IT資格道場

JSTQB認定テスト技術者資格 Foundation Level(CTFL) 学習用テキスト

予備知識ゼロから合格ラインへ —— 読んで理解する1冊目

無料サンプル(第1章まで)

ISTQB テスト技術者資格制度 Foundation Level シラバス 日本語版 Version 2023V4.0.J02(2024年7月10日)準拠
2026年11月版

SAMPLE

© IT資格道場｜無料サンプル｜転載・再配布・AI学習利用禁止

ご利用にあたって

- このテキストは「IT資格道場」(www.shikaku-doujou.com)の購入者限定テキストのうち、学習用テキストの第1章までを収録した**無料サンプル**です。どなたでもご覧いただけます。
- 無料サンプルも著作物です。転載・改変しての再配布・二次販売はできません。
- AI モデルの学習データとしてのご利用(学習目的の入力・データセット化を含む)はできません。
- 製品版(学習用テキスト・頻出度別ノート・暗記用ノートの3点)は、ご購入者を識別できる透かし入りのPDFとして提供されます。
- 本PDFはスマートフォン・タブレットでの閲覧に合わせたA5版面・文字サイズで組版しています。

SAMPLE

目次

この試験と、この本の方針

この試験は何か

試験の基本情報

試験の設計図（この本の章立ての根拠）

第1章 テストの基礎 [1.1.1/1.1.2/1.2.1/1.2.2/1.2.3/1.3.1/1.4.1/1.4.2/1.4.3/1.4.4/1.4.5/1.5.1/1.5.2/1.5.3]

テストの全体像 [1.1.1]

テストとデバッグ [1.1.2]

テストが必要な理由 [1.2.1]

テストと品質保証 [1.2.2]

エラー、欠陥、故障と根本原因 [1.2.3]

テストの 7 原則 [1.3.1]

テスト活動とタスク [1.4.1]

コンテキストに応じたテストプロセス [1.4.2]

テストウェア [1.4.3]

テストベースとテストウェアのトレーサビリティ [1.4.4]

テストの役割 [1.4.5]

テストに必要な汎用的スキル [1.5.1]

チーム全体アプローチ [1.5.2]

テストの独立性 [1.5.3]

サンプルはここまでです

SAMPLE

3 冊の使い方は、ダウンロード欄の案内を参照してください。

📌 **見出し末尾の `[1.1.1]` `[4.2.1/4.2.2]` は、日本語シラバスの学習の目的（細目）番号**です。どの節がどの出題範囲に当たるかを、見出しだけで追えます。

SAMPLE

この試験と、この本の方針

この試験は何か

JSTQB認定テスト技術者資格 Foundation Level（CTFL）とは、ソフトウェアテストの入門資格です。[1]

- 実施：JSTQB（日本ソフトウェアテスト資格認定委員会）
- 受験資格：なし
- 言語：日本語で受験できる

この本の対象は、バージョン4.0（日本語シラバス 2023V4.0.J02）。試験は2024年11月1日から、このバージョンで実施されています。[2]

出題の軸は次の2つ。

1. **用語の意味を正しく言えるか**（例：エラー・欠陥・故障の違い、テストとデバッグの違い）
2. **場面と技法・考え方を結びつけられるか**（例：この入力にはどのテスト技法を使うか、この状況はどのレビュー種別か）

計算して答える問題が集まる第4章は、手順を覚えれば得点源になります。

試験の基本情報

項目	内容
問題数	**40問**（1問1点）
試験時間	**60分**
合格点	**40点中26点（65%）**

SAMPLE

項目	内容
受験料	**22,000円（税込）**
受験資格	**なし**
言語	**日本語**

[3]

試験の設計図（この本の章立ての根拠）

章ごとの出題数と、認知レベルの内訳。K1 は想起、K2 は理解して説明、K3 は手順を当てはめて解く水準を指します。[4]

章	出題数
第1章 テストの基礎	8問
第2章 ソフトウェア開発ライフサイクル全体を通してのテスト	6問
第3章 静的テスト	4問
第4章 テスト分析と設計	11問
第5章 テスト活動のマネジメント	9問
第6章 テストツール	2問

- 認知レベル別の問数：K1 が8問、K2 が24問、K3 が8問
- 第4章と第5章だけで40問のうち20問を占める。学習の入り口はこの2章。

SAMPLE

第1章 テストの基礎 [1.1.1/1.1.2/1.2.1/1.2.2/1.2.3/1.3.1/1.4.1/1.4.2/1.4.3/1.4.4/1.4.5/1.5.1/1.5.2/1.5.3]

この章は、試験 40 問のうち 8 問を占めます。出題の中心は用語の区別です。エラーと欠陥と故障、テストとデバッグ、品質コントロールと品質保証のように、似た言葉を取り違えないかが問われます。後半の活動やテストウェアは、場面を読んで「これはどの活動の仕事か」を当てる問題になります。

テストの全体像 [1.1.1]

ソフトウェアは日々の生活の土台になっています（業務システム、家電、スマートフォンなど）。期待どおりに動かなかった経験は誰にでもあります。正しく動かないソフトウェアは、経済的な損失、時間の浪費、信用の失墜を生み、極端な場合は傷害や死亡事故にもつながります。ソフトウェアテストは、品質を評価し、運用環境で故障が起きるリスクを下げる手助けをします。

テストとは、欠陥を発見し、ソフトウェアアーティファクトの品質を評価するための一連の活動です。評価の対象になる作業成果物を、テスト対象と呼びます（英語では test object）。

テストにはよくある誤解が 2 つあります。1 つ目は、テストはテスト実行だけだという誤解です。テスト実行とは、ソフトウェアを動かして結果を確かめる作業を指します。実際のテストには、計画、分析、設計、実装、完了など、他の活動も含まれます。しかもソフトウェア開発ライフサイクル（SDLC）に合わせて進める必要があります（第 2 章）。2 つ目は、テストは検証だけに集中するという誤解です。テストは検証に加えて、妥当性確認も行います。

