進する。スポンサ、開発者およびユーザは、一定したペースを継続的に維持できるようにしなければならない
- 技術的卓越性と優れた設計に対する不断の注意が、機敏さを高める
- シンプルさ(やらない仕事を最大化する技)が本質である
- 最良のアーキテクチャ、要求および設計は、自己組織的なチームから生み出される
- もっと効率を高めることができるかを開発チームは定期的にふりかえり、それに基づいて自分たちのやり方を最適に調整する

さまざまなアジャイル方法論が、これらの価値と原則を実行に移すための規範的なプラクティスを提供する。

1.1.2 チーム全体アプローチ

チーム全体アプローチは、プロジェクトを成功させるために必要な知識とスキルを持つすべての人々を巻き込むことを意味する。チームには、プロダクトのフィーチャを最終決定する顧客およびビジネスのステークホルダの代表にも参加してもらう。チームの規模は比較的小さくする必要がある。これまでに成功したチームは、少なくとも 3 人、多くて 9 人のメンバで構成されている。理想的には全員参加型のチームは同じ作業場所を共有することが望ましい。というのも、共同の作業環境は強力にコミュニケーションと対話を促進するからである。チーム全体アプローチは、チームのすべてのメンバが参加するデイリースタンドアップミーティングによって支えられている(2.2.1 項を参照)。このミーティングを通して、作業の進捗が周知され、進捗に対するすべての障害が明らかにされる。チーム全体アプローチは、より効果的かつ効率的なチームの活動力を高める。

プロダクト開発へのチーム全体アプローチの効用は、アジャイル開発の主要な利点の一つであり、それには次のことが含まれる。

- チーム内のコミュニケーションと協調が強化される
- チーム内の多様なスキルを、プロジェクト全体の利益に広げることを可能にする
- 品質を全員の責務にする

アジャイルプロジェクトにおいては、チーム全体が品質に対する責任を持っている。チーム全体アプローチの本質は、テスト担当者、開発者およびビジネス代表者が、開発プロセスの各ステップで協力して作業することにある。テスト担当者は、開発者およびビジネス代表者と密接に連携し、望まれる品質レベルが確実に達成されるようにする。これには、適切な受け入れテストを作成できるようにビジネス代表者をサポートし協調すること、開発者と協調して、テスト戦略に合意し、テスト自動化の方式を決定することが含まれる。このように、テスト担当者は、テストに関する知識を他のチームメンバに伝達および展開して、プロダクトの開発に良い影響をおよぼすことができる。

チーム全体で、プロダクトのフィーチャの提示、分析、または見積りが行われるすべての検討会やミーティングに参加する。テスト担当者、開発者、ビジネス代表者がフィーチャの検討すべてに参加するというコンセプトは、

「三人の力」として知られている[Crispin08]。

1.1.3 早期かつ頻繁なフィードバック

アジャイルプロジェクトは短期のイテレーションで構成され、プロジェクトチームは開発ライフサイクル全体を通して、プロダクト品質に関して早期かつ継続的フィードバックを受け取ることができる。フィードバックを迅速に提供する一つの方法は、継続的インテグレーションである（1.2.4 項を参照）。

シーケンシャル開発アプローチを使用していると、多くの場合、顧客はプロジェクト完了直前までプロダクトを見ることはできない。プロジェクト完了直前では、開発者が顧客の経験するすべての問題に効果的に対処するには遅すぎる場合が多い。プロジェクトが進行する間に顧客からのフィードバックを頻繁に取得することにより、アジャイルチームは新しい変更の大部分を、プロダクト開発のプロセスで取り込むことができる。早期かつ頻繁なフィードバックは、チームが最も高いビジネスバリューをもたらすフィーチャやそれに付随するリスクに集中することを助け、これらのフィーチャを優先して顧客にデリバリーできる。また、チームでできることが全員に見えるようになるため、よりチームをマネジメントしやすくなる。たとえば、スプリントまたはイテレーションでできる作業の量、よりスピードアップするために必要なこと、作業の妨げとなっているものは何かである。

早期かつ頻繁にフィードバックを得ることの利点には、次のものがある。

- 要件の誤解が、修正コストがより高く付く開発サイクルの終盤まで検出されないことを避けることができる
- 顧客がフィーチャに要求することを明確化でき、顧客はフィーチャを早く使うことができる。これにより、顧客の希望がプロダクトに、より正しく反映される
- 品質問題を早期に発見（継続的インテグレーションを介して）、分離および解決できる
- アジャイルチームに対して、自らの生産性とデリバリーの能力についての情報を提供する
- 一貫したプロジェクトの推進を促す

1.2 アジャイルアプローチの特徴

組織で使われているアジャイルアプローチはいくつもある。その中でも、ほとんどのアジャイル組織における共通のプラクティスは、最終のリリースを含めた各イテレーションに対して、ユーザストーリーの共同作成、ふりかえり、継続的インテグレーションおよび計画である。次の項では、アジャイルアプローチのいくつかを説明する。

1.2.1 アジャイルソフトウェア開発のアプローチ

アジャイルソフトウェア開発アプローチにはいくつかの種類があり、それぞれがアジャイルマニフェストの価値と原則を異なる方法で実現する。このシラバスでは、代表的な三つのアジャイルアプローチ、エクストリームプログラミング（XP）、スクラムおよびカンバンについて説明する。

エクストリームプログラミング

エクストリームプログラミング（XP）は、元々Kent Beck [Beck04]により考案された、明確化された価値、原則および開発プラクティスにより特徴付けられるソフトウェア開発のアジャイルアプローチである。

XP では、開発を手助けするために 5 つの価値（コミュニケーション、シンプリシティ、フィードバック、勇気およびリスペクト）を重視する。

XP では追加のガイドラインとして一連の原則を示している。それらは、「人間性、経済性、相互利益、自己相似性、改善、多様性、ふりかえり、流れ、機会、冗長性、失敗、品質、ベイビーステップおよび責任の引き受け」である。

XP は 13 の主要なプラクティスを示している。それらは、全員同席、チーム全体、情報満載のワークスペース、生き生きとした仕事、ペアプログラミング、ストーリー、週次サイクル、四半期サイクル、ゆとり、10 分ビルド、継続的インテグレーション、テストファーストプログラミングおよびインクリメンタルな設計である。

XP およびその価値と原則は、現在使用されているアジャイルソフトウェア開発アプローチの多くに、影響をおよぼしている。たとえば、スクラムを採用するアジャイルチームは、しばしば XP のプラクティスを取り込んでいる。

スクラム

スクラムは、次の手法やプラクティスを構成要素とするアジャイルマネジメントフレームワークである[Schwaber01]。

- スプリント：スクラムはプロジェクトを、固定期間（通常 2～4 週間）のイテレーション（スプリントと呼ばれる）に分割する
- プロダクトインクリメント：各スプリントは、潜在的にリリース可能／出荷可能なプロダクト（インクリメントと呼ばれる）を生成する
- プロダクトバックログ：プロダクトオーナーは、計画済みのプロダクトアイテムの優先度付けされたリスト（プロダクトバックログと呼ばれる）をマネジメントする。プロダクトバックログは、スプリントが進むにつれて進化する（バックログのリファインメントと呼ばれる）
- スプリントバックログ：各スプリントの開始時に、スクラムチームはプロダクトバックログから最高の優先度を持つアイテムのセット（スプリントバックログと呼ばれる）を選択する。プロダクトオーナーではなくスクラムチームがスプリント内で具現化する必要のあるアイテムを選択するので、その選択は、プッシュ原則ではなくプル原則に基づくと言われる
- 完了（done）の定義：各スプリントの終了時に潜在的にリリース可能なプロダクトが確実に存在するようにするために、スクラムチームはスプリントが完了したと見なす適切な条件を討論して定義する。討論することにより、バックログアイテムとプロダクト要件に対するチームの理解が深まる
- タイムボックス化：チームがスプリント内で終了することを期待するタスク、要件、または機能のみが、スプリントバックログを構成する。開発チームは、スプリント内でタスクを終了できない場合、付随するプロダクトフィーチャをスプリントから削除し、タスクをプロダクトバックログに戻す。タイムボックス化は、タスクだけでなく、他の状況（ミーティングの開始時刻と終了時刻の強制など）にも適用される
- 透明性：開発チームは、デイリースクラムと呼ばれる日次ベースのミーティングにおいて、スプリントのステータスを報告し、更新する。これにより、現在のスプリントの内容と進捗（テスト結果を含む）が、チーム、マネジメント層およびすべての関係者に明らかになる。たとえば、開発チームは、スプリントのステータスをホワイトボードで示すことができる
-

スクラムは、次の三つの役割を定義する。

- スクラムマスター：　スクラムのプラクティスと規則が実行されて遵守されていることを確認し、チームが実践と規則を遵守することを妨げるあらゆる違反、リソースの問題、またはその他の障害を解決する。スクラムマスターは、チームリードではなくコーチである
- プロダクトオーナー：　顧客を代表し、プロダクトバックログの生成、保守および優先度付けを行う。プロダクトオーナーは、チームリードではない
- 開発チーム：　プロダクトを開発しテストする。開発チームは自己組織的である。チームリードは存在せず、チームで意思決定する。また、チームはクロスファンクショナルである（2.3.2 項と 3.1.4 項を参照）

スクラムは XP と異なり、特定のソフトウェア開発技法（テストファーストプログラミングなど）を強制しない。また、スクラムは、スクラムプロジェクトでテストを行う方法についてのガイダンスを規定しない。

カンバン

カンバン[Anderson13]は、アジャイルプロジェクトで使用されることもあるマネジメントアプローチである。一般的な目的は、付加価値連鎖の中で仕事の流れを可視化し、最適化することである。カンバンは次の三つの手法を使用する[Linz14]。

- カンバンボード：カンバンボードは、マネジメント対象の価値の連鎖を可視化する。各列はステーションを示し、それはたとえば開発やテストなどの関連するアクティビティのセットである。生成するアイテムまたは処理するタスクは、ステーションを通してボードを左から右へ移動するチケットによって表される
- 仕掛り（WIP）の制限：並行して実行するタスクの総数を厳密に制限する。これは、単一のステーションまたはボード全体で許容されるチケットの最大数に基づいて制御される。ステーションに空きがある場合は、作業者はチケットを前のステーションから移動させる
- リードタイム：（平均）リードタイムを最小化することによって、タスクの継続的フローを最適化し、理想的なバリューストリームとするためにカンバンは使われる

カンバンは、いくつかの点で、スクラムと類似している。どちらのフレームワークでも、アクティブなタスクを（たとえば人目に付く場所にあるホワイトボードなどで）可視化することにより、タスクの内容と進捗が明確になる。ステーションに余裕（生産能力）がある場合、バックログにあるスケジュールが立てられていないタスクを、速やかにカンバンボードへ移す。

カンバンにおいては、イテレーションまたはスプリントは必須ではない。カンバンプロセスでは、成果物をイテレーションまたはスプリントの単位ではなく、アイテム単位でリリースできる。このため、同期メカニズムとしてのタイムボックス化も、オプションである。これは、すべてのタスクをスプリント内で同期するスクラムとは異なる。

1.2.2　共同でのユーザストーリー作成

不十分な仕様は、多くの場合、プロジェクトを失敗させる重大な要因となる。仕様の問題は、真のニーズに対するユーザの洞察不足、システムに対する全体的な視点の欠如、無駄な、または矛盾するフィーチャ、およびその他の誤ったコミュニケーションにより発生する可能性がある。アジャイル開発において、ユーザストーリーは、開発者、テスト担当者およびビジネス代表者の視点から、要件を捉えるために記述される。シーケンシャル開発では、フィーチャのこの共有ビジョンは、要件が記述された後に公式レビューを通して達成される。アジャイル開発では、この共有ビジョンは、要件の記述が行われている最中に頻繁な非公式レビューを通して達成される。

ユーザストーリーは、機能面および非機能面の両方の特性に対処しなくてはならない。各ストーリーは、これらの特性の受け入れ基準を含む。ビジネス代表者、開発者およびテスト担当者は共同で、これらの基準を定義する必要がある。またユーザストーリーは、ビジネス代表者が確認するフィーチャまでおよぶビジョンを、開発者とテスト担当者にもたらす。アジャイルチームは、一連の受け入れ基準が満たされると、タスクが終了したと見なす。

一般的に、テスト担当者は特有の視点で詳細記述や非機能要件の不足を見つけ、ユーザストーリーを改善する。テスト担当者は、ビジネス代表者にユーザストーリーについての自由形式の質問を行い、ユーザストーリーをテストする方法を提案し、受け入れ基準を確認することによって貢献できる。

共同でユーザストーリーを作成する際には、ブレインストーミングやマインドマップなどの技法を使用できる。テスト担当者は、INVEST 技法を使える [INVEST]。

- 独立している（Independent）
- 調整可能である（Negotiable）
- 価値を提供する（Valuable）
- 見積り可能である（Estimable）
- 小さい（Small）
- テスト可能である（Testable）

3C コンセプト[Jeffries00]によれば、ユーザストーリーは、次の三つの要素で構成される。

- カード：カードはユーザストーリーを記述する物理メディアである。カードは、要件、その重要度、期待される開発とテストの期間およびそのストーリーの受け入れ基準を特定する。その記述はプロダクトバックログで使用されるので、正確である必要がある。

- 対話：対話は、ソフトウェアの使用方法を説明する。対話は、文書化されるか、または口頭で行われる。開発者やビジネス代表者とは異なる視点を持つテスト担当者は[ISTQB_FL_SYL] 考えや意見および経験をやりとりする際に有益な貢献をする。対話は、リリース計画フェーズで開始し、ストーリーがスケジュールされるときも継続する
- 確認：対話の中で討論される受け入れ基準は、ストーリーが完了（done）したことを確認するために使用される。これらの受け入れ基準は、複数のユーザストーリーに関連することがある。基準を網羅するためには、正常系テストと異常系テストの両方を用いなくてはならない。確認の期間では、さまざまな参加者がテスト担当者の役割を果たす。これらの参加者には、開発者に加えて、性能、セキュリティ、相互運用性および他の品質特性を重点的に担当する専門家も含まれる。ストーリーが完了したことを確認するために、定義済みの受け入れ基準をテストし、それらが満たされていることを示す必要がある

