🧪 単体テスト仕様書 完全ガイド

> **対象読者**: 新人エンジニア・テスト設計初心者
> **目的**: 単体テストの仕様書作成と観点の理解
> **所要時間**: 約15分

📚 目次

🎯 1. 単体テストとは

定義

単体テスト（Unit Test） は、プログラムの最小単位（関数・メソッド・クラス・画面など）が設計通りに動作するかを確認するテストです。

目的の図解

```
[開発フロー]
 設計書作成
   ↓
 コーディング
   ↓
 🧪 単体テスト ← ここで品質を担保！
   ↓
 結合テスト
   ↓
 システムテスト
```

なぜ重要？

- ✅ **バグの早期発見** - 後工程で見つかるより修正コストが10倍以上安い
- ✅ **品質の可視化** - テスト結果で品質を客観的に示せる

- ✅ **仕様の確認** - 設計通りに実装されているか確認できる
- ✅ **リファクタリングの安全網** - 修正後も動作を保証できる

📄 2. テスト仕様書の基本構成

基本的な項目一覧

項目	説明	記載例
テストNo	一意の識別番号	`UT-001`, `TC_Login_001`
イベント/操作	テスト対象の操作	「検索ボタン押下」「ログイン実行」
前提条件	テスト実施前の状態	「ユーザーAでログイン済み」
入力値/条件	テスト時の入力データ	「検索キーワード: "東京"」
想定結果	期待される結果	「検索結果が10件表示される」
取得エビデンス	証跡の種類	「画面キャプチャ」「DBログ」
実施結果	実際の結果	`OK` / `NG`
備考	補足事項	「IE11では動作確認済み」

テスト仕様書の構造イメージ

```
┌──────────────────────────────────────────┐
│ 【単体テスト仕様書】                      │
│ 対象: ユーザー検索機能                    │
│ 作成日: 2025/09/30                       │
├──────────────────────────────────────────┤
│ テストNo: UT-001                         │
│ 観点: 正常系 - 検索結果の表示             │
│ ──────────────────────────────────────── │
│ 前提条件:                                │
│  - マスタDBに100件のユーザーデータあり    │
│  - ログイン済み                          │
│ ──────────────────────────────────────── │
│ 操作:                                    │
│  1. 検索キーワードに「田中」を入力        │
│  2. 検索ボタンをクリック                  │
│ ──────────────────────────────────────── │
│ 想定結果:                                │
│  - 検索結果が3件表示される                │
│  - ユーザー名に「田中」を含む             │
│  - 表示順は登録日時の降順                 │
│ ──────────────────────────────────────── │
```

```
エビデンス:
  - 画面キャプチャ (検索結果画面)
  - APIレスポンスログ
```

🔍 3. テスト観点の完全リスト

テスト観点は「何を確認すべきか」を明確にするための視点です。以下、実務で使われる代表的な観点を分類して紹介します。

📱 3-1. 画面・操作系

✅ 画面遷移

```
テスト観点の例:
├── 遷移先画面が正しいこと
├── 画面遷移時のパラメータが引き継がれること
├── ブラウザバック時の挙動が正しいこと
└── 別タブで開いた場合も正常動作すること
```

観点	確認内容	例
遷移先の正確性	設計通りの画面に遷移するか	ログイン後→トップ画面
初期表示内容	遷移先画面の初期値が正しいか	前画面の検索条件が保持される
パラメータ引継ぎ	URLパラメータやセッションが正しいか	`?userId=123` が渡される

✅ UI制御

```
├── ボタンの活性/非活性制御
├── 項目の表示/非表示制御
├── 入力項目のreadonly制御
└── ローディング表示
```

具体例:

- 必須項目が未入力の場合、登録ボタンが非活性
- 管理者権限がない場合、削除ボタンが非表示
- 確定済みデータは編集不可（readonly）

✅ 入力制御

観点	確認内容
初期値設定	デフォルト値が正しく設定されるか
IME制御	全角/半角の自動切り替えが動作するか
入力文字種制限	数字のみ、英数字のみなどの制限が効くか
最大文字数制限	maxlength属性が正しく動作するか
パスワードマスキング	type="password"が効いているか

📊 3-2. データ取得・表示系

✅ API/サーバー通信

```
確認ポイント:
├── Request回数（無駄な通信が発生していないか）
├── Requestパラメータの内容
├── Responseのステータスコード
├── タイムアウト処理
└── エラーハンドリング
```

テストケース	入力	期待結果
正常系: データ取得成功	検索条件: "東京"	200 OK, 10件取得
異常系: 0件ヒット	検索条件: "存在しない都市"	200 OK, 0件（エラーにならない）
異常系: タイムアウト	通信遅延30秒	エラーメッセージ表示

✅ データ表示・編集

```
├── 日付フォーマット: 2025/09/30
├── 数値フォーマット: 1,234,567円
├── 小数点以下の桁数: 99.99%
├── NULL値の表示: "-" または "未設定"
└── リンク・アイコンの表示
```