SAMPLE

用語	確かめること	見る基準
検証（verification）	指定された要件をシステムが満たしているか	仕様書や要件
妥当性確認（validation）	ユーザーやその他のステークホルダーのニーズを、運用環境でシステムが満たしているか	利用者の期待

たとえば、仕様書どおりに作られたことの確認は検証です。実際の利用者が使ってみて困らないことの確認は妥当性確認です。仕様書そのものが利用者のニーズから外れていれば、検証が全部通っても妥当性確認は通りません。この差は 1.3 節の第 7 原則にそのままつながります。

テストはもう 1 つの軸でも二つに分かれます。動的テストはソフトウェアの実行を伴います。静的テストは実行を伴わず、レビューと静的解析を含みます。動的テストでは、テスト技法やテストアプローチを使ってテストケースを導出します（第 4 章）。静的テストは第 3 章で扱います。

テストは技術的な作業だけでもありません。計画、マネジメント、見積り、モニタリング、コントロールも必要です。テスト担当者はツールを使いますが、テストは大部分が知的活動です。専門知識を持ち、分析スキルを使い、批判的思考やシステム思考を働かせることが求められます。ISO/IEC/IEEE 29119-1 という標準が、ソフトウェアテストの概念をさらに詳しく説明しています。[5]

典型的なテスト目的は次の 9 つです。

- 要件、ユーザーストーリー、設計、コードなどの作業成果物を評価する。
- 故障を引き起こし、欠陥を発見する。
- 求められるテスト対象のカバレッジを確保する。

SAMPLE

- ソフトウェア品質が不十分な場合のリスクレベルを下げる。
- 仕様化した要件が満たされているかを検証する。
- テスト対象が契約、法律、規制の要件に適合していることを検証する。
- ステークホルダーに根拠ある判断をしてもらうための情報を提供する。
- テスト対象の品質に対する信頼を積み上げる。
- テスト対象が完成し、ステークホルダーの期待どおりに動作するかを妥当性確認する。

カバレッジとは、決めたカバレッジアイテムのうち、テストで実際に通したものの割合です（パーセントで表します）。何をカバレッジアイテムとして数えるかは第 4 章で出てきます。

テスト目的は固定ではありません。テスト対象のコンポーネントかシステムか、テストレベル、リスク、従う SDLC モデル、企業構造や競争上の考慮事項や市場投入までの時間といったビジネスコンテキストによって変わります。たとえば、契約で性能の達成が決まっている案件なら、契約への適合の検証が前面に出ます。市場投入を急ぐ製品なら、リスクレベルを下げる目的の比重が高まります。

選択式の問題では、目的の一覧に載っていない言い回しに注意してください。「開発担当者の評価を行う」「欠陥ゼロを保証する」はテスト目的ではありません。テストは欠陥のないことを保証しません。この理由は 1.3.1 の第 1 原則で説明します。

テストとデバッグ [1.1.2]

テストとデバッグは別の活動です。テストは、欠陥が引き起こす故障を発生させる活動（動的テスト）か、テスト対象の欠陥を直接見つける活動（静的テス

SAMPLE

ト）です。デバッグは、故障の原因である欠陥を見つけ、原因を解析し、取り除く活動です。

動的テストで故障が見つかった場合の流れは次のとおりです（3 段階）。

1. 故障を再現する。
2. 診断する。根本原因を見つける。
3. 原因を修正する。

修正後には確認テストを行い、問題が解決したかを調べます。確認テストは、最初のテストを行った人が行うのが望ましいです。修正が他の部分に新しい故障を生んでいないかを調べるために、続けてリグレッションテストを行うこともあります。確認テストとリグレッションテストの違いは 2.2.3 項で詳しく扱います。ここでは、確認テストは「直したか」を見る、リグレッションテストは「壊していないか」を見るテストで、2つは見る物が同じではありません。

場面	誰の仕事か	活動
テストケースを実行して画面のエラー表示を観察する	テスト担当者	テスト
ログを読んで、どの変数が不正値になったか調べる	開発担当者	デバッグ（診断）
不正値を出していた条件分岐を直す	開発担当者	デバッグ（修正）
同じ手順を再実行し、画面が正しくなったか見る	テスト担当者	確認テスト

静的テストで欠陥を見つけた場合は、デバッグとはその欠陥を取り除くことです。静的テストは欠陥を直接見つけるもので、故障を引き起こしません。だから

SAMPLE

再現も診断も要りません。「静的テストで見つけた欠陥のデバッグは、故障の再現から始める」は誤りです。

テストの仕事は故障の発生と欠陥の発見までで、原因の解析と修正はデバッグの仕事です。この線引きが、この項の出題の中心です。[6]

テストが必要な理由 [1.2.1]

テストは品質コントロールの一形式で、範囲・時間・品質・予算の制約の中で、合意されたゴールの達成を助けます。成功へのテストの貢献は、テストチームの活動だけに限りません。ステークホルダーなら誰でも、プロジェクトを成功に近づけるためにテストスキルを活かせます。コンポーネント、システム、関連するドキュメントのテストは、ソフトウェアの欠陥の識別に役立ちます。

テストが成功に貢献する理由は 4 つあります。

貢献	中身
欠陥を見つけるコスト効果のある手段	見つけた欠陥はデバッグで取り除ける。取り除くのはテスト活動ではないから、テストの貢献は間接的
品質を直接評価する手段	SDLC の各段階で測れる。結果はリリース判定など、次の段階へ進む判断に使う
ユーザーの代弁（間接的）	テスト担当者が理解したユーザーのニーズを、開発ライフサイクルを通じて反映できているかを確かめる
契約・法律・規制基準への適合	契約上や法律上の要件を満たす、規制基準に従うために必要になることがある