ユーザストーリーをどのように文書化するのかは、アジャイルチームによって異なる。ユーザストーリーを文書化するために採用されるアプローチに関係なく、ドキュメントは簡潔かつ必要十分でなければならない。

1.2.3　ふりかえり

アジャイル開発において、ふりかえりとは各イテレーションの最後に行われるミーティングである。何が成功で、何が改善可能で、その改善を将来のイテレーションにいかに取り込んで成功させるのかを議論する。ふりかえりでは、プロセス、人、組織、人間関係、ツールなどがトピックとなる。適切なフォローアップ活動を行うためには、ふりかえりミーティングを定期的に開催することが、開発およびテストの自己組織化と継続的改善にとって必須である。

ふりかえりは、結果として、テスト効率、テスト生産性、テストケース品質、チームの満足に重点を置いて、テスト関連の改善を決定することがある。また、アプリケーション、ユーザストーリー、フィーチャ、またはシステムインターフェースの試験性に対処することがある。欠陥の根本原因分析は、テストと開発の改善を促進できる。一般的に、チームはイテレーションごとに 2、3 の改善に止めるべきである。そうすれば、継続的な改善を一定のペースで持続できる。

ふりかえりの開催時期と体制は、用いられたアジャイルの手法次第である。ファシリテーターはふりかえりを計画、運営し、ビジネス代表者とチームは参加者として出席する。場合によっては、チームは他の参加者をミーティングに招待することがある。

テスト担当者は、ふりかえりで重要な役割を担う。テスト担当者はチームのメンバであり、ユニークな視点を提供する（[ISTQB_FL_SYL]、1.5 節）。テストは各スプリントで行われ、成功にとって極めて重要である。すべてのチームメンバ（テスト担当者もテスト担当者以外も）は、テストとテスト以外の両方の活動に対して貢献できる。

ふりかえりは、信頼関係が築かれているプロフェッショナルな環境であれば自然に発生する。ふりかえりを成功させるための条件は、『Foundation Level シラバス』[ISTQB_FL_SYL]の 3.2 節で説明されているレビューの場合と同じである。

1.2.4　継続的インテグレーション

プロダクトインクリメントをデリバリーするには、各スプリントの終了時に、信頼性が高く、動作可能で、統合されたソフトウェアが必要である。継続的インテグレーションは、ソフトウェアに対して行われたすべての変更をマージし、すべての変更済みコンポーネントを少なくとも 1 日に 1 回、定期的に統合することによって、この課題に対処する。構成管理、コンパイル、ソフトウェアビルド、デプロイおよびテストは、単一の自動化された繰り返し可能なプロセスにまとめることができる。それにより、開発者は定常的に成果物を統合し、ビルドし、テストするので、コード内の欠陥はより素早く検出される。

継続的インテグレーションプロセスは、開発者のコーディング、デバッグおよび共有ソースコードリポジトリへのチェックインに続く、以下のような自動化された活動からなる。

- 静的コード解析：静的コード解析を実行し、結果を報告する

- コンパイル：コードのコンパイルとリンクを行い、実行可能ファイルを生成する
- ユニットテスト：ユニットテストを実行し、コードカバレッジをチェックして、テスト結果を報告する
- デプロイ：ビルドをテスト環境にインストールする
- 統合テスト：統合テストを実行し、結果を報告する
- 報告（ダッシュボード）：これらすべての活動のステータスを閲覧可能なところに公開するか、チームにステータスを電子メールで送信する

自動化されたビルドとテストプロセスは基本的に毎日実行され、統合後のエラーを早期かつ素早く検出する。継続的インテグレーションは、アジャイルテスト担当者が自動テストを定期的に実行するのを可能にする。継続的インテグレーションプロセスそのものの一部として自動テストを実行することもでき、チームにコード品質に関する素早いフィードバックをもたらすことが可能になる。これらのテスト結果は、特に自動化された報告処理がプロセスに統合されている場合、すべてのチームメンバが見ることができる。自動回帰テストは、イテレーション全体を通して継続できる。良い自動回帰テストは、以前のイテレーションでデリバリーされたユーザストーリーを含む、可能な限り多くの機能をカバーする。適切なカバレッジで回帰テストを自動化すると、大規模な統合システムの構築（およびテスト）が容易になる。回帰テストを自動化すると、アジャイルテスト担当者は、新しいフィーチャと実装された変更に対する手動テストおよび欠陥の修正の確認テストに集中できるようになる。

自動テストに加えて、継続的インテグレーションを行っている組織は、継続的に品質管理を実施するため、一般的に、ビルドツールを使用する。ユニットテストと統合テストの実行だけでなく、そのようなツールを使用することで、その他の静的および動的テストの実行、性能の測定とプロファイル化、ソースコードからのドキュメントの抽出と整形および手作業による品質保証プロセスを容易にすることができる。このように品質管理を継続的に適用する目的は、すべての開発が完了した後に品質管理を適用する従来のプラクティスを置き換えることで、デリバリーにかかる時間を短縮するだけでなく、プロダクト品質を向上させることである。

ビルドツールは自動デプロイツールと連動させることができる、それにより、適切なビルドを継続的インテグレーションまたはビルドサーバから取得して、一つ以上の開発、テスト、ステージング、さらには本番の各環境にデプロイできる。これにより、各環境にリリースをインストールための専任のスタッフやプログラマに依存することに関連するエラーや遅延を減らすことができる。

継続的インテグレーションは、次の利点をもたらす。

- 統合の問題や競合する変更がより早く検出できるようになり、根本原因分析もより容易になる
- コードが機能しているかどうかに関するフィードバックを、開発チームに定期的に提供する
- 1日に開発されたソフトウェアのバージョンを、テストされた状態に維持する
- 小規模な変更セットを適用するごとにコードベースを迅速に再テストするので、開発者によるコードのリファクタリングに付随する回帰リスクを軽減する
- 日ごとの開発作業が確固たる基盤に基づいていることを確信させる
- プロダクトインクリメントの完成に向けて進捗を可視化し、開発者とテスト担当者の励みとなる
- ビッグバン統合に付随するスケジュールリスクを軽減する
- テスト、デモンストレーション、または教育の目的で、実行可能ソフトウェアがスプリント全体を通して常に利用できる
- 手動テストの繰り返しを削減できる
- 品質とテストを改善するために行われた判断に対するフィードバックを迅速に提供できる

ただし、継続的インテグレーションには、リスクと課題がないわけではない。

- 継続的インテグレーションツールを導入して保守する必要がある
- 継続的インテグレーションプロセスを定義して確立する必要がある
- テスト自動化には追加リソースを必要とし、確立するには複雑になる可能性がある
- 自動テストの利点を得るために、テストカバレッジを徹底して確保することが不可欠である
- チームがユニットテストに依存しすぎて、システムテストと受け入れテストをほとんど実施しないことがある

継続的インテグレーションでは、テストのためのツール、ビルドプロセスを自動化するためのツールおよびバージョンコントロールのためのツールなど、さまざまなツールを使用する必要がある。

1.2.5 リリースとイテレーション計画

『Foundation Level シラバス』[ISTQB_FL_SYL]で説明しているように、計画は継続的な活動であり、アジャイルライフサイクルの場合も同様である。アジャイルライフサイクルでは、リリース計画とイテレーション計画の二つの計画が発生する。

リリース計画は、プロダクトのリリースに備えるもので、多くの場合プロジェクトの開始から数カ月後となる。リリース計画では、プロダクトバックログを定義および再定義し、それには、大きなユーザストーリーを小さなストーリーの集まりに洗練することが含まれる。リリース計画は、すべてのイテレーションにまたがるテストアプローチとテスト計画の基準となる。リリース計画は、概要レベルで作成する。

リリース計画では、ビジネス代表者がチームと協調して、リリース用のユーザストーリーを確定し、優先度を割り当てる（1.2.2 項を参照）。これらのユーザストーリーに基づいて、プロジェクトリスクと品質リスクを識別し、概要レベルの工数見積りを行う（3.2 節を参照）。

テスト担当者はリリース計画に参加し、特に次の活動に寄与する。
- 受け入れ基準を含むテスト可能なユーザストーリーを定義する
- プロジェクトリスクと品質リスクの分析に参加する
- ユーザストーリーに付随するテスト活動を見積もる
- 必要なテストレベルを定義する
- リリースに向けてテストを計画する

リリース計画が完了したら、最初のイテレーションに向けてイテレーション計画を始める。イテレーション計画は、イテレーションバックログに関連し、単一のイテレーションの終わりを見据えて行う。

イテレーション計画では、チームは優先度が割り当てられたリリースバックログからいくつかのユーザストーリーを選択し、それらのユーザストーリーを精査し、リスク分析を行い、それぞれのユーザストーリーで必要な活動を見積もる。ユーザストーリーが非常に曖昧で、それを明確化する試みにも失敗した場合、チームはそのユーザストーリーの受け入れを拒否し、優先度に基づいて次のユーザストーリーを取り扱う。各ストーリーについて、何を実装し、どのようにテストするのかチームが理解できるように、ビジネス担当者は各ストーリーに関するチームの質問に回答しなくてはならない。

選択するストーリーの数は、チームの確立されたベロシティと、選択したユーザストーリーの見積り規模に基づいて決定する。イテレーションの内容を最終的に決定したら、ユーザストーリーは、タスクにブレークダウンされ、適切なチームメンバによって実行される。

テスト担当者はイテレーション計画に参加し、特に次の活動に寄与する。
- ユーザストーリーの詳細なリスク分析に関与する
- ユーザストーリーの試験性を決定する
- ユーザストーリーの受け入れテストを作成する
- ユーザストーリーをタスク（特にテスト）にブレークダウンする
- すべてのテストのテスト活動を見積もる
- システムの機能的および非機能的側面のうち、テストすべきものを識別する
- 複数のテストレベルにわたってテスト自動化に関与し支援する

リリース計画はプロジェクトの進捗に従って、プロダクトバックログにおける個々のユーザストーリーへの変更も含めて、変更される可能性がある。これらの変更は、内外の要因により発生する。内部要因としては、デリバリー能力、ベロシティ、テクニカルな問題などがある。外部要因としては、リリースの目的およびリリース日程の変更を発生させる可能性のある新しい市場や機会、新しい競合他社、ビジネス脅威の発見などがある。さらに、イテレーション計画はイテレーションの間に変更される可能性がある。たとえば、見積時に比較的単純であると見なされた特定のユーザストーリーが、想定よりも複雑であることが判明する場合がある。

これらの変更は、テスト担当者にとって大きな課題である。『Foundation Level シラバス』[ISTQB_FL_SYL] 1.4

節で説明しているように、テスト担当者は、テスト計画作業のためにリリースの概要を把握する必要があり、テスト開発のために各イテレーションの適切なテストベースとテストオラクルを掌握する必要がある。必要な情報はテスト担当者が早期に利用できなければならない。それでも、変更はアジャイル原則に従い取り込まなくてはならない。このジレンマにより、テスト戦略とテストドキュメントの作成については注意深く判断を行う必要がある。アジャイルテストの課題の詳細については、[Black09]第 12 章を参照されたい。

リリース計画とイテレーション計画では、開発活動の計画と同じようにテスト計画作業にも取り組まなくてはならない。次に示すように、テストに関連する特有の事項がある。

- テストのスコープ、スコープ内の各エリアに対するテストの深さ、テストのゴールおよびこれらの判断の理由
- テスト活動を実行するチームメンバ
- 必要なテスト環境とテストデータ、これらが必要になる時期、プロジェクトの開始前または期間中にテスト環境またはテストデータに対する追加もしくは変更が発生するかどうか
- 機能および非機能のテスト活動のタイミング、順序付け、依存関係および前提条件（たとえば、回帰テストを実行する頻度、フィーチャと他のフィーチャまたはテストデータとの依存関係など）。これには、テスト活動が開発活動にどのように関連または依存するかも含まれる
- プロジェクトリスクおよび品質リスクへの対応（3.2.1 項を参照）

さらに、より大きなチームの見積りには、要求されたテスト活動の完了のために必要な期間と工数を、より考慮しなければならない。

2. アジャイルテストの基本的な原則、プラクティスおよびプロセス － 105 分

用語

ビルド検証テスト、構成アイテム、構成管理

基本的なアジャイルテストの原則、プラクティスおよびプロセスの学習の目的

2.1 従来のテストとアジャイルアプローチのテストの違い

FA-2.1.1 (K2)アジャイルプロジェクトと非アジャイルプロジェクトとの間のテスト活動の違いを説明する。

FA-2.1.2 (K2)アジャイルプロジェクトでは開発とテストの活動がどのように統合されているのかを説明する。

FA-2.1.3 (K2)アジャイルプロジェクトでの独立したテストの役割を説明する。

2.2 アジャイルプロジェクトでのテストのステータス

FA-2.2.1 (K2)アジャイルプロジェクトで、テストの進捗やプロダクト品質を含めて、テストのステータスを伝達するために使用するツールと技法を説明する。

FA-2.2.2 (K2)複数のイテレーション間でテストが進化するプロセスを説明し、アジャイルプロジェクトで回帰リスクをマネジメントするためにテスト自動化が重要である理由を説明する。

2.3 アジャイルチームにおけるテスト担当者の役割とスキル

FA-2.3.1 (K2)アジャイルチームにおけるテスト担当者のスキル(対人、専門領域およびテストに関するスキル)を理解する。

FA-2.3.2 (K2)アジャイルチームにおけるテスト担当者の役割を理解する。

2.1 テストにおける従来型アプローチとアジャイルアプローチの違い

『Foundation Level シラバス』[ISTQB_FL_SYL]および[Black09]で説明しているように、テスト活動は開発活動と関連しており、このため、テストはライフサイクルごとにさまざまである。テスト担当者は、効果的かつ効率的に作業するために、従来型のライフサイクルモデル（例：V 字モデルなどのシーケンシャルモデルや RUP などのイテレーティブモデル）とアジャイルライフサイクルとの違いを理解する必要がある。アジャイルモデルは、テストと開発の活動の統合方法、プロジェクトの成果物、成果物の名称、さまざまなテストレベルで使用される開始基準と終了基準、ツールの使用、および独立したテストの効果的な活用方法の点で、従来のモデルと異なる。