実装例のチェックポイント:

- 日付が `YYYY/MM/DD` 形式で表示されるか
- 金額に3桁カンマ区切りが入っているか
- 小数第2位まで表示されるか
- 空データ時に `null` と表示されていないか

💾 3-3. DB更新系（登録・更新・削除）

✅ CRUD操作の基本

[Create: 登録]
├── 全項目が設計通りに登録されるか
├── 自動採番（AUTO_INCREMENT）が動作するか
├── デフォルト値が設定されるか
└── 登録日時・更新日時が自動設定されるか

[Read: 参照] ※ 3-2で確認

[Update: 更新]
├── 指定した項目のみ更新されるか
├── 更新対象レコードが正しいか（WHERE条件）
├── 更新日時が更新されるか
└── 排他制御が動作するか

[Delete: 削除]
├── 論理削除の場合、フラグが立つか
├── 物理削除の場合、レコードが消えるか
├── 関連テーブルも正しく削除されるか（CASCADE）
└── 削除権限チェックが動作するか

✅ 複数レコード処理

テストケース	確認内容
一括登録	複数件を一度にINSERTできるか
一括更新	複数件を一度にUPDATEできるか
条件付き削除	WHERE条件で複数件を削除できるか

✅ トランザクション・整合性

重要なテスト観点:
├── トランザクション内で例外発生時、ロールバックされるか
├── コミット前に他セッションから参照できないか
├── NotNull制約違反でエラーになるか
├── 外部キー制約が効いているか
└── 楽観ロック/悲観ロックが動作するか

実践例:

sql

```
-- テストケース: トランザクション途中でエラー
BEGIN;
INSERT INTO users (name) VALUES ('太郎'); -- 成功
INSERT INTO users (name) VALUES (NULL);   -- NotNull違反でエラー
ROLLBACK; -- ここで1件目も取り消される
```

🎨 3-4. レイアウト・表示形式

✅ デザイン確認

観点	確認内容
レイアウト崩れ	ブラウザサイズ変更時も崩れないか
フォント	指定されたフォント（ゴシック等）で表示されるか
文字サイズ	12px、14px等の指定が正しいか
色・背景色	CSSで指定した色が適用されているか
アイコン	Font Awesome等のアイコンが表示されるか

✅ 文字・記号の扱い

```
確認項目:
├── 囲み文字: 【】『』などが正しく表示
├── 区切り文字: スラッシュ、ハイフンが正しい位置
├── 文字コード: UTF-8で文字化けしない
├── 特殊文字: ©, ®, ™ などが表示される
└── 絵文字: 😀 🎉 などが表示される（必要に応じて）
```

テスト例:

- 「株式会社」と「㈱」の両方が正しく表示されるか
- 氏名に「髙」「﨑」などの異体字が含まれても文字化けしないか

📝 3-5. ログ・エラーハンドリング

✅ ログ出力

```
確認ポイント:
├── ログレベル（INFO, WARN, ERROR）が適切か
├── ログ出力タイミングが設計通りか
├── 個人情報がログに出力されていないか
├── ログローテーション設定が動作するか
└── ログファイルの保存先が正しいか
```

良いログの例:

```
[2025-09-30 10:15:23] INFO  UserService - ユーザー検索開始 userId=123
[2025-09-30 10:15:24] INFO  UserService - 検索結果: 5件
```

悪いログの例 (個人情報漏洩リスク):

```
[2025-09-30 10:15:23] INFO  UserService - password=abc123 ← NG!
```

✅ エラーメッセージ

観点	確認内容
メッセージ内容	ユーザーに分かりやすい文言か
表示位置	画面上部/該当項目の下など適切か
複数エラー時	すべてのエラーが表示されるか
エラーコード	システムエラー時にコードが表示されるか
リカバリー方法	対処法が示されているか

実装例:

```
❌ 悪い例: "エラーが発生しました"
✅ 良い例: "メールアドレスの形式が正しくありません。例: user@example.com"
```

💡 4. 実践例：テストケースの書き方

ケーススタディ: ログイン機能

機能仕様

- ユーザーIDとパスワードを入力してログインする
- 認証成功後、トップ画面に遷移する
- 3回失敗するとアカウントロックされる

テストケース設計

No	観点	入力	想定結果
UT-001	正常系: 認証成功	ID: user01, PW: pass123	トップ画面に遷移
UT-002	異常系: パスワード誤り	ID: user01, PW: wrong	エラーメッセージ表示
UT-003	異常系: 存在しないID	ID: nobody, PW: pass123	エラーメッセージ表示
UT-004	異常系: 空入力	ID: (空), PW: (空)	必須エラー表示
UT-005	境界値: 3回失敗でロック	2回失敗後、3回目失敗	アカウントロック
UT-006	DB確認: ログ記録	正常ログイン	ログインログが記録される