ユーザーの代弁には代わりの方法もあります。代表的なユーザーを何人か開発プロジェクトに参加させる方法です（ユーザーの関与）。ただしコストが高

SAMPLE

く、適切なユーザーを確保できないことが多いため、通常は実現できません。そこでテスト担当者がユーザーの立場を代弁します。

ここでの引っかけは、貢献の直接と間接です。欠陥を見つけること自体は直接の貢献に見えますが、品質が上がるのは欠陥を取り除いた後です。取り除くのはデバッグですから、品質向上への貢献は間接的と区別します。品質の評価は測定結果そのものなので、間接の手段ではありません。[7]

テストと品質保証 [1.2.2]

「テスト」と「品質保証」（QA）を同じ意味で使う人は多いのですが、同じではありません。テストは品質コントロール（QC）の一形式です。

	品質コントロール（QC）	品質保証（QA）
指向	プロダクト指向	プロセス指向
性格	是正アプローチ	予防的アプローチ
焦点	適切な品質の達成を支援する活動	プロセスの実装と改善
考え方	作ったものを調べて直す	よいプロセスが正しく行われれば、よいプロダクトができる
例	テスト、形式的手法（モデル検査・定理証明）、シミュレーション、プロトタイピング	開発プロセスとテストプロセスの改善と徹底

テストは QC の主要な形式ですが、QC にはテスト以外の手段もあります。QA は開発プロセスとテストプロセスの両方に適用され、プロジェクトに参加するすべての人が責任を持ちます。QA 部門だけの仕事ではありません。

SAMPLE

テスト結果は QA と QC の両方で使います。QC では欠陥の修正に使います。QA では、開発プロセスとテストプロセスがどの程度うまくいっているかのフィードバックに使います。

場面で考えると、次のように見分けられます。出荷前の製品を実際に動かして不具合を探す作業は、プロダクト指向の是正なので QC（テスト）であって、QA ではありません。「レビューを必ず行う」という手順を決めて全員に守らせ、手順が機能しているかを見直す作業は、プロセス指向の予防なので QA です。同じ不具合の一覧を、修正のために使えば QC、不具合が多い工程を見つけて手順を直すために使えば QA です。

エラー、欠陥、故障と根本原因 [1.2.3]

人間はエラー（誤り）を起こします。そのエラーが欠陥（フォールト、バグ）を生み、欠陥が故障につながることがあります。この 3 つは因果の順に並んでいて、置かれている場所も同じではありません。

用語	何か	どこにあるか
エラー	人間の誤り	人の頭や行動
欠陥	エラーが作り込んだ不備	作業成果物の中
故障	実行中のシステムが、すべきことをしない、またはすべきでないことをする事象（多くは欠陥の実行で起き、環境条件で起きることもある）	動いているシステムの外から見える振る舞い

人がエラーを起こす理由は、時間的なプレッシャー、作業成果物・プロセス・インフラ・相互作用の複雑さ、疲れ、十分なトレーニングを受けていないことなど、さまざまです。

SAMPLE

欠陥は、要件仕様書やテストスクリプトのようなドキュメント、ソースコード、ビルドファイルのような補助的なアーティファクトの中から見つかります。SDLCの初期に作られた成果物の欠陥は、検出されないと、しばしば後の工程で欠陥のある成果物につながります。コードの欠陥が実行されると、システムがすべきことをしない、あるいはすべきでないことをする、つまり故障が起きることがあります。

欠陥が故障になる条件は一定ではありません。実行すれば必ず故障になる欠陥もあれば、特定の状況下でしか故障にならない欠陥もあり、絶対に故障にならない欠陥もあります。

具体例です。仕様は「購入金額が 10,000 円以上なら送料無料」だったとします。開発担当者が条件を「10,000 円より大きい」と書き間違えました。この書き間違いが欠陥で、書き間違えた人の誤りがエラーです。購入金額 15,000 円の注文では、どちらの条件でも送料無料になるので、欠陥はあっても故障は起きません。購入金額がちょうど 10,000 円の注文では、仕様は送料無料なのに送料が請求されます。これが故障です。同じ欠陥でも、入力しだいで故障になったり、ならなかったりします。

故障の原因は、エラーや欠陥だけではありません。放射線や電磁波によってファームウェアに欠陥が生じる場合など、環境条件によっても故障は発生します。つまり、故障は欠陥から生まれることが多いものの、環境条件が原因になることもあります。「すべての故障は作業成果物の中の欠陥が原因である」は誤りです。

根本原因とは、問題が発生する根底の理由です。たとえば、エラーにつながる状況が該当します（時間の圧力、訓練不足など）。根本原因は根本原因分析によって特定します。根本原因分析は、故障が発生したときや欠陥が確認された

SAMPLE

ときに行います。根本原因を取り除くなどの対処をすると、同様の故障や欠陥を防いだり、頻度を減らしたりできます。

先ほどの送料の例に当てはめます。送料が誤って請求された事象が故障、条件式の不等号が欠陥、不等号を書き間違えたことがエラーです（上の表の 3 語）。その書き間違いが起きた背景が根本原因です。要件のレビューをしていなかった、仕様の「以上」と「より大きい」の違いを開発チームが理解していなかった、納期が迫って確認の時間がなかった、などが候補になります。根本原因を特定して手を打てば、不等号の書き間違いだけでなく、似た誤りの再発も減らせます。

区別するコツは、時間の順と場所です。エラーは作る人の行動、欠陥は成果物の中、故障は動作の結果で、この 3 つを混同してはなりません。問題文が「テスト実行中に画面が止まった」と書いていれば故障、「ソースコードの条件式が誤っていた」と書いていれば欠陥、「担当者が仕様を読み違えた」と書いていればエラーです。読み違えが起きる状況そのものを突き止める話なら、根本原因です。[8]

場面で確かめます。組込み機器が、放射線の影響でメモリの値が書き換わり、一度だけ誤動作する場合を考えます。ソースコードを全部調べても、誤った記述は見つかりません。この場合、故障は起きていますが、コードの欠陥が原因とは限りません。環境条件による故障の例です。逆に、故障が起きていなくても、実行すれば必ず誤る欠陥がコードに残っていることはあります。まだ実行されていないだけです。したがって、「欠陥があれば必ず故障が観察される」も、「故障があれば必ずコードに欠陥がある」も誤りです。根本原因分析は、故障が発生したときや欠陥が確認されたときに行うのが一般的です（シラバス 1.2.3 項）。