テスト担当者は、組織によってそのライフサイクルの実施が大きく異なることを念頭に置く必要がある。理想的なアジャイルライフサイクル（1.1 節参照）から逸脱していたとしても、プラクティスを賢明にカスタマイズして適用した結果かもしれない。（実際に行われているソフトウェア開発への適応を含む）特定のプロジェクト状況における適応能力が、テスト担当者にとって成功するための鍵である。

2.1.1 テスト活動と開発活動

アジャイルライフサイクルの従来型ライフサイクルとの主な違いの一つは、極めて短いイテレーションという考え方であり、イテレーションごとにビジネスステークホルダに価値あるフィーチャをデリバリーする、動くソフトウェアが作成されることである。プロジェクトの開始直後は、リリース計画の期間である。続いて、一連のイテレーションが実行される。各イテレーションの開始直後は、イテレーション計画の期間である。イテレーションのスコープが確立されると、選択されたユーザストーリーが作成され、システムに統合後にテストされる。これらのイテレーションは極めて動的であり、開発、統合およびテストの各活動が、少なからず並行・重複して各イテレーション全体を通して実行される。テスト活動は、最後の活動としてではなく、イテレーション全体を通して行われる。

テスト担当者、開発者およびビジネスステークホルダは皆、従来型のライフサイクルと同様にテストでの役割がある。開発者は、ユーザストーリーからフィーチャを開発しながら、ユニットテストを実行する。続いてテスト担当者は、それらのフィーチャをテストする。ビジネスステークホルダも開発中にストーリーをテストする。ビジネスステークホルダは、ドキュメント化されたテストケースを使用することもあるが、開発チームに迅速にフィードバックを提供するために、単純にフィーチャを試してみたり、使用したりすることもある。

場合によっては、残存する欠陥や他の形態の技術的負債を解決するため、硬化（ハードニング）イテレーションまたは安定化（スタビライゼーション）イテレーションを定期的に実行する。しかし、ベストプラクティスは、いずれのフィーチャも、システムに統合してテストするまでは完了したと見なさないことである[Goucher09]。その他の良いプラクティスとして、前のイテレーションで解決されていない欠陥を次のイテレーションの開始時にそのイテレーションのバックログとして対処することがある（「バグフィックスファースト」と呼ばれる）。ただし、この方法では、イテレーションで完了（done）させなければならない作業量の合計が不明となり、残りのフィーチャを完了できる時期を見積もることがさらに困難になるという批判がある。一連のイテレーションの終了時に、デリバリーの準備が整ったソフトウェアを得られるので、まとめてリリース活動を行うことができるが、場合によっては各イテレーションの終了時にデリバリーを行う。

リスクベースドテストをテスト戦略の一つとして使用する場合、リリース計画時に概要レベルのリスク分析を行う。この際、テスト担当者が分析を主導することが多い。ただし、各イテレーションに付随する特定の品質リスクは、イテレーション計画で識別し、評価する。このリスク分析は、開発の順序と、フィーチャのテストの優先度と詳細度に影響をおよぼす可能性がある。また、各フィーチャで必要なテスト活動の見積りにも影響をおよぼす。（3.2 節参照）。

一部のアジャイルのプラクティス（エクストリームプログラミングなど）では、ペアリングを使用する。ペアリングでは、2 人のテスト担当者が 1 組となって一緒にフィーチャをテストする。また、1 人のテスト担当者と 1 人の開発者が共同でフィーチャの開発とテストを行うこともある。ペアリングは、テストチームが分散している場合には困難になることもあるが、プロセスとツールをうまく使うことにより、分散型のペアリングが可能になることもある。分散テストの詳細については、[ISTQB_ALTM_SYL] 2.8 節を参照されたい。

テスト担当者は、チーム内でテストと品質のコーチの役割を担い、チーム内でテストの知識を共有し、品質保証活動をサポートすることもある。こうすることで、プロダクト品質に関する共同責任の意識を高める。

多くのアジャイルチームでは、すべてのテストレベルでテスト自動化を行う。このことは、テスト担当者が自動テストとその結果について、作成、実行、監視および保守に時間を割いていることを意味する。アジャイルプロジェクトでは、テスト自動化を多用するため、手動テストについてはソフトウェア攻撃、探索的テスト、エラー推測など、経験ベースおよび欠陥ベースの技法を使用して行うことが多い（[ISTQB_ALTA_SYL] 3.3 節と 3.4 節および [ISTQB_FL_SYL] 4.5 節を参照）。開発者がユニットテストの作成に専念する一方で、テスト担当者は自動化された統合テスト、システムテストおよびシステム統合テストの作成に専念する必要がある。このため、アジャイルチームでは、高い技能を持つとともにテスト自動化の経験があるテスト担当者が好まれる傾向にある。

アジャイルの核となる原則の一つは、変更がプロジェクト全体を通して発生することである。このため、アジャイルプロジェクトでは、成果物のドキュメント化を軽量にすることを推奨する。既存のフィーチャに対する変更は、テスト、特に回帰テストに影響をおよぼす。自動テストの使用は、変更に付随するテスト活動の総量をマネジメントするための一つの方法である。しかし、変更の頻度が、変更によるリスクに対処するプロジェクトチームの能力を超えないことが重要である。

2.1.2 プロジェクト成果物

アジャイルテスト担当者に直接関連するプロジェクト成果物は、一般的に次の三つに分類される。

1. ビジネス指向の成果物：必要なものおよびその使用方法を記述する成果物（それぞれ要求仕様、ユーザドキュメントなど）
2. 開発成果物：システムの構築方法を記述する成果物、実際のシステムの実装、またはコードの個々の部品を評価する成果物（それぞれ、データベースエンティティ関係図、コード、自動化されたユニットテストなど）
3. テスト成果物：システムのテスト方法を記述する成果物、システムを実際にテストする成果物、またはテスト結果を示す成果物（それぞれテスト戦略と計画、手動テストおよび自動テスト、2.2.1 項で説明されるテストダッシュボードなど）

アジャイルプロジェクトでは、一般的に大量のドキュメントを生成することはなく、代わりに、要件に従っていることを示す自動テストと、動くソフトウェアに重点を置く。ドキュメントを削減するというこの奨励策は、顧客に価値をデリバリーしないドキュメントにのみ適用される。アジャイルプロジェクトを成功させるには、ドキュメントを減らして効率を上げることと、ビジネス、テスト、開発およびメンテナンスの各活動をサポートするためのドキュメントを十分に提供することを、うまく両立させる必要がある。チームはリリース計画時に、必要となる成果物と、そのドキュメント化の要求レベルを判断する必要がある。

アジャイルプロジェクトにおけるビジネス指向の成果物は、一般的に、ユーザストーリーと受け入れ基準を含む。ユーザストーリーは要求仕様のアジャイル版であり、単独で首尾一貫したフィーチャまたは機能についてのシステムの振る舞いを説明する。ユーザストーリーではフィーチャを、単一のイテレーションで完了する小規模なものとして規定する。関連するフィーチャのより大きな集まり、または単一の複雑なフィーチャを構成するサブフィーチャの集まりは、「エピック」と呼ばれる。エピックは、異なる開発チームのユーザストーリーを包含することもある。たとえば、あるユーザストーリーにおいて API レベル（ミドルウェア）で必要なものとして説明される一方、別のユーザストーリーでは、UI レベル（アプリケーション）で必要なものとされる。これらの集合体は、複数のスプリントにまたがって開発されることもある。各エピックとそのユーザストーリーには、受け入れ基準が関連づけられているべきである。

アジャイルプロジェクトの開発者が作成する典型的な成果物には、コードが含まれる。また、アジャイル開発者は、多くの場合、自動ユニットテストも作成する。これらのテストは、コードの開発後に作成することも可能である。ただし、場合によっては、開発者は、コードの部分ごとに、コードを記述する前にテストを作成する。これは、コードのその部分を記述した際に、それが期待どおりに動作するかどうかを検証する方法を提供するためである。このア

プローチはテストファースト開発またはテスト駆動開発とも呼ばれるが、それらのテストは、実際には単なるテストではなく、実行可能な詳細レベルの設計仕様の形態に近いものである[Beck02]。

アジャイルプロジェクトの一般的なテスト担当者の成果物は、自動テストに加えて、テスト計画、品質リスクカタログ、手動テスト、欠陥レポート、テスト結果ログなどのドキュメントを含む。ドキュメントは、可能な限り軽量な方法で作成する。これは、多くの場合、従来型のライフサイクルでのドキュメントについても当てはまる。また、テスト担当者は、欠陥レポートおよびテスト結果ログからテストメトリクスも取得するが、この場合も、軽量化のアプローチに重点が置かれる。

特に、法規制がある、安全性が強く求められる、分散されている、または非常に複雑であるプロジェクトおよびプロダクトを対象とするアジャイル開発では、これらの成果物をさらに形式化する必要がある。たとえば、一部のチームは、ユーザストーリーと受け入れ基準をより形式化された要求仕様に書き換える。垂直および水平方向のトレーサビリティレポートを、監査、規制および他の要件を満たすために準備することがある。

2.1.3 テストレベル

テストレベルとは、論理的に関連性があるテスト活動であり、通常、テストしているアイテムの成熟度または完成度で関連づけられている。

シーケンシャルライフサイクルモデルのテストレベルは、しばしば、あるレベルの終了基準が次のレベルにおける開始基準の一部であるというように定義する。一部のイテレーションモデルには、このルールは当てはまらない。テストレベルは重なり合う。要求仕様、設計仕様および開発活動が、テストレベルと重なり合うこともある。

一部のアジャイルライフサイクルでは、要件、設計およびコードに対する変更がイテレーションの任意のポイントで発生する可能性があるので、作業の重なり合いが発生する。スクラムは、理論的にはイテレーション計画後でのユーザストーリーに対する変更を許可しないが、実際には、そのような変更は起きることがある。一般的には、イテレーション時に特定のユーザストーリーが、次のテスト活動を通して順次確認される。

- ユニットテスト：一般的に開発者が実行する
- フィーチャ受け入れテスト：次の二つの活動に分割される場合がある
  - フィーチャ検証テスト：開発者またはテスト担当者が、ユーザストーリーの受け入れ基準に関する自動テストを実行する場合が多い
  - フィーチャ妥当性確認テスト：一般的に、開発者、テスト担当者、ビジネスステークホルダが協調して手動でテストする。フィーチャが利用に適しているかどうかを確認し、進捗の可視性を改善し、ビジネスステークホルダから実際のフィードバックを受け取る

さらに、多くの場合、イテレーション全体を通して、回帰テストプロセスを並行して実行する。これは、一般的には現在のイテレーションと以前のイテレーションで実行した自動ユニットテストとフィーチャ検証テストを、継続的インテグレーションフレームワークで再実行することである。

一部のアジャイルテストでは、システムテストレベルが存在する。このテストは、最初のユーザストーリーがテストできる状況になったときに開始する。このテストでは、機能テストだけでなく、性能、信頼性、使用性および他の関連するテストタイプの非機能テストを実行する。

アジャイルチームは、さまざまな形式の受け入れテストを採用できる（『Foundation Level シラバス』[ISTQB_FL_SYL]で説明されている用語を使用）。内部によるアルファテストと外部によるベータテストを、各イテレーションの最終段階、各イテレーションの完了後、または一連のイテレーションの後に、実施することがある。また、ユーザ受け入れテスト、運用受け入れテスト、規定による受け入れテストおよび契約による受け入れテストも、各イテレーションの最終段階、各イテレーションの完了後、または一連のイテレーションの後に、実施することがある。

2.1.4　テストと構成管理

アジャイルプロジェクトでは、多くの場合、自動化ツールを多用して、開発、テストおよびソフトウェア開発のマネジメントを行う。開発者は、静的解析、ユニットテストおよびコードカバレッジの計測のためにツールを使用する。開発者は、自動化されたビルドとテストのフレームワークを使用して、コードとユニットテストを継続的に構成管理システムにチェックインする。これらのフレームワークを使用することで、新しいソフトウェアをシステムに継続的インテグレーションすることができる。また、新しいソフトウェアをチェックインするごとに、静的解析とユニットテストを繰り返し実行できる[Kubaczkowski]。

さらに、これらの自動テストは、統合レベルおよびシステムレベルで機能テストを含むこともできる。そのような自動化された機能テストは、機能テストハーネス、オープンソースのユーザインターフェース機能テストツール、または市販ツールを使用して作成でき、継続的インテグレーションフレームワークの一部として実行する自動テストに統合できる。機能テストの期間に応じて、機能テストをユニットテストから分離し、より少ない頻度で実行することもある。たとえば、新しいソフトウェアをチェックインするごとにユニットテストを実行し、時間のかかる機能テストは、数日ごとに実行することもある。

自動テストの目的の一つは、ビルドが機能しており、インストールできることを確認することである。いずれかの自動テストが失敗すると、チームは次のコードチェックインに間に合うように原因である欠陥を修正する必要がある。このためには、リアルタイムのテストレポーティングの仕組みを構築し、テスト結果を適切に視認できるようにする必要がある。このアプローチでは、ビルドを破壊したりソフトウェアをインストールできなくしたりする変更をすぐに検出できるので、従来型のプロジェクトで多数発生する高コストで非効率な「ビルド、インストール、失敗、再ビルド、再インストール」のサイクルを容易に削減できる。

自動テストとビルドのツールは、アジャイルプロジェクトで頻繁に発生する変更に付随する回帰リスクをマネジメントするのに役立つ。ただし、これらのリスクをマネジメントするために自動ユニットテストのみに依存しすぎることは、問題になる可能性がある。これは、ユニットテストには十分な欠陥検出能力がないためである[Jones11]。統合レベルとシステムレベルでの自動テストも必要である。