詳細なテストケース例

【テストNo】: UT-001
【観点】: 正常系 - 認証成功

【前提条件】
- ユーザーマスタに以下のデータが登録済み
  - ユーザーID: user01
  - パスワード: pass123（ハッシュ化済み）
  - アカウント状態: 有効

【操作手順】
1. ログイン画面を開く
2. ユーザーID欄に「user01」を入力
3. パスワード欄に「pass123」を入力
4. ログインボタンをクリック

【想定結果】
✅ トップ画面に遷移すること
✅ ヘッダーに「user01さん、こんにちは」と表示されること
✅ セッションにユーザーIDが保存されること
✅ ログインログにINFOレベルで記録されること

【取得エビデンス】
- 画面キャプチャ（遷移後の画面）
- ブラウザ開発者ツール（セッション確認）
- ログファイル（login.log）

【実施結果】: OK
【実施日】: 2025/09/30
【実施者】: 山田太郎

⚠️ 5. よくある失敗例と対策

失敗例 1: 正常系しかテストしない

❌ **悪い例**:

```
テストケース: 検索機能
- キーワードに「東京」を入力して検索 → 結果が表示される
```

✅ **良い例**:

```
テストケース: 検索機能
- 正常系: キーワード「東京」→ 10件表示
- 異常系: キーワード「存在しない」→ 0件（エラーにならない）
- 境界値: キーワード100文字→ 正常動作
- 境界値: キーワード101文字→ エラーメッセージ
- 特殊文字: キーワード「<script>」→ サニタイズされる
```

失敗例 2: 想定結果が曖昧

❌ **悪い例**:

```
想定結果: 検索結果が表示される
```

✅ **良い例**:

```
想定結果:
- 検索結果リストに5件表示されること
- 各行に「ユーザー名」「メールアドレス」「登録日」が表示されること
- 登録日の新しい順に並んでいること
- ページング表示が「1/3」となっていること
```

失敗例 3: エビデンスが不足

❌ **悪い例**:

```
エビデンス: 画面キャプチャ
```

✅ **良い例**:

エビデンス:
- 画面キャプチャ（検索結果画面全体）
- ブラウザ開発者ツール（Network タブのAPIレスポンス）
- DBクエリ結果（取得された5件のレコード）
- ログファイル（search.log の該当行）

✅ 6. チェックリスト

テスト仕様書作成時に確認すべき項目をチェックリストにしました。

📋 仕様書の完成度チェック

- [ ] すべてのテストケースに一意のテストNoが振られているか
- [ ] 前提条件が明確に記載されているか
- [ ] 操作手順が具体的で再現可能か
- [ ] 想定結果が具体的で測定可能か
- [ ] 必要なエビデンスの種類が明記されているか

🎯 観点の網羅性チェック

- [ ] 正常系のテストケースがあるか
- [ ] 異常系のテストケースがあるか（エラー、0件、NULL等）
- [ ] 境界値のテストケースがあるか（最大値、最小値、0、1等）
- [ ] DB更新がある場合、更新内容の確認ケースがあるか
- [ ] 画面遷移がある場合、遷移先の確認ケースがあるか

🔍 品質チェック

- [ ] テストケースに重複がないか
- [ ] 漏れている観点がないか（デザインレビュー実施）
- [ ] 実施可能なテストケースか（環境、データ、権限が揃うか）
- [ ] エビデンスの取得方法が現実的か
- [ ] 第三者が読んでも理解できる内容か

📚 参考資料

推奨する学習リソース

- **書籍**:『テスト駆動開発』（Kent Beck著）
- **Webサイト**: ソフトウェアテストの教科書

- **ツール**: Selenium, JUnit, Jest, Postman

テンプレート

以下のようなExcelテンプレートを用意しておくと便利です：

```
| テストNo | 大分類 | 中分類 | 観点 | 前提条件 | 操作 | 想定結果 | エビデンス | 結果 | 備考 |
|----------|--------|--------|------|----------|------|----------|----------|------|------|
```

🎓 まとめ

新人エンジニアが押さえるべき3つのポイント

1. **観点を網羅的に考える**
   正常系だけでなく、異常系・境界値・性能も意識する
2. **想定結果を具体的に書く**
   「正しく動く」ではなく「○○が△△になる」と明記する
3. **エビデンスを確実に残す**
   画面キャプチャ、ログ、DBの状態を記録する

最後に

単体テスト仕様書は、ただのチェックリストではなく**「設計の意図を確認するための道具」**です。

丁寧にテストケースを設計することで：

- ✅ バグを早期に発見できる
- ✅ 仕様の理解が深まる
- ✅ 品質に自信を持ってリリースできる

わからないことがあれば、遠慮なく先輩エンジニアやチームに相談しましょう！

📝 **作成日**: 2025年9月30日
👩‍💻 **対象**: 新人エンジニア
🏷️ **タグ**: 単体テスト, テスト仕様書, 品質保証, UT, QA