SAMPLE

テストの 7 原則 [1.3.1]

テストの原則は、長い間にわたって提唱され、あらゆるテストに適用できる一般的なガイドラインになってきました。シラバスは次の 7 つを示します。

番号	原則	言いたいこと
1	テストは欠陥があることは示せるが、欠陥がないことは示せない	欠陥が見つからなくても、正しさの証明にはならない
2	全数テストは不可能	すべての入力と条件の組合せは試せない
3	早期テストで時間とコストを節約	欠陥は早く取り除くほど安い
4	欠陥の偏在	欠陥は少数のコンポーネントに集中する
5	テストの弱化	同じテストの繰り返しは、新しい欠陥の検出に効かなくなる
6	テストはコンテキスト次第	唯一普遍のアプローチは存在しない
7	「欠陥ゼロ」の落とし穴	検証だけでは正しいシステムになるとは限らない

原則 5 の名前は、旧シラバス（2018 年版）では「殺虫剤のパラドックスにご用心」でした。同じ殺虫剤を撒き続けると虫が耐性を持つ、という例えです。現行の日本語シラバスの名前は「テストの弱化」ですが、内容は同じです。

原則 1: テストは、テスト対象に欠陥があることを示せます。欠陥がないことは証明できません。テストによって残っている未検出の欠陥の数は減らせます

SAMPLE

が、欠陥が見つからなくても正しさの証明にはなりません。「100 件のテストが全部合格したので、欠陥はない」は誤りです。

原則 2: すべてをテストするのは、ごく単純なソフトウェア以外では非現実的です。たとえば、入力項目が 10 個あり、それぞれ 10 通りの値を取るだけでも、組合せは 10 の 10 乗で 100 億通りになります。全数テストの代わりに、テスト技法（第 4 章）、テストケースの優先順位付け（5.1.5 項）、リスクベースドテスト（5.2 節）を使って、テストにかける労力を集中させます。

原則 3: プロセスの早い段階で欠陥を取り除くと、その欠陥に起因する後の成果物の欠陥が生まれません。SDLC の後半で起きる故障が減るので、品質コストが下がります。早く欠陥を見つけるためには、静的テストと動的テストの両方をなるべく早く始めます。要件のレビューは、コードが 1 行もない時期に始められる静的テストです。

原則 4: 見つかる欠陥や運用時の故障の大部分は、少数のシステムコンポーネントに集中します。これはパレートの法則の現れです（少数の要因に大半が集中）。予測した欠陥の偏在と、実際にテスト中や運用中に観察した偏在は、リスクベースドテストの大事な入力になります。たとえば、あるモジュールが前回のリリースで欠陥の大半を出したなら、今回はそのモジュールのテストを厚くします。

原則 5: 同じテストを何度も繰り返すと、新しい欠陥の検出に対する効果が薄れます。克服するには、テストとテストデータを変更したり、新しくテストを作ったりする必要が生じることがあります。ただし、自動化されたリグレッションテストのように、同じテストを繰り返すことが有益な結果を示す場合もあります（2.2.3 項）。繰り返しが目的に合うかどうかで判断します。新しい欠陥を探すための繰り返しは効果が落ち、既存機能が壊れていないことの確認には繰り返しが向いています。

SAMPLE

原則 6: テストに唯一普遍的に適用できるアプローチはありません。テストはコンテキストによって、異なる方法で行われます。たとえば、安全性が最重要の医療機器と、短期間で改善を重ねる Web サービスでは、テストの深さも進め方も違います。

原則 7: ソフトウェアを検証するだけでシステムを正しく作れると期待するのは、誤り、つまり思い込みです。指定された要件のすべてを徹底的にテストし、検出した欠陥のすべてを修正しても、次のようなシステムが作られることがあります。ユーザーのニーズや期待を満たさないシステム、顧客のビジネスゴールの達成に役立たないシステム、競合システムに比べて劣るシステムです（いずれも検証だけでは防げません）。検証に加えて妥当性確認も行うべきだというのが、この原則の結論です。

原則の取り違えは、場面問題でよく出ます。次の見分け方を使います。

場面の特徴	当てはまる原則
「テストで欠陥が見つからなかったから品質は完璧」	1
組合せが多すぎて全部は試せないので、優先順位を付ける	2
要件レビューで曖昧さを見つけ、後工程の手戻りを避けた	3
欠陥の多くが 1 つの決済モジュールに集まっている	4
毎回同じテストを回しているのに、新しい欠陥が出なくなった	5
医療機器と通販サイトで、テストのやり方がまるで違う	6

SAMPLE

場面の特徴	当てはまる原則
仕様どおりで欠陥も直したが、使いにくくて顧客が離れた	7

原則 2 と 4 は、リスクベースドテストと結びつく原則です（原則 2 は労力を集中させる手段の 1 つとして、原則 4 はリスクベースドテストの重要なインプットとして出てきます）。原則 5 は、繰り返しが有益な場合も認めているので、「繰り返しは常に無意味」とする選択肢は成り立ちません。[9]

テスト活動とタスク [1.4.1]

テストプロセスは、テスト目的を達成する確率を高めるための、汎用的なテスト活動のセットから構成されます。どの活動を、どのように、いつ実施するかは、通常、特定の状況に対するテスト計画の一部として決めます。シラバスは、主な活動のグループを 7 つ示します。

- テスト計画
- テストモニタリングとテストコントロール
- テスト分析
- テスト設計
- テスト実装
- テスト実行
- テスト完了

これらの活動の多くは、論理的には順番どおりに行うように見えます。しかし実際には、反復的に、あるいは並行して実施されることが多いです。システムやプロジェクトに合わせた調整も必要です。