2.1.5　独立したテストに関する組織上の選択肢

『Foundation Level シラバス』[ISTQB_FL_SYL]で説明しているように、独立したテスト担当者は、一般的に、欠陥を効果的に見つける能力が高い。一部のアジャイルチームでは、開発者が自動テストの形態で多くのテストを作成する。それらのチームには 1 人以上のテスト担当者が存在し、多くのテストのタスクを実行する。ただし、これらのテスト担当者のチーム内における立場を考えると、独立性と評価の客観性を失うリスクが存在する。

別のアプローチでは、アジャイルチームは完全に独立かつ分離したテストチームを持ち、各スプリントの終盤にオンデマンドでテスト担当者を割り当てる。この場合、独立性が維持され、テスト担当者はソフトウェアについて客観的かつ偏りのない評価を行うことができる。ただし、時間のプレッシャー、プロダクト内の新しいフィーチャの理解不足およびビジネスステークホルダや開発者との人間関係の行き詰まりにより、このアプローチでは問題が発生することがある。

三つ目のアプローチは、独立かつ分離したテストチームを持ち、そのチームに属するテスト担当者に、プロジェクトの開始から長期にわたってアジャイルチームを担当させる。このアプローチでは、テスト担当者は独立性を維持し、プロダクトを十分に理解し、他のチームメンバと強固な関係を築くことができる。さらに、独立したテストチームは、専門能力のあるテスト担当者をアジャイルチーム外に長期に配置して、自動テストツールの開発、非機能テストの実行、テスト環境とテストデータの作成とサポート、スプリントとして実施することがないテストレベル（たとえばシステム統合テスト）の実行など、イテレーションに依存しない活動に従事させることができる。

2.2 アジャイルプロジェクトでのテストステータス

アジャイルプロジェクトでは、変更が急に発生する。このような変更は、テストステータス、テスト進捗およびプロダクト品質が定常的に変化することを意味し、テスト担当者は、それらの情報をチームに伝達する方法を考案して、チームが各イテレーションの正常な完了に向けて順調に作業を進められるようにする必要がある。また、変更は以前のイテレーションで作成した既存のフィーチャに影響をおよぼすことがある。このため、手動テストおよび自動テストを更新して、回帰リスクに効果的に対処する必要がある。

2.2.1 テストステータス、進捗およびプロダクト品質についてのコミュニケーション

アジャイルチームは、各イテレーションの終了時に、動くソフトウェアを手にすることで進捗する。チームは、動くソフトウェアを手に入れる時期を確認するために、イテレーションとリリース内のすべての作業アイテムの進捗を監視する必要がある。アジャイルチームのテスト担当者は、さまざまな方法を使用してテストの進捗とステータスを記録する必要がある。これには、自動テストの結果、アジャイルタスクボードでのテストタスクとストーリーの進捗およびチームの進捗を示すバーンダウンチャート を含む。これらの情報は、Wiki ダッシュボードやダッシュボードスタイルの電子メールなどのメディア、またはスタンドアップミーティングでの発言により、チームの他のメンバへ伝達する。アジャイルチームは、テスト結果およびタスク進捗に基づいてステータスレポートを自動的に生成するツールを使用できる。そして、Wiki スタイルのダッシュボードと電子メールを順番に更新する。このコミュニケーション方法は、テストプロセスからメトリクスを収集し、プロセス改善に役立てることもできる。そのような自動化された方法を使用してテストステータスを伝達すると、テスト担当者の時間を節約し、テスト担当者はより多くのテストケースの設計と実行に重点を置けるようになる。

チームはバーンダウンチャートを使用してリリース全体と各イテレーション内の進捗を追跡できる。バーンダウンチャート[Crispin08]は、リリースまたはイテレーションに割り当てられている時間に対する、完了すべき残作業の量を表す。

チーム全体の現在のステータス（テストのステータスを含む）を一目で詳細に分かるように表現するために、チームはアジャイルタスクボードを使用できる。イテレーション計画（1.2.5 項を参照）で作成したストーリーカード、開発タスク、テストタスクおよび他のタスクのステータスをタスクボードに掲載する。タスクボードでは、多くの場合、色分けしたカードを使用してタスクの種類を判別する。イテレーション時には、これらのタスクをタスクボードの未着手（to do）、仕掛り（WIP）、検証（verify）、完了（done）などの列に移動させて、進捗をマネジメントする。アジャイルチームはツールを使用して、ストーリーカードとアジャイルタスクボードをメンテナンスする。そして、ダッシュボードとステータスの更新を自動化できる。

タスクボード上のテストタスクは、ユーザストーリー用に定義した受け入れ基準に関連する。テストタスクの自動テストスクリプト、手動テストおよび探索的テストが終了基準に到達すると、タスクはタスクボードの完了（done）列に移動する。チーム全員がタスクボードのステータスを定期的（日々のスタンドアップミーティング時など）にレビューし、タスクが意図した速度でボードを移動していることを確認する。いずれかのタスク（テストタスクを含む）が移動していないか、移動速度が遅い場合、チームは、それらのタスクの進捗を妨げている可能性のあるあらゆる問題を調査し対処する。

日々のスタンドアップミーティングには、テスト担当者を含む、アジャイルチームの全員が参加する。このミーティングで、現在の状況を周知する。各メンバは次のアイテムを報告する[Agile Alliance Guide]。

- 前回のミーティング以降に完了したもの
- 次回のミーティングまでに完了する予定のもの
- 進捗の妨げとなるもの

進捗を妨げる可能性のある、あらゆる問題を日々のスタンドアップミーティングの中で話題に挙げ、チーム全員が問題を認識し、状況に応じて解決する。

全体的なプロダクト品質を改善するために、多くのアジャイルチームは顧客満足度調査を行い、プロダクトが顧

客の期待を満たしているかどうかに関するフィードバックを受け取る。チームは、従来型の開発方法論同様、その他のメトリクスも収集して、プロダクト品質を改善する。これらのメトリクスには、テストの合格／不合格の割合、欠陥検出率、再テストと回帰テストの結果、欠陥密度、検出され修正された欠陥、要件カバレッジ、リスクカバレッジ、コードカバレッジ、コードチャーン（バージョン間で追加、変更、または削除されるコード行）などがある。

あらゆるライフサイクルと同様に、取得しレポートするメトリクスは、意思決定と関連性があり、意思決定を支援する必要がある。メトリクスは、チームメンバに対する報償、懲罰、チームから外す手段として使用してはならない。

2.2.2　手動テストケースと自動テストケースの増加に伴う回帰リスクのマネジメント

アジャイルプロジェクトでは、イテレーションが終了するごとに、プロダクトが成長する。このため、テストスコープも広がる。テスト担当者は、現在のイテレーションで追加されたコード変更をテストするだけでなく、以前のイテレーションで開発されテストされたフィーチャがデグレードしていないことを検証する必要がある。アジャイル開発では、コードチャーン（バージョン間で追加、変更、または削除されるコード行）が多く発生するので、回帰が取り込まれるリスクが高くなる。変更に対応することはアジャイル原則の核心であるので、ビジネスニーズを満たすために、以前にデリバリーされたフィーチャに対して変更が行われることもある。大量の技術的負債を招くことなくベロシティを維持するため、可能な限り早期に、すべてのテストレベルでテスト自動化に取り組むことが、チームにとっては不可欠である。自動テスト、手動テストケース、テストデータ、他のテスト成果物など、すべてのテスト資産をイテレーションごとに最新にしておくことは極めて重要である。バージョンコントロールを有効にするとともに、すべてのチームメンバが簡単にアクセスできるようにし、テスト資産の履歴情報を保つとともに、機能の変更により必要になったテスト資産の変更の支援するために、すべてのテスト資産を構成管理ツールで維持することを強く推奨する。

特に時間的な制約が厳しいアジャイルプロジェクトではすべてのテストを完全に繰り返すことはほとんど不可能であるので、テスト担当者は各イテレーションで、以前および現在のイテレーションの手動テストケースおよび自動テストケースをレビューし、回帰テストスイート用の候補を選択し、関連性のなくなったテストケースを削除する時間を確保する必要がある。フィーチャは変化したり、新しいフィーチャが以前のフィーチャの振る舞いを変更したりするので、特定のフィーチャを検証するために以前のイテレーションで記述されたテストは、以降のイテレーションでほとんど価値を持たなくなるかもしれない。

テストケースをレビューする際には、テスト担当者は自動化に適しているかどうかを考慮する必要がある。チームは、以前および現在のイテレーションのテストを可能な限り多く自動化する必要がある。これにより、自動回帰テストを通して、手動回帰テストが必要とするよりも少ない工数で回帰リスクを軽減できる。また、回帰テストの工数が削減されることから、テスト担当者は、現在のイテレーションの新しいフィーチャや機能をより徹底してテストできるようになる。

現在のイテレーションで行われた変更の影響を受ける、以前のイテレーションおよびリリースのテストケースを迅速に識別し、更新する能力をテスト担当者が持つことは、極めて重要である。チームがテストケースを設計、記述および格納する方法を定義することは、リリース計画で行う必要がある。テストの設計と実装のための優れた方法を、早期に採用し、一貫して適用する必要がある。各イテレーションでテストに十分な時間を割り当てないまま、定常的に変更が発生すると、不十分なテスト設計と実装による影響が増加する。

すべてのテストレベルで自動テストを行うことにより、アジャイルチームはプロダクト品質に関するフィードバックを迅速に提供できる。適切に記述された自動テストは、システムの機能性に関する有用なドキュメントを提供する[Crispin08]。自動テストと対応するテスト結果を構成管理システムにチェックインし、プロダクトビルドのバージョンコントロールと整合をとることにより、アジャイルチームは、特定の時点における特定のビルドに関してテストした機能性とテスト結果をレビューできる。

自動ユニットテストは、コード変更がソフトウェアビルドを破壊しないように、ソースコードを構成管理システムのメインラインにチェックインする前に実行する。チーム全体の進捗を遅延させる可能性のあるビルド破壊を軽減するために、すべての自動テストをパスしたコードをチェックインする必要がある。自動ユニットテストの結果は、コー

ドとビルドの品質についてのフィードバックを即座に提供するが、プロダクト品質については提供しない。

自動受け入れテストを、継続的インテグレーションのフルシステムにおけるビルドの一環として定期的に実行する。これらのテストは、完全なシステムに対して 1 日に 1 回以上実行するが、一般的に、コードチェックインごとには実行しない。これらのテストの実行は自動ユニットテストよりも時間がかかり、コードチェックインを遅延させる可能性があるためである。自動受け入れテストのテスト結果は、最後のビルド以降の回帰に関するプロダクト品質のフィードバックを提供するが、全体的なプロダクト品質の状況については提供しない。

自動テストは、システムに対して継続的に実行できる。重要なシステムの機能と結合箇所をカバーする自動テストの最初のサブセットは、新しいビルドをテスト環境にデプロイした直後に作成する必要がある。これらのテストは、一般的に、ビルド検証テストと呼ぶ。ビルド検証テストの結果は、ソフトウェアに関するフィードバックをデプロイ直後に提供するので、チームは、不安定なビルドをテストして時間を浪費することはない。

回帰テストセットに含まれる自動テストは、一般的に、継続的インテグレーション環境で日々のメインビルドの一環として実行し、新しいビルドをテスト環境にデプロイした際に再び実行する。自動回帰テストが失敗するとすぐに、チームは作業を停止し、テスト失敗の理由を調査する。テストの失敗理由が現在のイテレーションの正当な機能変更である場合、新しい受け入れ基準を反映するようにテストおよびユーザストーリーを更新することが必要になる。また、変更に対処するために別のテストを作成した場合は、変更前の機能のテストは削除することが必要になる。ただし、テストが欠陥のために失敗した場合、チームは欠陥を修正してから新しいフィーチャに進むことが良い方法である。

テスト自動化に加えて、次のテストタスクも自動化できる。

- テストデータ生成
- システムへのテストデータのロード
- テスト環境へのビルドのデプロイ
- ベースラインへのテスト環境（データベースや Web サイトのデータファイルなど）の復元
- データ出力の比較

これらのタスクを自動化することにより、オーバーヘッドを軽減でき、チームは新しいフィーチャの開発とテストに時間をかけることができる。

2.3 アジャイルチームにおけるテスト担当者の役割とスキル

アジャイルチームでは、テスト担当者は他のすべてのチームメンバおよびビジネスステークホルダと密接に協調する必要がある。このことは、テスト担当者が持っておくべきスキル、アジャイルチーム内での活動に、多くの影響を与える。

2.3.1 アジャイルテスト担当者のスキル

アジャイルテスト担当者は、『Foundation Level シラバス』[ISTQB_FL_SYL]で説明しているすべてのスキルを習得しておく必要がある。これらのスキルに加えて、アジャイルチーム内においては、テスト自動化、テスト駆動開発、受け入れテスト駆動開発、ホワイトボックス、ブラックボックスおよび経験ベースのテストに十分な能力を持っている必要がある。

アジャイル方法論では、チームメンバ間およびチーム外のステークホルダとの協調、コミュニケーション、対話が不可欠である。よって、アジャイルチームのテスト担当者は、優れた人間関係の構築スキルを習得している必要がある。アジャイルチームのテスト担当者が持つべきスキルは以下のとおりである。

- チームメンバおよびステークホルダに対して、建設的かつソリューション指向で向き合う
- プロダクトについて、批判的で、品質指向の、懐疑的な思考を発揮する
- ステークホルダから情報を積極的に入手する（ドキュメント化された仕様に全面的に頼ることはしない）
- テスト結果、テスト進捗およびプロダクト品質を正確に評価して報告する

- テスト可能なユーザストーリー、特に受け入れ基準について定義するために、顧客の代表者やステークホルダと共に効果的に働くチーム内で協調し、プログラマおよび他のチームメンバとペアになって作業する
- テストケースの変更、追加、または改善を含めて、変更に迅速に対応する
- テスト担当者側での作業を計画し、準備する