SAMPLE

テスト計画: テスト目的を定義します。そのうえで、全体のコンテキストが課す制約の下で、目的を最も効果的に達成するアプローチを選びます（5.1 節）。

テストモニタリングとテストコントロール: テストモニタリングは、すべてのテスト活動を継続的にチェックし、実際の進捗をテスト計画と比べます。テストコントロールは、テスト目的を達成するために必要な行動をとります。詳細は 5.3 節で扱います。モニタリングが「見る」、コントロールが「手を打つ」で、見るだけでは手を打ったことになりません。

テスト分析: テストベースを分析して、テスト可能なフィーチャーを識別します。関連するテスト条件を定義して優先順位を付け、関連するリスクとリスクレベルも分析します。テストベースとテスト対象を評価し、含まれている可能性のある欠陥を識別したり、試験性のアセスメントをしたりします。テスト技法を使うことも多く、計測可能なカバレッジ基準から見て「何をテストするか」という問いに答える活動です。

テスト設計: テスト条件を、テストケースやテストチャーターなどのテストウェアに落とし込みます（テストチャーターは探索的テストで使う）。テストケースの入力を指定するガイドとして働くカバレッジアイテムの識別も、多くの場合含まれます。テストデータ要件の定義、テスト環境の設計、必要なインフラストラクチャとツールの識別もここで行います。「どのようにテストするか」という問いに答える活動です。

テスト実装: テスト実行に必要なテストウェア（たとえばテストデータ）を作成または取得します。テストケースはテストプロシジャーにまとめられ、テストスイートにも編成します（定義は 1.4.3 項）。手動と自動のテストスクリプトを作成します。テストプロシジャーは、効率的な実行のために、テスト実行スケジュール内で優先順位を付けて並べます（5.1.5 項）。テスト環境を構築し、正しく設定されているかを検証します。

SAMPLE

テスト実行: テスト実行スケジュールに従ってテストを走らせます（テストラン）。手動でも自動でも構いません。継続的テストやペアテストセッションなど、さまざまな形式があります。実際のテスト結果を期待結果と比較し、テスト結果を記録します。不正（期待結果と実際の結果の食い違い）を分析して、考えられる原因を識別します。この分析により、観察された故障に基づいて不正を報告できます（5.5 節）。

テスト完了: 通常、プロジェクトのマイルストーン（リリース、イテレーションの終了、テストレベルの完了など）で行います。対象は、未解決の欠陥、変更要求、作成したプロダクトバックログアイテムです。将来役立つ可能性のあるテストウェアをすべて識別して保管するか、適切なチームへ引き渡します。テスト環境は合意した状態でシャットダウンします。将来のイテレーション、リリース、プロジェクトに向けて、教訓と改善点を識別するために、テスト活動を分析します（2.1.6 項）。テスト完了レポートを作成して、ステークホルダーへ伝えます。

隣り合う活動の違いは、次の表で見分けます。

比べる活動	分かれ目
テスト分析と設計	分析は「何を」（テスト条件の識別・優先順位付け）、設計は「どのように」（テストケースの作成・カバレッジアイテム）
テスト設計と実装	設計はテストケースを作る。実装はそのケースを実行できる形（プロシジャー、スクリプト、スイート、実行スケジュール、環境）にそろえる
テスト実装と実行	実装は準備。実行はスケジュールに従って走らせ、実際の結果と期待結果を比べる

SAMPLE

比べる活動	分かれ目
モニタリングとコントロール	モニタリングは進捗の比較、コントロールは目的達成のための行動
テスト実行と完了	実行は結果の記録と不正の分析まで。教訓の識別、テストウェアの保管、環境のシャットダウン、完了レポートは完了

場面で判断する練習です。あるチームが、ログイン機能の要件を読み、パスワードの文字数制限や誤入力回数の上限など「確かめたいこと」を洗い出し、重要度の順に並べたとします。これはテスト条件の識別と優先順位付けなのでテスト分析であり、テスト設計ではありません。洗い出した条件ごとに、具体的な入力値と期待結果を書いたテストケースを作った段階に進めば、それはテスト設計です。作ったテストケースを手順書にまとめ、自動スクリプトを書き、実行する順番を決め、テスト環境がきちんと動くかを確認した段階はテスト実装です。それを順番に走らせ、期待結果と比べて食い違いを記録するのがテスト実行です。リリース後に、教訓をまとめて、使ったテストデータを次のイテレーションに引き継ぐ作業は、テスト完了です。

同じ場面に「進捗が計画より遅れていたので、テスト担当者を 1 名増やした」という出来事が入っていたら、これはテストコントロールであって、モニタリングではありません。「毎日の実施件数を計画と比べて遅れに気づいた」はテストモニタリングです。

実行と完了の境目も、場面で問われます。全テストケースを走らせて結果を記録し、不正の原因を分析して欠陥レポートを書くところまでが、テスト実行です。マイルストーンに達したあと、未解決の欠陥と変更要求を整理し、次のリリースで使えるテストウェアを保管し、テスト環境を合意した状態で停止し、教訓をまとめて完了レポートを渡すのが、テスト完了です。テスト環境を作って設定

SAMPLE

の正しさを確認するのは、実行の前の実装であって、完了ではありません。完了の活動に環境の構築は入りません。

コンテキストに応じたテストプロセス [1.4.2]

テストは単独では行われません。テスト活動は、組織内で行われる開発プロセスの不可欠な要素です。テストはステークホルダーによって資金が提供され、最終ゴールはステークホルダーのビジネスニーズの充足を支援することです。そのため、テストの行い方は次のコンテキストの要因に左右されます。

要因	含まれるもの
ステークホルダー	ニーズ、期待、要件、協力の意思
チームメンバー	スキル、知識、経験レベル、空き状況、トレーニングの必要性
ビジネスドメイン	テスト対象の重要性、識別したリスク、市場ニーズ、特定の法的規制
技術的要因	ソフトウェアの種類、プロダクトのアーキテクチャー、利用技術
プロジェクトの制約	スコープ、時間、予算、リソース
組織的要因	組織構造、現行のポリシー、使用する実践例
ソフトウェア開発ライフサイクル	エンジニアリングの実践例、開発手法
ツール	利用可能な状況、使用性、標準適合性

これらの要因は、利用するテスト戦略、テスト技法、テスト自動化の度合い、求められるカバレッジの度合い、テストドキュメントの詳細度合い、レポート作業など、テストに関する多くの事柄に影響します。

SAMPLE

場面に当てはめるときは、上の表の左の列にある名前を探します（8 行）。「納期が短縮され、人員も増やせない」はプロジェクトの制約、「法規制の対象となる金融システムなので、記録を詳しく残す」はビジネスドメイン、「開発をスクラムで行っているため、テストも短い反復に収める」はソフトウェア開発ライフサイクルです。要因が変われば、同じ機能でもテストの深さ、自動化の範囲、文書の厚みが変わります。[10]

テストウェア [1.4.3]

テストウェアは、1.4.1 項で説明した各テスト活動の出力として作られる作業成果物です。作成の方法、形式、名称、整理と管理の方法は組織によって大きく異なります。適切な構成管理（5.4 節）が、作業成果物の一貫性と整合性を支えます。次の表は、活動ごとの代表例です。すべてを網羅した一覧ではありません。

活動	主なテストウェア
テスト計画	テスト計画書、テストスケジュール、リスクレジスター、開始基準と終了基準
テストモニタリングとコントロール	テスト進捗レポート、コントロールのための指示を文書にしたもの、リスク情報
テスト分析	優先順位を付けたテスト条件（受け入れ基準など）、テストベース内の欠陥についての欠陥レポート（直接修正しない場合）
テスト設計	優先順位を付けたテストケース、テストチャーター、カバレッジアイテム、テストデータ要件、テスト環境要件
テスト実装	テストプロシジャー、自動テストスクリプト、テストスイート、テストデータ、テスト実行スケジュール、テスト環境

SAMPLE

活動	主なテストウェア
テスト実行	テスト結果の記録、欠陥レポート
テスト完了	テスト完了レポート、後続のプロジェクトやイテレーションを改善するアクションアイテム、得られた教訓のドキュメント、変更要求

補足があります。リスクレジスターは、リスクの可能性、リスクの影響、リスク軽減に関する情報とともに、リスクを列挙したものです。テストスケジュール、リスクレジスター、開始基準、終了基準は、多くの場合、テスト計画書の一部です。テスト環境には、スタブ、ドライバー、シミュレーター、サービス仮想化が含まれることがあります。変更要求は、たとえばプロダクトバックログアイテムとして出されます（アジャイルの場合）。

用語の違いも出題されます。テストデータは、テストの入力や、テストの前提として用意しておく値です。テストケースは、入力、実行条件、期待結果などを組にしたものです。テストプロシジャーは、テストケースを実行する順番にまとめた手順です。テストスイートは、特定のテストランで実行するテストスクリプトやテストプロシジャーのまとまりです（テストプロシジャーより 1 つ上の単位）。テストチャーターは、テストセッション（探索的テストなど）のゴールや目的を書いたドキュメントです。テスト条件は、コンポーネントやシステムのうち、テストできて、テストしようとしている側面です。テストログ（テスト結果の記録）は、実行したテストの結果を残したものです。

場面で選ぶ練習です。テストを実行して画面の誤表示を見つけたとき、作るのは欠陥レポートで、テスト完了レポートではありません。テスト完了レポートは、完了の時点で作ります。テスト分析の段階で、要件の記述に矛盾が見つかり、テスト担当者が直接直さない場合も、欠陥レポートを作ります。この欠陥は

SAMPLE

テストベースの中のものです。テスト実行の欠陥レポートとは別に、テスト分析の成果物にも欠陥レポートが載っている点が、見落としやすい所です。[10]

テストベースとテストウェアのトレーサビリティ [1.4.4]

テストのモニタリングとコントロールを効果的に行うには、テストベースの各要素と、関連するテストウェア（テスト条件、リスク、テストケース）、テスト結果、検出した欠陥との間のトレーサビリティを、テストプロセス全体を通して確立し、維持することが大切です。トレーサビリティとは、どの要素がどの要素から導かれたかをたどれる関係です。

正確なトレーサビリティは、カバレッジの評価を助けます。そのため、測定可能なカバレッジ基準をテストベースに定義してあると非常に役立ちます。カバレッジ基準は、テスト目的がどの程度達成できたかを示す活動を行うためのKPI として働きます（1.1.1 項）。

たどる関係	使い道
テストケースと要件	要件がテストケースでカバーされているかを検証できる
テスト結果とリスク	テスト対象に残る残存リスクのレベルを評価できる

カバレッジの評価に加えて、優れたトレーサビリティには次の効果もあります。変更の影響を判断できます。テストの監査を容易にします。IT ガバナンスの基準を満たす助けになります。また、テストベースの各要素の状況を含めることで、テスト進捗レポートと完了レポートがわかりやすくなります（5.3.2 項）。テストの技術的な側面を、ステークホルダーにわかりやすく伝える助けにもなります。さらに、プロダクト品質、プロセス能力、ビジネスゴールに対するプロジェクトの進捗を評価するための情報を提供します。

SAMPLE

たとえば、要件 R-12 が変更されたとします。要件とテストケースの間のトレーサビリティがあれば、R-12 に結びついたテストケースがすぐ分かり、見直すべきテストの範囲を絞れます。これが、上で挙げた「変更の影響を判断できる」効果に当たります（表の使い道とは別の効果）。逆に、どの要件ともつながらないテストケースがあれば、要件に無いものをテストしている可能性が見えます。テストケースを持たない要件は、カバーされていない要件です。[11]

テストの役割 [1.4.5]

シラバスは、テストにおける 2 つの主要な役割を取り上げます。テストをする役割と、テストマネジメントをする役割です。どちらの役割にどの活動とタスクが割り当てられるかは、プロジェクトやプロダクトのコンテキスト、各役割を担う人のスキル、組織などの要因で決まります。