人間関係の構築スキルを向上させることを含めた、継続的なスキル向上は、アジャイルチームのテスト担当者を含めて、すべてのテスト担当者にとって不可欠である。

2.3.2 アジャイルチームにおけるテスト担当者の役割

アジャイルチームにおけるテスト担当者の役割は、テストステータス、テスト進捗およびプロダクト品質だけでなく、プロセス品質に関してもフィードバックを生成し、提供する活動を含む。このシラバスの別の場所で説明している活動に加えて、以下を含む。

- テスト戦略を理解、実施および最新化する
- 適用可能なすべてのカバレッジの観点について、横断的にテストカバレッジを測定し報告する
- テストツールを適切に使用していることを確実にする
- テスト環境とテストデータを構成、使用およびマネジメントする
- 欠陥を報告し、チームと共に欠陥を解消する
- テストの視点から他のチームメンバをコーチする
- リリース計画時およびイテレーション計画時に適切なテストタスクをスケジュールしていることを確実にする
- 開発者およびビジネスステークホルダと積極的に協調して、要件を、特に試験性、整合性および完全性に関して明確にする
- チームのふりかえりに積極的に参加し、改善を提案および実施する

アジャイルチーム内では、各チームメンバはプロダクト品質に責任を持ち、テスト関連のタスクを実行する役割を果たす。

アジャイル組織は、テストに関する組織的なリスクに直面することがある。

- テスト担当者が、開発者とあまりに密接に作業して、テスト担当者としての心構えを忘れる
- テスト担当者が、チーム内での効率の悪い、効果のない、または品質の低いプラクティスについて容認するか、沈黙する
- テスト担当者が、時間制約のあるイテレーションで、発生する変更に対応するペースを維持できない

これらのリスクを軽減するため、組織は 2.1.5 項で取り上げた、独立性を維持するための選択肢を考慮すると良い。

3. アジャイルテストの方法、技法、およびツール － 480 分

用語

受け入れ基準、探索的テスト、性能テスト、プロダクトリスク、品質リスク、回帰テスト、テストアプローチ、テストチャータ、テスト見積り、テスト実行自動化、テスト戦略、テスト駆動開発、ユニットテストフレームワーク

アジャイルテストの方法、技法およびツールの学習の目的

3.1 アジャイルテストの方法

FA-3.1.1　(K1)テスト駆動開発、受け入れテスト駆動開発およびビヘイビア駆動開発の概念を想起する。
FA-3.1.2　(K1)テストピラミッドの概念を想起する。
FA-3.1.3　(K2)アジャイルテストの四象限およびそれらのテストレベルとテストタイプとの関係をまとめる。
FA-3.1.4　(K3)特定のアジャイルプロジェクトに対して、スクラムチームでテスト担当者の役割を実践する。

3.2 品質リスクの評価とテスト工数の見積り

FA-3.2.1　(K3)アジャイルプロジェクト内の品質リスクを評価する。
FA-3.2.2　(K3)イテレーションのコンテンツと品質リスクに基づいてテスト工数を見積もる。

3.3 アジャイルプロジェクトの技法

FA-3.3.1　(K3)テスト活動をサポートするために関連情報を理解する。
FA-3.3.2　(K2)テスト可能な受け入れ基準を定義する方法をビジネスステークホルダに説明する。
FA-3.3.3　(K3)特定のユーザストーリーで、受け入れテスト駆動開発のテストケースを記述する。
FA-3.3.4　(K3)機能および非機能の両方の振る舞いに関して、ブラックボックステスト設計技法を使用して、特定のユーザストーリーに基づくテストケースを記述する。
FA-3.3.5　(K3)アジャイルプロジェクトのテストをサポートするために、探索的テストを実行する。

3.4 アジャイルプロジェクトにおけるツール

FA-3.4.1　(K1)アジャイルプロジェクトの目的と活動に従って、テスト担当者が使用できるさまざまなツールについて想起する。

3.1 アジャイルテストの方法

品質の高いプロダクトを開発するために各開発プロジェクト（アジャイルまたはそれ以外）で規範にできるテストプラクティスはいくつかある。振る舞いを完全に表現することに先立ってテストを記述する、欠陥の予防、早期検出、除去に重点を置く、正しいテストタイプを正しいタイミングで正しいテストレベルとして実施していることを確認する、などである。アジャイルの実践者は、これらのプラクティスの早期導入を目指す。アジャイルプロジェクトのテスト担当者は、ライフサイクル全体を通してこれらのテストプラクティスを主導する重要な役割を担う。

3.1.1 テスト駆動開発、受け入れテスト駆動開発およびビヘイビア駆動開発

テスト駆動開発、受け入れテスト駆動開発およびビヘイビア駆動開発の三つの相互補完的な技法は、アジャイルチームがいくつかのテストレベルを通してテスト実施に使用する。これらの技法は、コーディングよりも前にテストを定義するという、テストの基本的な原則の例である（そして、早期のテストと QA 活動の利点を示している）。

テスト駆動開発

テスト駆動開発（TDD）は、自動テストケースをガイドとしてコードを開発する際に使われる。テスト駆動開発のプロセスは、次の手順で行う。

- プログラマが考えている、コードの最小単位に対して求められる機能を表すテストを追加する
- テストを実行する。まだコードが存在しないので、テストは失敗する
- コードを記述し、テストをパスするまで、短い間隔でテストを実行する
- テストをパスしたらコードをリファクタリングし、テストを再実行して、リファクタリングしたコードが引き続きパスすることを確認する
- コードの次の最小単位に対して、このプロセスを繰り返す。以前のテストは追加したテストと同様に実行する

記述したテストは主としてユニットレベルであり、コードに重点を置いている。ただし、テストは統合レベルまたはシステムレベルで記述することもある。テスト駆動開発は、エクストリームプログラミングを通して広く普及したが [Beck02] 他のアジャイル方法論や一部のシーケンシャルライフサイクルでも使用している。この方法を使用すると、開発者は明確に定義された期待結果に重点を置くことができる。テストは自動化して、継続的インテグレーションで使用する。

受け入れテスト駆動開発

受け入れテスト駆動開発[Adzic09]は、ユーザストーリーの作成時に受け入れ基準とテストを定義する（1.2.2 項を参照）。受け入れテスト駆動開発はチーム間の協調を前提としたアプローチである。ソフトウェアコンポーネントがどのように振る舞うべきか、およびこの振る舞いを確実にするために、開発者、テスト担当者およびビジネス代表者は何をすべきかを、すべてのステークホルダが理解できるようにする。受け入れテスト駆動開発のプロセスは、3.3.2 項で説明する。

受け入れテスト駆動開発では、回帰テスト用の再利用可能なテストを作成する。特定のツールが、継続的インテグレーションプロセス内で、そのようなテストを作成し実行することをサポートする。これらのツールは、アプリケーションのデータ層とサービス層に接続できるので、テストはシステムレベルまたは受け入れレベルで実行できる。受け入れテスト駆動開発では、フィーチャの振る舞いについて、欠陥を早期に解決し、妥当性を確認できる。また、フィーチャの受け入れ基準が満たされているかどうかを容易に確認できる。

ビヘイビア駆動開発

ビヘイビア駆動開発[Chelimsky10]では、開発者はソフトウェアの期待される振る舞いに基づいてコードをテストすることに重点を置くことができる。テストはソフトウェアの振る舞いに基づいて行うので、他のチームメンバやステークホルダは、一般的に、テストを容易に理解できる。

特定のビヘイビア駆動開発フレームワークを使用して、Given/When/Then 形式で受け入れ基準を定義できる。
Given は事前条件を、

When はイベントの発生を、
Then は確認すべき結果を表す。

ビヘイビア駆動開発フレームワークは、これらの要件を基に、開発者がテストケースを作成するために使用できるコードを生成する。ビヘイビア駆動開発では、開発者はテスト担当者を含む他のステークホルダと協調して、ビジネスニーズに焦点を当てた正確なユニットテストを定義できる。

3.1.2 テストピラミッド

ソフトウェアシステムは、さまざまなレベルでテストすることがある。典型的なテストレベルは、ピラミッドの底部から頂点になぞられる、ユニット、統合、システムおよび受け入れである（[ISTQB_FL_SYL] 2.2 節参照）。テストピラミッドは、低いレベル（ピラミッドの底部）ほど多くのテストが存在し、開発が上位のレベル（ピラミッドの頂点）に進むに従って、テストの件数が減少することを表す。通常、ユニットレベルと統合レベルのテストは自動化し、API ベースのツールを使用して作成する。システムレベルと受け入れレベルでは、GUI ベースのツールを使用して自動テストを作成する。テストピラミッドのコンセプトは、早期に QA とテストを開始するというテストの原則に基づく（欠陥をライフサイクルの中で可能な限り早期に削減する）。

3.1.3 アジャイルテストの四象限、テストレベルおよびテストタイプ

Brian Marick が定義したアジャイルテストの四象限[Crispin08]は、テストレベルをアジャイル方法論の適切なテストタイプに対応付ける。アジャイルテストの四象限のモデルとそのバリエーションは、すべての重要なテストタイプとテストレベルが開発ライフサイクルに含まれていることを確認するのに役立つ。このモデルは、開発者、テスト担当者およびビジネス代表者を含むすべてのステークホルダに対して、テストタイプを区別し説明する方法を提供する。

アジャイルテストの四象限では、テストを、ビジネス（ユーザ）、テクノロジ（開発者）の面に分ける。いくつかのテストは、アジャイルチームがソフトウェアの振る舞いを確認することをサポートする。その他のテストは、プロダクトの検証を行う。テストは、完全手動、完全自動、手動と自動の組み合わせ、またはツールに支援された手動テストのいずれかである。四象限の定義を次に示す。

- 第一象限（Q1）は、ユニットのレベルで、テクノロジ指向であり、開発者をサポートする。この象限はユニットテストを含む。これらのテストは、自動化し、継続的インテグレーションプロセスに含める必要がある
- 第二象限（Q2）は、システムのレベルで、ビジネス指向であり、プロダクトの振る舞いを確認する。この象限は、機能テスト、使用例、ストーリーテスト、ユーザエクスペリエンスのプロトタイプおよびシミュレーションを含む。これらのテストは、受け入れ基準をチェックするものであり、手動または自動で実行できる。それらは、ユーザストーリーの開発時に作成され、ひいてはユーザストーリーの品質を改善する。さらに、自動回帰テストスイートを作成するのに役立つ
- 第三象限（Q3）は、システムのレベルまたはユーザ受け入れのレベルで、ビジネス指向であり、現実的なシナリオやデータを使用してプロダクトを評価するテストを含む。この象限は、探索的テスト、シナリオ、プロセスフロー、使用性テスト、ユーザ受け入れテスト、アルファテストおよびベータテストを含む。これらのテストは、ほとんどが手動で実行され、ユーザ指向である
- 第四象限（Q4）は、システムのレベルまたは運用受け入れのレベルで、テクノロジ指向であり、プロダクトを評価するテストを含む。この象限は、性能、ロード、ストレス、拡張性、セキュリティ、保守性、メモリ管理、互換性と相互運用性、データ移行、インフラストラクチャおよび回復の各テストを含む。これらのほとんどのテストは、自動化の対象となる

イテレーションによっては、いずれかまたはすべての象限のテストが必須である。アジャイルテストの四象限は、静的テストではなく動的テストに適用する。

3.1.4 テスト担当者の役割

このシラバス全体を通して、アジャイルの方法と技法およびさまざまなアジャイルライフサイクルでのテスト担当者の役割について全般的に説明している。この項では、特にスクラムライフサイクル[Aalst13]に従っているプロジェクトにおけるテスト担当者の役割について説明する。

チームワーク

チームワークは、アジャイル開発での基本的な原則である。アジャイルでは、開発者、テスト担当者およびビジネス代表者が一緒に作業するチーム全体アプローチが重要である。次に、スクラムチームでの組織上および行動上のベストプラクティスを示す。

- クロスファンクショナル：各チームメンバは、それぞれ異なるスキルセットをチームに持ち込む。チームは、テスト戦略、テスト計画作業、テスト仕様書、テスト実行、テスト評価、テスト結果のレポーティングに関して一緒に作業する
- 自己組織化：チームが開発者のみで構成されることもあるが、2.1.5 項で説明しているように、1 人以上のテスト担当者が存在していることが望ましい
- 同一作業環境：テスト担当者は、開発者およびプロダクトオーナーと隣り合って同じ場所で作業する。
- 協調：テスト担当者は、チームメンバ、他のチーム、ステークホルダ、プロダクトオーナーおよびスクラムマスターと協調して作業する
- 裁量権の付与：設計とテストに関するテクニカルな判定は、必要に応じてプロダクトオーナーや他のチームと協調し、チーム（開発者、テスト担当者およびスクラムマスター）が全体として行う
- コミット：テスト担当者は、顧客およびユーザの期待やニーズに関するプロダクトの動作や特性を疑問視し、評価することを託される
- 透明性：開発とテストの進捗は、アジャイルタスクボードで確認できる（2.2.1 項を参照）
- 信頼：テスト担当者は、テストに対する戦略の実装と実行の信頼性を確保しなくてはならない。これに失敗すると、ステークホルダはテストの結果を信頼しなくなる。信頼を確保するには、テストプロセスの情報をステークホルダに頻繁に提供する
- フィードバックの受容：フィードバックはどのようなプロジェクト（特にアジャイルプロジェクト）にとっても、成功にかかせない重要な側面を持つ。フィードバックに基づくふりかえりにより、チームは成功および失敗から学ぶことができる
- レジリエント：テストは、アジャイルプロジェクトの他のすべての活動と同じように、変更に対応できる必要がある

これらのベストプラクティスを実践することにより、スクラムプロジェクトのテストが成功する確率が最大となる。

スプリントゼロ

スプリントゼロは、多くの準備活動を行うプロジェクトの最初のイテレーションである（1.2.5 項を参照）。テスト担当者は、このイテレーションで次の活動をチームと協調して実行する。

- プロジェクトの範囲（プロダクトバックログ）を識別する
- 最初のシステムアーキテクチャと概要レベルのプロトタイプを作成する
- 必要なツール（テストマネジメント、欠陥マネジメント、テスト自動化および継続的インテグレーション）を計画、入手およびインストールする
- すべてのテストレベルについて最初のテスト戦略を作成する。この中で、（他のトピックも含めて）テストスコープ、テクニカルリスク、テストタイプ（3.1.3 項を参照）およびカバレッジの目標を考慮する
- 最初の品質リスク分析を実行する（3.2.1 項を参照）
- テストメトリクスを定義して、テストプロセス、プロジェクトでのテストの進捗、およびプロダクト品質を測定する
- 「完了（done）」の定義を明確にする
- タスクボードを作成する（2.2.1 項を参照）
- システムを顧客にデリバリーする前に、テストを継続するか終了するかを明確にする

スプリントゼロでは、スプリント全体を通して、テストで何を達成しなければならないのか、それをテストでどのよう

に達成しなければならないのか、それらの方向性を設定する。

統合

アジャイルプロジェクトの目標は、顧客に価値を継続して（望ましくはスプリントごとに）デリバリーすることである。これを達成するために、設計とテストの両方で統合の戦略を考慮しなくてはならない。デリバリー済みの機能と特性についての継続的なテスト戦略を可能とするには、根底にある機能とフィーチャの間のすべての依存関係を識別することが重要である。

テスト計画作業

テストはすべてアジャイルチーム内で行われるので、テスト計画作業をリリース計画セッション中に開始し、各スプリント内で更新する必要がある。リリース用および各スプリント用のテスト計画作業では、1.2.5 項で説明している問題に対処する必要がある。

スプリント計画では、一連のタスクをタスクボードに貼り付け、各タスクには 1 日または 2 日の作業日を割り当てる。さらに、テスト時の問題がすべて追跡されテストの進捗が着実に進行するようにする。

アジャイルテストのベストプラクティス

スクラムチームのテスト担当者に役立つ多くのベストプラクティスのいくつかを次に示す。

- ペアリング： 2 人のチームメンバ（テスト担当者と開発者、2 人のテスト担当者、テスト担当者とプロダクトオーナーなど）が 1 台の端末の前に隣り合わせて座って、テストまたは他のスプリントタスクを実行する
- インクリメンタルなテスト設計：テストケースとチャータは、ユーザストーリーと他のテストベースを基に徐々に作成し、単純なテストから始まり、より複雑なテストへと進化する
- マインドマップ：マインドマップは、テスト時に有用なツールである[Crispin08]。たとえば、テスト担当者はマインドマップを使用して、実行するテストセッションの識別、テスト戦略の提示およびテストデータの説明を行うことができる

これらのプラクティスは、このシラバスおよび『Foundation Level シラバス』[ISTQB_FL_SYL]の第 4 章で説明しているプラクティスに加えて使用できるものである。

3.2 品質リスクの評価とテスト工数の見積り

アジャイルか従来型かによらず、プロジェクトにおけるテストの典型的な目的は、プロダクト品質の問題によるリスクを軽減し、リリース前に受け入れ可能なレベルにすることである。アジャイルプロジェクトのテスト担当者は、従来型プロジェクトで使用されるのと同じ種類の技法を使用して、品質リスク（またはプロダクトリスク）を特定し、関連するリスクレベルを評価し、それらのリスクを十分に軽減するために必要な工数を見積り、テストの設計、実装、実行を通してそれらのリスクの軽減を行うことができる。ただし、アジャイルプロジェクトでのイテレーション期間の短さおよび変更の発生率は、それらの技法の改良を必要とする。

3.2.1 アジャイルプロジェクトにおける品質リスクの評価

テストにおける多くの課題の一つは、テスト条件を適切に選択、割り当ておよび優先度付けすることである。これには、やるべきテスト作業の有効性と効率性が最適となるように、各テストを網羅するために割り当てる適切な工数を決定し、テストの順序付けをすることが含まれる。アジャイルチームのテスト担当者が使う、リスク識別、分析およびリスク軽減戦略は、実行するテストケースの許容できる数を決定するのに役立つ。ただし、相互に影響する多くの制約や変動要因により妥協を余儀なくされることもある。

リスクとは、有害なまたは望ましくない結果や事象が起きる可能性のことである。リスクのレベルは、リスクの発生する可能性とリスクの影響を評価して判定する。プロダクトの品質に大きな影響をおよぼすような潜在的な問題

を、品質リスクまたはプロダクトリスクと呼ぶ。プロジェクトの成功に大きな影響をおよぼすような潜在的な問題を、プロジェクトリスクまたは計画リスクと呼ぶ[Black07] [vanVeenendaal12]。

アジャイルプロジェクトでは、品質リスク分析を二つのタイミングで行う。

- リリース計画：リリースのフィーチャを把握しているビジネス代表者は、リスクについて高いレベルでの概要を規定し、テスト担当者を含むチーム全体は、リスク識別と評価を支援する
- イテレーション計画：チーム全体が品質リスクを識別し評価する

システムの品質リスクの例を次に示す。
- レポートでの不正な計算（正確性に関連する機能リスク）
- ユーザ入力に対する応答の遅れ（効率性と応答時間に関連する非機能リスク）
- 画面とフィールドの分かりにくさ（使用性と理解性に関連する非機能リスク）

前述しているように、イテレーションはイテレーション計画で始まり、イテレーション計画は見積もったタスクをタスクボードに掲示することで終了する。これらのタスクには、関係する品質リスクのレベルに一部基づいて、優先度を割り当てることができる。関係するリスクが高いタスクほど、早くに開始し、より多くのテスト工数を割り当てる。関係するリスクが低いタスクほど、遅くに開始し、より少ないテスト工数を割り当てる。

アジャイルプロジェクトの品質リスク分析プロセスをイテレーション計画時に実行する方法の例を、次に示す。
1. テスト担当者を含むアジャイルチームメンバを招集する
2. 現在のイテレーションのすべてのバックログアイテムを（タスクボードなどに）一覧化する
3. 各アイテムに関係する品質リスクを、すべての関連する品質特性を考慮して識別する
4. 識別した各リスクを評価する。ここでは、欠陥の影響と発生確率に基づいて、リスクの分類と、リスクレベルの決定という、二つの活動を行う
5. リスクレベルに応じて、テストの範囲を決定する
6. リスク、リスクレベルおよび関連する品質特性に基づいて、各リスクを軽減するための適切なテスト技法を選択する

テスト担当者は、テスト設計、実装および実行を通して、リスクを軽減する。この対象には、顧客、ユーザおよびステークホルダの満足度に影響するフィーチャ、振る舞い、品質特性およびその他の特性を総合的に含む。

プロジェクトの期間中、テストチームは、一連の品質リスク、または判明している品質リスクのリスクレベルを変更する可能性のある追加情報を、常に認識している必要がある。テストの修正をもたらす品質リスク分析の見直しを、定期的に行うべきである。見直しには、新しいリスクの識別、既存のリスクレベルの再評価およびリスク軽減活動の有効性の評価を含む。

また、品質リスクは、テスト実行開始前に軽減することが可能である。たとえば、リスク識別時にユーザストーリーに関する問題を検出した場合、プロジェクトチームは軽減戦略として、徹底的なユーザストーリーのレビューを実施できる。

3.2.2　コンテンツとリスクに基づくテスト工数の見積り

リリース計画時に、アジャイルチームは、リリースを完了するのに必要な工数を見積もる。見積りでは、テスト工数も同様に考慮する。アジャイルチームは一般的な見積り技法として、合意に基づく技法であるプランニングポーカーを使用する。プロダクトオーナーまたは顧客は、ユーザストーリーを見積り担当者に伝える。各見積り担当者は、フィボナッチ数列（つまり、0、1、2、3、5、8、13、21、34、55、89、…）に類似の値または他のお気に入りの数列（XS から XXL までのワイシャツのサイズなど）のカードデッキを用意する。値は、ストーリーポイント数、工数（日数）、またはチームが見積りで使用する他の単位の数値を表す。フィボナッチ数列が推奨される理由は、数列内の値が、ストーリーの規模に比例して不確実性が大きくなることを反映しているからである。見積りの値が大

きい場合は、一般的にストーリーが正しく理解されていないか、複数のより小さなストーリーに分割する必要があることを意味する。

見積り担当者はフィーチャを検討し、必要に応じてプロダクトオーナーに質問する。開発工数およびテスト工数の側面では、ストーリーの複雑度およびテストの範囲が、見積りに影響を与える。このため、プランニングポーカーセッションを開始する前に、バックログアイテムに、プロダクトオーナーが指定した優先度に加えて、リスクレベルも含めることを推奨する。見積り担当者間でフィーチャを十分に検討したら、各見積り担当者は自身の見積りを表すカードを非公開で選択する。次に選択したカードを全員が同時に公開する。すべての見積り担当者が同じ値を選択した場合、その値が見積りになる。異なる場合、見積り担当者間で見積りの相違を討論し、その後、合意に達するまでポーカーラウンドを繰り返す。この際、意見の一致またはルール（平均スコアまたは最高スコアの使用）の適用のいずれかの手段を採用することにより、ポーカーラウンドの回数を制限する。これらの討論により、プロダクトオーナーが要求したプロダクトバックログアイテムを完了するのに必要な工数を正確に見積もることができ、実施すべきことについてのチームとしての理解を深める助けになる[Cohn04]。

3.3 アジャイルプロジェクトの技法

従来型のプロジェクトに適用できる多くのテスト技法とテストレベルを、アジャイルプロジェクトにも適用できる。ただし、アジャイルプロジェクトには、テスト技法、用語およびドキュメントにいくつかの固有の検討事項やバリエーションがある。

3.3.1 テストの受け入れ基準、適切なカバレッジおよびその他の情報

アジャイルプロジェクトでは、プロジェクトの開始時に、初期の要件を、優先度が割り当てられたバックログのユーザストーリーとして大まかに記述する。初期の要件は短く、一般的に規定の書式に従う（1.2.2 項を参照）。使用性や性能などの非機能要件も重要であり、固有のユーザストーリーとして指定することも、他の機能のユーザストーリーにまとめることもできる。非機能要件は、[ISO25000]のような規定の書式や標準、または業界標準に従うことができる。

ユーザストーリーは、重要なテストベースとして機能する。次のアイテムもテストベースとなる可能性がある。

- 以前のプロジェクトでの経験
- システムの既存の機能、フィーチャおよび品質特性
- コード、アーキテクチャおよび設計
- ユーザプロファイル（コンテキスト、システム構成およびユーザの振る舞い）
- 既存および以前のプロジェクトの欠陥に関する情報
- 欠陥分類法での欠陥の分類
- 適用可能な標準（航空関連ソフトウェアでは[DO-178B]など）
- 品質リスク（3.2.1 項を参照）

各イテレーション時に、開発者はユーザストーリーに記述されている機能とフィーチャおよび関連する品質特性を実装するコードを作成する。このコードは、受け入れテストで検証および妥当性を確認する。テスト可能にするために、受け入れ基準では、必要に応じて次のトピックを考慮する[Wiegers13]。

- 機能的振る舞い：特定の構成でユーザアクションとして入力操作を行った場合に、外見上観測される振る舞い
- 品質特性：特定の振る舞いをシステムがどのように実行するのか。この特性は、品質の属性または非機能要件とも呼ばれる。一般的な品質特性は、性能、信頼性、使用性などである
- シナリオ（ユースケース）：特定の目標またはビジネスタスクを達成するための外部アクター（多くの場合、ユーザ）とシステムとの間の一連のアクション
- ビジネスルール：システム外の手順および制約で定義されている特定の条件の下でのみシステムで実

行できる活動（保険会社で使用される、保険金の支払い申請を処理する手順など）
- 外部インターフェース：開発対象のシステムとシステム外との間の接続の記述。外部インターフェースは、異なるタイプ（ユーザインターフェース、他のシステムへのインターフェースなど）に分類できる。
- 制約：開発者の選択肢を制限する設計および実装上のあらゆる制約。ソフトウェアが埋め込まれているデバイスは、多くの場合、サイズ、重量、インターフェース接続などの物理的制約を受けることがある
- データ定義：顧客は、複雑なビジネスデータ構造を構成するデータアイテムについて、書式、データの型、許可される値およびデフォルト値を記述することがある（郵便物の郵便番号など）

ユーザストーリーおよび付随する受け入れ基準に加えて、次の情報もテスト担当者に関連する。
- どのようにシステムが動き、使われているのか
- システムをテストするために使用／アクセスするシステムインターフェース
- 現在のツールでテストを十分に行えるかどうか
- 必要なテストを実行するための十分な知識とスキルをテスト担当者が習得しているかどうか

テスト担当者は、イテレーション全体を通して追加情報（コードカバレッジなど）が必要であることを発見した場合、その情報を入手するため、他のアジャイルチームメンバと協力しなければならない。関連する情報は、特定の活動を完了できるかどうかを決定する際に重要な役割を果たす。完了（done）の定義に関するこのコンセプトは、アジャイルプロジェクトで極めて重要であり、次に説明しているように、さまざまな方法で適用される。

テストレベル

各テストレベルには、完了（done）に関する独自の定義がある。次のリストは、さまざまなテストレベルに関係する例を示す。
- ユニットテスト
  - 可能な場合、100%のデシジョンカバレッジを確保し、実行不可能なパスがないことを入念にレビューする
  - すべてのコードでの静的解析を実行する
  - 未解決の重大な欠陥が存在しない（優先度と重大度に基づいてランク付けされる）
  - 設計とコードに既知の受け入れ不可能な技術的負債が存在しない[Jones11]
  - すべてのコード、ユニットテストおよびユニットテストの結果がレビュー済みである
  - すべてのユニットテストを自動化している
  - 重要な特性（性能など）が合意した許容範囲内にある
- 統合テスト
  - 機能要件について、正常系テストと異常系テストの両方がテスト済みである。これらは、規模、複雑度およびリスクから想定されたテストの数を満たす
  - ユニット間のすべてのインターフェースがテスト済みである
  - 合意したテストスコープに従ってすべての品質リスクがカバーされている
  - 未解決の重大な欠陥が存在しない（リスクと重要度に従って優先度付けされる）
  - 検出したすべての欠陥が報告済みである
  - 可能な限り、すべての回帰テストを自動化している。すべての自動テストを共通のリポジトリに格納している
- システムテスト
  - ユーザストーリー、フィーチャおよび機能のエンドツーエンドのテストである
  - すべてのユーザペルソナをカバーしている
  - システムの最も重要な品質特性（性能、頑健性、信頼性など）をカバーしている
  - サポートされる（可能な限り）すべてのハードウェアとソフトウェアの構成を含む、本番相当の環境でテストが完了している
  - 合意したテストスコープに従ってすべての品質リスクがテスト済みである
  - 可能な限り、すべての回帰テストを自動化している。すべての自動テストを共通のリポジトリに格納している
  - 検出したすべての欠陥が報告済みであり、可能な限り修正済みである
  - 未解決の重大な欠陥が存在しない（リスクと重要度に従って優先度付けされる）