	テストマネジメントをする役割	テストをする役割
責任	テストプロセス、テストチーム、テスト活動のリーダーシップに対する全体的な責任	テストのエンジニアリング（技術的）側面に対する全体的な責任
主な活動	テスト計画、テストモニタリングとコントロール、テスト完了	テスト分析、テスト設計、テスト実装、テスト実行

テストマネジメントをする役割の実施の仕方は、場面ごとに変わります。アジャイルソフトウェア開発では、テストマネジメントのタスクの一部をアジャイルチームが担当することもあります（2.1 節）。複数のチームや組織全体にまたがるタスクは、開発チーム以外のテストマネージャーが行うことがあります。

役割は人の肩書きとは別です。テストマネジメントをする役割は、チームリーダーが担うこともあれば、テストマネージャーや開発マネージャーが担うこと

SAMPLE

もあります。さまざまな人が、さまざまなタイミングで、これらの役割を担います。また、一人がテストをする役割とテストマネジメントをする役割を同時に担う場合もあります（兼務）。

「テストマネージャーという肩書きの人だけがテストマネジメントをする」「テストをする役割は、実行だけを担う」は誤りです。テストをする役割は、分析・設計・実装・実行の 4 つの活動に重点を置きます。実行はその 1 つです。[11]

テストに必要な汎用的スキル [1.5.1]

スキルとは、知識、実践、適性からくる、何かをうまく行う能力です。よいテスト担当者は、仕事をうまくこなすための次のスキルを持つべきです。優れたテスト担当者は、効果的なチームプレーヤーでもあり、異なるテスト独立性のレベルでテストを実行できる人です。

スキル	何のために役立つか
テスト知識	テスト技法の活用などで、テストの有効性を高める
徹底さ、慎重さ、好奇心、細部へのこだわり、理路整然さ	発見しにくい欠陥を識別する
優れたコミュニケーションスキル、積極的な傾聴、チームプレーヤーになること	すべてのステークホルダーと効果的にやりとりし、情報を伝え、理解してもらい、欠陥を報告して議論する
分析的思考、批判的思考、創造性	テストの有効性を高める
技術的な知識	適切なテストツールを使い、テストの効率性を高める
ドメイン知識	エンドユーザーやビジネス側代表者を理解し、意思疎通できるようになる

SAMPLE

有効性と効率性の使い分けに注意してください。テスト知識と分析的思考は有効性（欠陥を見つける力）、技術的な知識は効率性（ツールで手早く進める力）に結びついています（有効性と効率性は別の語）。

テスト担当者は、悪い知らせの運び手になることが多いです。悪い知らせの運び手を責めるのは、人間の一般的な特性です。そのため、コミュニケーションスキルが非常に大事です。テスト結果を伝えると、プロダクトやその作成者への批判と受け取られることがあります。確証バイアスは、現在持っている信念と異なる情報を受け入れにくくします（確証バイアス）。テストはプロジェクトの成功やプロダクトの品質に大きく貢献していますが、破壊的な活動だと見る人もいます。この見方を改めるために、欠陥や故障の情報は建設的な方法で伝えるべきです。

たとえば、「この機能は欠陥だらけです。誰が作ったのですか」と報告すると、作成者との関係が悪くなります。再現手順、期待結果、実際の結果、影響を、事実として淡々と共有すれば、作成者は欠陥の修正に集中できます（事実の共有）。[12]

チーム全体アプローチ [1.5.2]

テスト担当者にとって大事なスキルの 1 つは、チームの中で効果的に働き、チームのゴールに積極的に貢献する能力です。チーム全体アプローチは、エクストリームプログラミング（2.1 節）に由来し、このスキルの上に成り立っています。

チーム全体アプローチでは、必要な知識とスキルを持つメンバーなら、どの仕事を行ってもよく、全員が品質に責任を持ちます。同じ場所で働くと情報伝達と相互作用が活発になるので、メンバーは同じワークスペース（物理的でも仮想的でもよい）を共有します。

SAMPLE

利点は次の 3 つです（チームダイナミクス・情報伝達・相乗効果）。チームダイナミクスが向上します。チーム内の情報伝達とコラボレーションが強化されます。チーム内のさまざまなスキルセットを、プロジェクトの利益のために活用できる相乗効果が生まれます。

テスト担当者は、望ましい品質レベルの達成のために他のメンバーと緊密に連携します。ビジネス側の代表者と協力して適切な受け入れテストを作ります（第 2 章）。開発担当者と協働して、テスト戦略に合意し、テスト自動化アプローチを決めます。こうして、テストに関する知識を他のメンバーに広め、プロダクトの開発に影響を与えられます。

コンテキストによっては、チーム全体アプローチがいつも適切とは限りません。たとえばセーフティクリティカルな状況では、高いレベルのテストの独立性が必要になることがあります。[12]

テストの独立性 [1.5.3]

ある程度の独立性があると、作成者とテスト担当者の認知バイアスの違いから、テスト担当者が欠陥をより効果的に見つけられます。ただし、独立性は、開発担当者の慣れ親しんだ作業の代わりにはなりません。たとえば、開発担当者は自分のコードにある多くの欠陥を、効率的に見つけることができます。

作業成果物のテストを誰が行うかで、独立性の段階は 4 つに分かれます。

誰がテストするか	独立性
作成者本人	独立性なし
作成者と同じチームの仲間	ある程度の独立性あり
作成者のチーム外だが組織内のテスト担当者	独立性が高い

SAMPLE

誰がテストするか	独立性
組織外のテスト担当者	独立性がとても高い

ほとんどのプロジェクトでは、複数の独立性のレベルでテストを実施するのが最善です。たとえば、開発担当者はコンポーネントテストとコンポーネント統合テストを行います。テストチームはシステムテストとシステム統合テストを行います（2.2 節）。ビジネス側の代表者は受け入れテストを行います。

	内容
長所	独立したテスト担当者は経歴、技術的視点、バイアスが異なるので、開発担当者と比べて異なる種類の故障や欠陥を認識しやすい。システムの仕様策定や実装のときにステークホルダーが置いた仮定を、検証し、異議を唱え、反証できる
短所	開発チームから孤立し、協調性の欠如、情報伝達の問題、開発チームとの敵対関係に陥る可能性がある。開発担当者が品質への責任感を失うことがある。独立したテスト担当者がボトルネックとみなされたり、リリースの遅れを責められたりすることがある