ユーザストーリー

ユーザストーリーの完了（done）の定義は、次の基準で決定できる。

- イテレーション用に選択したユーザストーリーは作成完了しており、チームは十分に理解しており、詳細でテスト可能な受け入れ基準を持つ
- ユーザストーリー受け入れテストを含む、ユーザストーリーのすべての要素が特定されてレビュー済みで、完了している
- チームは、選択したユーザストーリーを実装およびテストするために必要なタスクを識別済みであり、見積り済みである

フィーチャ

フィーチャは、複数のユーザストーリーまたはエピックにまたがる可能性があり、その完了（done）の定義は、次のようになる。

- 構成されるすべてのユーザストーリーは受け入れ基準と共に、顧客によって定義済みかつ承認済みである
- 設計は完了しており、既知の技術的負債は存在しない
- コードは完成しており、既知の技術的負債または未終了のリファクタリングは存在しない
- ユニットテストは実行済みであり、定義したカバレッジのレベルに到達済みである
- フィーチャ用の統合テストおよびシステムテストは、定義したカバレッジ基準に従って実行済みである
- 修正が必要な重大な欠陥は残っていない
- フィーチャの文書化は完了している。それには、リリースノート、ユーザマニュアル、オンラインヘルプの機能を含む

イテレーション

イテレーションにおける完了（done）の定義は、次のアイテムを含む。

- イテレーション用のすべてのフィーチャは、フィーチャごとの基準に従って準備が整っており、個別にテスト済みである
- 重大ではなく、イテレーションの制約で修正できない欠陥は、プロダクトバックログに追加済みで、優先度が決定済みである
- イテレーション用のすべてのフィーチャの統合は完了しており、テスト済みである
- ドキュメントは、記述済み、レビュー済みおよび承認済みである

イテレーションが正常に完了したこの時点で、ソフトウェアは潜在的にリリース可能である。ただし、すべてのイテレーションでリリースが行われるということではない。

リリース

リリースは、複数のイテレーションにまたがる可能性があり、その完了（done）の定義には、次のエリアを含む。

- カバレッジ：リリース内容のすべてについて、関係するすべてのテストベースの要素が、テストによってカバーされている。カバレッジの適切性は、新規または変更されたもの、その複雑度と規模および関連する故障のリスクによって決定される
- 品質：欠陥の量（1日当たり、またはトランザクション当たりで見つかる欠陥数など）、欠陥の密度（ユーザストーリー、工数、または品質の属性の数に対する、発見された欠陥数など）、残りの欠陥の推定数が受け入れ可能な範囲内にあること、未解決または未対応の欠陥の影響（重大度、優先度など）が認識済みであり受け入れ可能であること、識別した各品質リスクの残存レベルが認識済みであり受け入れ可能であること
- 時間：決められたデリバリー日に到達した際に、リリースする場合とリリースしない場合のビジネス上の考慮事項を検討する必要がある
- コスト：見積り済みのライフサイクルコストを使用して、デリバリー済みのシステムの投資効果を計算する（計算により算出される開発コストおよびメンテナンスコストは、プロダクトの予定総売上額よりもはるかに少額である必要がある）。ライフサイクルコストの主要なコストは、プロダクトがリリースされた後のメンテナンスに起因する。これは、しばしば欠陥が見逃されてプロダクトに存在するためである。

3.3.2　受け入れテスト駆動開発の適用

受け入れテスト駆動開発はテストファーストアプローチである。ユーザストーリーを実装する前に、テストケースを作成する。テストケースを作成するのは、開発者、テスト担当者およびビジネス代表者を含むアジャイルチームである[Adzic09]。テストの実行は手動または自動のいずれかを選択できる。最初の手順は、仕様ワークショップであり、開発者、テスト担当者およびビジネス代表者が、ユーザストーリーの分析、検討、記述を行う。このプロセス内で、ユーザストーリー内のすべての不完全性、曖昧性、またはエラーを修正する。

次の手順では、テストを作成する。テストは、チームまたは個々のテスト担当者が作成する。どのような場合でも、ビジネス代表者などの独立した人がテストの妥当性を確認する。テストは、ユーザストーリーの具体的な特性を示す代表的な例である。これらの代表例は、チームがユーザストーリーを正しく実装するのに役立つ。ユーザストーリーの代表例とテストは同じであるので、これらの用語はしばしば区別なく使用する。テスト作成の作業は、基本的な代表例と代表例に対する自由回答式の質問で開始される。

一般的に、最初のテストは、正常系のテストであり、例外またはエラーの条件を与えず正しく振る舞うことを確認する。これらのテストは、すべてが期待どおりに進行した場合に実行される一連の作業で構成する。正常系のパスのテストが完了したら、チームは異常系のパスのテストを記述し、非機能属性（性能、使用性など）も対象にする。テストは、すべてのステークホルダが理解できるような方法で表現し、事前条件（存在する場合）、入力および関連する出力を自然言語の文章で記述する。

ユーザストーリーの代表例は、ユーザストーリーのすべての特性を対象にすべきであり、ストーリーを増やすべきでない。これは、ユーザストーリー自体に記述されていないユーザストーリーの一面を記述する代表例は存在してはいけないということを意味する。さらに、ユーザストーリーの同じ特性を記述する代表例が二つ以上存在してはならない。

3.3.3　機能および非機能のブラックボックステスト設計

アジャイルテストでは、テスト担当者は開発者のプログラミング活動と並行して、多くのテストを作成する。開発者がユーザストーリーと受け入れ基準に基づいてプログラミングするのとまったく同じく、テスト担当者もユーザストーリーと受け入れ基準に基づいてテストを作成する。（探索的テストや他の経験ベースのテストなど一部のテストは、3.3.4 項で説明するように、テスト実行の間に作成する。）テスト担当者は、従来型のブラックボックステスト設計技法を適用してこれらのテストを作成できる。適用できる技法には、同値分割法、境界値分析、デシジョンテーブル、状態遷移テストなどがある。たとえば、顧客が購入時に選択できるアイテム数に制限がある場合は、境界値分析を使用して、テストの値を選択できる。

多くの状況で、非機能要件をユーザストーリーとして文書化できる。ブラックボックステスト設計技法（境界値分析など）を使用して、非機能品質特性用のテストを作成することもできる。ユーザストーリーには、性能や信頼性の要件を含めることができる。たとえば、特定の処理は制限時間を超えてはならない、操作の失敗は特定の回数を超えてはならない、などである。

ブラックボックステスト設計技法を使用する方法については、『Foundation Level シラバス』[ISTQB_FL_SYL]および『Advanced Level シラバス　テストアナリスト』[ISTQB_ALTA_SYL]を参照されたい。

3.3.4　探索的テストとアジャイルテスト

アジャイルプロジェクトでは、テスト分析にかけられる時間も、ユーザストーリーの詳細度も制限されているので、探索的テストは極めて有用である。最良の結果を得るために、対処的テスト戦略の一環として、探索的テストを他の経験ベースの技法と組み合わせなければならない。また、分析的リスクベースドテスト、分析的要件ベースドテスト、モデルベースドテスト、回帰回避テストなどの、その他のテスト戦略と融合しなければならない。テスト戦略と

その融合の方法については、『Foundation Level シラバス』[ISTQB_FL_SYL]を参照されたい。

探索的テストでは、準備済みのテストチャータに従って、テスト設計とテスト実行を同時に行う。時間の制限が厳しいテストセッションの間、テストチャータは、カバーすべきテスト条件を提供する。探索的テストでは、直近のテストの結果が、次のテストの指針となる。事前に設計済みのテストを実行するときと同じく、ホワイトボックスやブラックボックスの技法を使用して、テストを設計できる。

テストチャータには、次の情報を含めることができる。

- アクター：システムを使用するユーザ
- 目的：チャータのテーマ。アクターが達成したい特定の目標（テスト条件）を含む
- セットアップ：テスト実行を開始するために準備が整っている必要のあるもの
- 優先度：このチャータの相対的な重要度。付随するユーザストーリーまたはリスクレベルの優先度に基づく
- 参照：仕様（ユーザストーリーなど）、リスク、または他の情報ソース
- データ：チャータを実行するために必要なあらゆるデータ
- 活動：アクターがシステムに対して行う必要のあること（「スーパーユーザとしてシステムにログオンする」など）と、何をテストしたいのか（正常系と異常系のテスト両方）に関するアイデアのリスト
- オラクルについての覚え書き：正しい結果であることを確認するためにプロダクトを評価する方法（画面に表示されるものをキャプチャしてユーザマニュアルに記述されていることと比較するなど）
- バリエーション：活動の項で説明しているアイデアを補完するために代替となるアクションと評価

探索的テストのマネジメントには、セッションベースのテストマネジメントと呼ばれる方法を使用できる。セッションとは、たとえば 60～120 分の間、中断されることのないテストの時間と定義される。テストセッションは次の項目を含む。

- 調査セッション（動作の仕組みを学習）
- 分析セッション（機能性または他の特性の評価）
- 深さのカバー（コーナーケース、シナリオ、相互作用）

テストの品質は、テスト担当者がテストすべきことに関連した疑問を持つ能力に依存する。たとえば、次のようなものである。

- システムについて把握しておく必要のある最も重要なことは何か?
- どのような場合にシステムは機能しなくなるのか?
- ～の場合は何が起きるか?
- ～のときには何が起きるべきか?
- 顧客のニーズ、要件および期待は満たされているか?
- システムは、サポートされるすべての更新方法でインストールできるか（および必要に応じて削除できるか）?

テスト担当者はテスト実行時に、想像力、直感、認識力およびスキルを使用して、プロダクトに内在する可能性のある問題を発見する。また、テスト対象のソフトウェア、ビジネスドメイン、ソフトウェアの使用方法およびシステムが失敗したかどうかの判定方法に関する優れた知識と理解が必要である。

テスト時には、一連のヒューリスティックを適用できる。ヒューリスティックは、テストの実行方法と結果の評価方法についてテスト担当者をガイドする [Hendrickson]。たとえば、次のようなものがある。

- 境界
- CRUD（作成（Create）、読み取り（Read）、更新（Update）、削除（Delete））
- 構成のバリエーション
- 中断（ログオフ、シャットダウン、再起動など）

テスト担当者は、プロセスについて可能な限り多くを文書化することが重要である。文書化が十分でないと、システムでどのように問題が発見されたかを遡って確認することが困難になる。文書化に役立つ情報には、次のようなものがある。

- テストカバレッジ：使用した入力データ、どれくらいテストしたか、どれくらいのテストが残っているか
- 評価の覚え書き：テスト時の状況（テスト対象のシステムおよびフィーチャは安定していたか、何らかの欠陥が見つかったか）、現在の状況に従って次に行おうとしているテストおよびその他のアイデアのリスト
- リスク／戦略リスト：回避されたリスクは何か、最も重要なリスクの中で未対処のリスクはどれか、初期の戦略にそのまま従うか、何らかの変更が必要か
- 問題、疑問および異常：予期しない振る舞い、アプローチの効率性に関する疑問、アイデア／テストの試行についての懸念、テスト環境、テストデータ、機能の誤解、テストスクリプトまたはテスト対象システム
- 実際の振る舞い：保存する必要のある、システムの実際の振る舞いの記録（動画、スクリーンキャプチャ、出力データファイルなど）

記録した情報は、キャプチャして何らかのステータスマネジメントツール（テストマネジメントツール、タスクマネジメントツール、タスクボードなど）の形式にまとめる。この際、実行したすべてのテストにおける現在のステータスをステークホルダが簡単に理解できる方法を採用する。

3.4 アジャイルプロジェクトにおけるツール

『Foundation Level シラバス』[ISTQB_FL_SYL]で説明しているツールは、アジャイルチームのテスト担当者の使用にも適切である。すべてのツールを同じように使うことはなく、一部のツールは従来型プロジェクトよりもアジャイルプロジェクトにより適切である。たとえば、アジャイルチームは、テストマネジメントツール、要件マネジメントツール、インシデントマネジメントツール（欠陥追跡ツール）を使用できるが、一部のアジャイルチームはすべての機能を包含するツール（アプリケーションライフサイクルマネジメントまたはタスクマネジメント）の使用を選択する。このツールは、タスクボード、バーンダウンチャート、ユーザストーリーなど、アジャイル開発に関連する機能を提供する。アジャイルチームのテスト担当者は、すべてのレベルで自動テストが大量に存在し、付随する自動テストの成果物を格納およびマネジメントしなければならないので、構成管理ツールを使用することが重要である。

『Foundation Level シラバス』[ISTQB_FL_SYL]で説明しているツールに加えて、アジャイルプロジェクトのテスト担当者は、次の項で説明しているツールも使用することがある。チーム全体でこれらのツールを使用して、アジャイルプラクティスの核心であるチーム内協調と情報共有を確実に行う。

3.4.1 タスクのマネジメントと追跡のツール

アジャイルチームは、場合によっては、物理的なストーリー／タスクボート（ホワイトボード、コルクボードなど）を使用して、ユーザストーリー、テストおよび他のタスクを、各スプリント全体を通してマネジメントおよび追跡する。また、電子的なタスクボードを含むアプリケーションライフサイクルマネジメントおよびタスクマネジメントのソフトウェアを使用することもある。これらのツールは、次のような目的で使用する。