見分けのコツがあります。長所は「見る目が違う」ことから生まれ、短所は「距離ができる」ことから生まれます。因果が逆の選択肢は誤りです。たとえば「独立したテスト担当者は開発担当者と同じ認知バイアスを持つため、同じ欠陥を見つけやすい」は、長所の理由が反対になっています（正しくは、バイアスが異なるから別の欠陥が見つかる）。「独立性が高いほど開発担当者の品質への責任感が高まる」も、短所の内容を長所のように言い換えた誤りです。

SAMPLE

場面で判断します。ある企業が、作成者本人のレビューだけで出荷していた機能に、独立したテストチームを入れました。結果として、同じコードから以前は見落とされていた欠陥が見つかりました。これは長所の典型例です。一方で、そのチームが開発から離れた場所で作業し、仕様変更が共有されなくなったとします。これは短所の孤立と情報伝達の問題の典型例です。さらに、開発担当者が「どうせテストチームが見つけてくれる」と考えて自分でのチェックを減らせば、品質への責任感の低下です。リリースが遅れたとき、テストチームだけが責められるなら、ボトルネックとみなされた状態に当たります（短所の 3 つ目）。

セーフティクリティカルなシステムでは、独立性の高いテストが必要な場面が多くあります。このため、チーム全体アプローチのように全員で品質を担う進め方でも、独立性の必要度は別に判断します。[1][2]

SAMPLE

サンプルはここまでです

続き(第2章〜巻末)は、JSTQB Foundation Level 問題集のご購入で、頻出度別ノート・暗記用ノートと合わせて3点すべてダウンロードできます。

Footnotes

1. JSTQB 試験実施案内 https://www.jstqb.jp/guidance/ ／ よくある質問 https://www.jstqb.jp/faq/ ↩
2. JSTQB 日本語シラバス 2023V4.0.J02（2024 年 7 月 10 日） https://www.jstqb.jp/wordpress/wp-content/uploads/2026/02/CORE-FOUNDATION-FLシラバスCTFL.pdf ／ ISTQB CTFL v4.0 https://istqb.org/certifications/certified-tester-foundation-level-ctfl-v4-0/ ↩
3. Pearson VUE JSTQB 試験のページ（受験料・試験時間） https://www.pearsonvue.com/jp/ja/jstqb.html ／ JSTQB 試験実施案内 https://www.jstqb.jp/guidance/ ↩
4. ISTQB Exam Structures and Rules v1.21（章ごとの問数と K レベルの内訳） https://istqb.org/?sdm_process_download=1&download_id=3832 ↩
5. ISTQB テスト技術者資格制度 Foundation Level シラバス Version 2023V4.0.J02（JSTQB）1.1 節・1.1.1 項 https://www.jstqb.jp/wordpress/wp-content/uploads/2026/02/CORE-FOUNDATION-FL シラバス CTFL.pdf#page=16 ↩

SAMPLE

6. 同シラバス 1.1.2 項 https://www.jstqb.jp/wordpress/wp-content/uploads/2026/02/CORE-FOUNDATION-FL_シラバス_CTFL.pdf#page=17 ↩
7. 同シラバス 1.2.1 項・1.2.2 項 https://www.jstqb.jp/wordpress/wp-content/uploads/2026/02/CORE-FOUNDATION-FL_シラバス_CTFL.pdf#page=17 ↩
8. 同シラバス 1.2.3 項 https://www.jstqb.jp/wordpress/wp-content/uploads/2026/02/CORE-FOUNDATION-FL_シラバス_CTFL.pdf#page=18 ↩
9. 同シラバス 1.3.1 項 https://www.jstqb.jp/wordpress/wp-content/uploads/2026/02/CORE-FOUNDATION-FL_シラバス_CTFL.pdf#page=18 。英語版 v4.0.1 の 1.3 節 https://istqb.org/?sdm_process_download=1&download_id=3345 。用語は ISTQB 用語集 https://glossary.istqb.org/ja_JP/ ↩
10. 同シラバス 1.4.1 項・1.4.2 項・1.4.3 項 https://www.jstqb.jp/wordpress/wp-content/uploads/2026/02/CORE-FOUNDATION-FL_シラバス_CTFL.pdf#page=19 ↩ ↩2
11. 同シラバス 1.4.4 項・1.4.5 項 https://www.jstqb.jp/wordpress/wp-content/uploads/2026/02/CORE-FOUNDATION-FL_シラバス_CTFL.pdf#page=21 ↩ ↩2
12. 同シラバス 1.5.1 項・1.5.2 項・1.5.3 項 https://www.jstqb.jp/wordpress/wp-

SAMPLE

content/uploads/2026/02/CORE-FOUNDATION-FL シ ラ バ ス CTFL.pdf#page=22 ↩ ↩2 ↩3

SAMPLE

JSTQB認定テスト技術者資格 Foundation Level(CTFL) 学習用テキスト(無料サンプル)

版

2026年11月版(ISTQB テスト技術者資格制度 Foundation Level シラバス 日本語版 Version 2023V4.0.J02(2024年7月10日)準拠)

発行

IT資格道場(https://www.shikaku-doujou.com)

お問い合わせ

info@shikaku-doujou.com

ISTQB® は International Software Testing Qualifications Board の登録商標です。JSTQB は ISTQB の加盟団体である日本ソフトウェアテスト資格認定委員会の略称です。記載されているその他の会社名・製品名・サービス名は、各社の商標または登録商標です。

本教材は公式シラバスの内容を基に作成したもので、JSTQB および ISTQB との提携・承認関係はありません。

本文・図表・設問はすべてオリジナルの著作物です。無断での転載・複製・再配布・二次販売はできません。お気づきの点は info@shikaku-doujou.com までご連絡ください。

SAMPLE