- スプリント内で失われるものがないようにするために、ストーリーおよび関連する開発タスクとテストタスクを記録する
- イテレーション計画セッションを効率的に実施するために、タスクに関するチームメンバの見積りを取り込み、ストーリーを実現するために必要な工数を自動的に計算する。
- ストーリーを実現するために必要なチームの工数の全体像を把握するため、開発タスクとテストタスクをその同じストーリーに関連づける
- 開発者とテスト担当者が作業を完了した際に、両者の更新情報をタスクステータスにまとめる。その結果、各ストーリー、イテレーションおよび全体的なリリースのステータスに関する最新の集約されたスナップショットを自動的に提供する

- 各ユーザストーリー、イテレーションおよびリリースの現在の状態を、メトリクス、チャートおよびダッシュボードを介して視覚的に表現して、地理的に分散しているチームのメンバを含むすべてのステークホル

ダがステータスを迅速に把握できるようにする
- 構成管理ツールと統合して、タスクに対するコードチェックインとビルドを自動的に記録し、場合によっては、タスクのステータスを自動的に更新できるようにする

3.4.2 コミュニケーションツールと情報共有ツール

電子メール、ドキュメントおよび口頭によるコミュニケーションに加えて、多くの場合、アジャイルチームはさらに三種類のツール（Wiki、インスタントメッセージ、デスクトップ共有）を使用して、コミュニケーションと情報共有を行う。

チームは Wiki を使用して、次のようなプロジェクトのさまざまな側面に関するオンラインナレッジベースを構築して共有できる。
- プロダクトのフィーチャ図、フィーチャについての討論、プロトタイプの図表、ホワイトボードでの討論の写真およびその他の情報
- チームのメンバにより有益であることが判明した、開発およびテスト用のツールや技法
- プロダクトステータスに関するメトリクス、チャートおよびダッシュボード。Wiki をビルドサーバやタスクマネジメントシステムなどのその他のツールと統合すると、このツールはプロダクトステータスを自動的に更新できるので、これらの情報は特に有用となる
- チームメンバ間の会話。インスタントメッセージや電子メールに類似しているが、チーム内の全員で共有される方法で行う

インスタントメッセージ、電話会議および動画チャットツールは、次の利点を提供する。
- チームメンバ間、特に分散しているチームメンバ間で、リアルタイムかつ直接的なコミュニケーションを行うことができる
- 分散チームでスタンドアップミーティングを行うことができる
- VoIP（Voice-over-IP）テクノロジを使用して電話代を削減し、分散チームでチームメンバ間のコミュニケーション不足を助長させるコスト面での制約を取り払うことができる

デスクトップ共有ツールとキャプチャツールは次の利点を提供する。
- 分散チームで、プロダクトデモンストレーション、コードレビュー、さらにはペアでの作業を実施できる
- プロダクトデモンストレーションを各イテレーションの終了時にキャプチャして、チームの Wiki に投稿できる

これらのツールは、アジャイルチームにおける対面コミュニケーションの置き換えではなく、補完および強化に使われるべきである。

3.4.3 ソフトウェアビルドとディストリビューションツール

このシラバスで既に説明したように、アジャイルチームの核心となるプラクティスは、ソフトウェアのデイリービルドとデプロイである。このためには、継続的インテグレーションツールとビルドディストリビューションツールを使用する必要がある。これらのツールの使用方法、利点、リスクについては、1.2.4 項を参照されたい。

3.4.4 構成管理ツール

アジャイルチームでは構成管理ツールを使用して、ソースコードと自動テストを格納し、多くの場合、プロダクトソースコードと同じリポジトリに手動テストと他のテスト成果物も格納する。これにより、テストしたソフトウェアのバージョンとテストの特定バージョンの組み合わせを追跡でき、履歴情報を失うことなく迅速な変更が可能になる。バージョンコントロールシステムの主要な種類には、集中型と分散型がある。特定のアジャイルプロジェクトにとって適切なバージョンコントロールシステムを決定するには、チームの規模、体制、場所、他のツールと統合するための要件を考慮する。

3.4.5 テスト設計、実装および実行ツール

ソフトウェアテストプロセスの特定の時点で、アジャイルテスト担当者にとって役立ついくつかのツールがある。次に挙げるツールのほとんどは新しくもなくアジャイル専用でもないが、アジャイルプロジェクトでの迅速な変化にとって重要な能力を提供する。

- テスト設計ツール：新しいフィーチャ用のテストを迅速に設計および定義する際には、マインドマップなどのツールがますます使われるようになってきている
- テストケースマネジメントツール：アジャイルで使用するテストケースマネジメントツールの中には、チーム全体のアプリケーションライフサイクルマネジメントツールまたはタスクマネジメントツールの一部となっているものがある
- テストデータ準備および生成ツール：アプリケーションをテストする際に大量のデータおよびデータの組み合わせが必要な場合は、データを生成してアプリケーションのデータベースをセットアップするツールが極めて役立つ。また、アジャイルプロジェクトの期間中にプロダクトが変更になったためにデータベース構造を再定義し、データを生成するためのスクリプトをリファクタリングする場合も役立つ。このように、これらのツールを使用すると、変更が発生した場合に、テストデータを迅速に更新できる。一部のテストデータ準備ツールは、本番データソースを元データとして使用し、スクリプトを使用して機密データを除去したり匿名化したりする。大量のデータ入力または出力を検証するのを支援するテストデータ準備ツールもある
- テストデータロードツール：テスト用のデータを生成したら、それらをアプリケーションにロードする必要がある。手動でデータを入力すると時間がかかり、エラーも発生しやすいので、データロードツールを使用して、プロセスの信頼性と効率性を高める。実際、多くのデータ生成ツールに、データロードコンポーネントが統合されている。その他にも、データベース管理システムを使用して一括ローディングを行うこともできる
- 自動テスト実行ツール：アジャイルテストにさらに適したテスト実行ツールがある。ビヘイビア駆動開発、テスト駆動開発、受け入れテスト駆動開発などテストファーストアプローチをサポートする特定のツールを、市販およびオープンソースで入手できる。テスト担当者やビジネス担当者はこれらのツールを使用して、システムの期待される振る舞いを、キーワードを使用して表形式または自然言語で表現できる
- 探索的テストツール：テスト担当者や開発者は、探索的テストセッションの実行時にアプリケーションで実行したアクティビティをキャプチャおよび記録するツールを使用して、実行したアクションを記録できる。これらのツールは、故障が発生する前に行ったアクションをキャプチャし、開発者に欠陥を報告するために使用できるので、欠陥が見つかった際に役に立つ。探索的テストセッションで実行したステップを記録することは、テストが最終的に自動回帰テストスイートに取り込まれる場合に役立つ場合がある

3.4.6 クラウドコンピューティングと仮想化ツール

仮想化を採用すると、単一の物理リソース（サーバ）をより多く分割し、少ないリソースで動作させることができる。チームは仮想マシンまたはクラウドインスタンスを使用することにより、開発やテストにより多くのサーバを利用できる。このため、物理サーバが利用可能になるのを待つことにより生じる遅延を避けることができる。新しいサーバをプロビジョニングしたりサーバを復元したりする操作は、ほとんどの仮想化ツールに組み込まれているスナップショット機能を使用することで、さらに効率的に行うことができる。一部のテストマネジメントツールは、仮想化テクノロジを利用してフォールトが検出された時点でのサーバのスナップショットを取得できる。そのため、テスト担当者はフォールトを調査している開発者とスナップショットを共有できる。

4. 参考文献

4.1 標準

- [DO-178B] RTCA/FAA DO-178B, Software Considerations in Airborne Systems and Equipment Certification, 1992.
- [ISO25000] ISO/IEC 25000:2005, Software Engineering - Software Product Quality Requirements and Evaluation (SQuaRE), 2005.

4.2 ISTQB ドキュメント

- [ISTQB_ALTA_SYL]: ISTQB Advanced Level Test Analyst Syllabus, Version 2012、
JSTQB 訳注）日本では「テスト技術者資格制度 Advanced Level シラバス日本語版 テストアナリスト Version 2012.J01」として発行されている
- [ISTQB_ALTM_SYL]: ISTQB Advanced Level Test Manager Syllabus, Version 2012、
JSTQB 訳注）日本では「テスト技術者資格制度 Advanced Level シラバス日本語版 テストマネージャ Version 2012.J03」として発行されている
- [ISTQB_FA_OVIEW]: ISTQB Foundation Level Agile Tester Overview, Version 1.0
- [ISTQB_FL_SYL]: ISTQB Foundation Level Syllabus, Version 2011、
JSTQB 訳注）日本では「テスト技術者資格制度 Foundation Level シラバス日本語版 Version 2011.J02」として発行されている

4.3 書籍

[Aalst13]: Leo van der Aalst and Cecile Davis, "TMap NEXT® in Scrum," ICT-Books.com, 2013.

[Adzic09]: Gojko Adzic, "Bridging the Communication Gap: Specification by Example and Agile Acceptance Testing," Neuri Limited, 2009.

[Anderson13]: David Anderson, "Kanban: Successful Evolutionary Change for Your Technology Business," Blue Hole Press, 2010.
JSTQB 訳注）日本では「カンバン ソフトウェア開発の変革 Improving Service Delivery in Technology Business」（リックテレコム, 2014 年）として発行されている。

[Beck02]: Kent Beck, "Test-driven Development: By Example," Addison-Wesley Professional, 2002.
JSTQB 訳注）日本では「テスト駆動開発」（オーム社, 2017 年）として発行されている。

[Beck04]: Kent Beck and Cynthia Andres, "Extreme Programming Explained: Embrace Change, 2e" Addison-Wesley Professional, 2004.
JSTQB 訳注）日本では「エクストリームプログラミング」（オーム社, 2015 年）として発行されている。

[Black07]: Rex Black, "Pragmatic Software Testing," John Wiley and Sons, 2007.

[Black09]: Rex Black, "Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 3e," Wiley, 2009.
JSTQB 訳注）日本では"Managing the Testing Process: Practical Tools and Techniques for Managing Hardware and Software Testing, 2e,"が「基本から学ぶテストプロセス管理 コンピュータシステムのテストを成功させるために」（日経 BP 社, 2004 年）として発行されている。

[Chelimsky10]: David Chelimsky et al, "The RSpec Book: Behavior Driven Development with Rspec, Cucumber, and Friends," Pragmatic Bookshelf, 2010.
JSTQB 訳注）日本では「The RSpec Book」（翔泳社, 2012 年）として発行されている。

[Cohn04]: Mike Cohn, "User Stories Applied: For Agile Software Development," Addison-Wesley Professional, 2004.

[Crispin08]: Lisa Crispin and Janet Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams," Addison-Wesley Professional, 2008.
JSTQB 訳注）日本では「実践アジャイルテスト テスターとアジャイルチームのための実践ガイド」（翔泳社, 2009 年）として発行されている。

[Goucher09]：Adam Goucher and Tim Reilly, editors, "Beautiful Testing: Leading Professionals Reveal How They Improve Software," O'Reilly Media, 2009.
JSTQB 訳注）日本では「ビューティフルテスティング ソフトウェアテストの美しい実践」（オライリージャパン, 2010 年）として発行されている。
[Jeffries00]：Ron Jeffries, Ann Anderson, and Chet Hendrickson, "Extreme Programming Installed," Addison-Wesley Professional, 2000.
JSTQB 訳注）日本では「XP エクストリームプログラミング導入編 ― XP 実践の手引き（The XP Series）」（ピアソン・エデュケーション, 2001 年）として発行されている。
[Jones11]：Capers Jones and Olivier Bonsignour, "The Economics of Software Quality," Addison- Wesley Professional, 2011.
JSTQB 訳注）日本では「ソフトウェア品質の経済的側面」（共立出版, 2013 年）として発行されている。
[Linz14] Tilo Linz, "Testing in Scrum: A Guide for Software Quality Assurance in the Agile World," Rocky Nook, 2014.
[Schwaber01] Ken Schwaber and Mike Beedle, "Agile Software Development with Scrum," Prentice Hall, 2001.
JSTQB 訳注）日本では「アジャイルソフトウェア開発スクラム」（ピアソン・エデュケーション, 2003 年）として発行されている。
[vanVeenendaal12] Erik van Veenendaal, "The PRISMA approach", Uitgeverij Tutein Nolthenius, 2012.
[Wiegers13] Karl Wiegers and Joy Beatty, "Software Requirements, 3e," Microsoft Press, 2013.
JSTQB 訳注）日本では「ソフトウェア要求 第 3 版」（日経 BP 社, 2014 年）として発行されている。

4.4 アジャイル用語集

ISTQB 用語集に掲載されているキーワードを、各章の先頭に定義している。一般的なアジャイル用語については、それらの定義を提供している、広く認められているインターネットリソースに従っている。

http://guide.agilealliance.org/
http://whatis.techtargetcom/glossary
http://www.scrumalliance.org/

このドキュメントで使用しているアジャイル関係の用語が不明な場合は、これらのサイトを参照されたい。これらのリンクは、このドキュメントのリリース時点でアクティブであった。

4.5 その他の参照元

以下は、インターネットなどで参照できる情報を示している。これらの参照については、本シラバス発行時にチェックしているが、リファレンスが既に参照できなくなっていても、ISTQB はその責を負わない。

- [Agile Alliance Guide]：Various contributors, http://guide.agilealliance.org/.
- [Agilemanifesto]：Various contributors, www.agilemanifesto.org.
- [Hendrickson]：Elisabeth Hendrickson, "Acceptance Test-driven Development," testobsessed.com/2008/12/acceptance-test-driven-development-atdd-an-overview.
- [INVEST]：Bill Wake, "INVEST in Good Stories, and SMART Tasks," xp123.com/articles/invest- in-good-stories-and-smart-tasks.
- [Kubaczkowski]：Greg Kubaczkowski and Rex Black, "Mission Made Possible, " www.rbcs- us.com/images/documents/Mission-Made-Possible.pdf.

5. 索引

す

せ

そ

た

ち

て

と

は

ひ

ふ

へ

ま

ゆ

り