第3章　製造

3.1　コーディング

コーディングを行う際には、コメントの書き方や変数、関数の命名規則、推奨されないコードの書き方などをあらかじめコーディング規約にまとめます。コーディング規約に沿ったソースコードは、読みやすさの向上や不具合が抑制され、一定の品質を保つようになります。

コーディング規約の内容はプロジェクトごとに異なりますが、一般的には以下の点について規定するとよいでしょう。

観　点	概　要
コメント	ソースコードのヘッダ部分、関数、メソッドの概要、変数宣言やコード本体へのコメントの書き方など Java言語など、言語標準でコメントの書き方を規定している場合はそれも考慮すること プログラムの内容理解に役立ち、保守性が向上する
スタイル	コードのインデント（字下げ）や1行の文字数、改行の位置、構文の書き方など コーディングスタイルを自動でチェックするツールを活用することも検討する（checkstyleなど） プログラムの読みやすさや内容理解に役立ち、保守性が向上する
命名規則	変数や関数、メソッド、クラス名等の命名規則 使用目的が分かるように命名する プログラムの読みやすさや内容理解に役立ち、保守性が向上する
コード作成上のルール	変数宣言時には必ず初期化を行う、特定の構文は利用しないなど、コード作成上のノウハウをまとめる 不具合を抑制することができ、品質が向上する

ソースコード作成後は、コーディングレビューを行います。ソースコードが詳細設計書通りに作成されているか、作成者以外の視点で机上点検を行います。コーディングレビューを行うことで不具合を早期に検出し、プログラムの不具合を抑制することができます。

コーディングレビューのチェックポイント

- □　プログラム設計書に基づいてプログラミングしているか
- □　コーディング基準に基づいてプログラムを作成しているか
- □　プログラムテスト結果を評価し、記録および保管しているか
- □　重要プログラムはプログラム作成者以外の者がテストしているか

3.2　単体テスト

単体テストでは、作成したソースコードのうち、クラス、メソッド、関数といった個々の部品（モジュール）に着目して、詳細設計書通りに正しく作成されていることを実際に動作させてテストします。単体テストは「Unit Testing」の略称である「UT」と呼ばれることがあります。また、今日では単体テスト のツールとしてテスティングフレームワークであるxUnitが利用されることが増えています。なお、単体テストは「製造」「テスト」という分類でみた場合に、一般的には「製造」に区分されます。

3.2.1　単体テスト計画の立案

単体テストのテスト方針やテスト環境、使用ツール、スケジュール、体制、留意事項などを計画して単体テスト計画書にまとめます。

単体テストのテスト方針には以下に示すものがあります。テスト対象となるモジュールの特性にあわせて選択します。

テスト方針	概　要
ブラックボックステスト	モジュールの入出力に着目したテスト ある入力に対して得られる出力結果が正しいことを確認する ブラックボックステストではモジュールの内部構造（どの命令が実行されたか、どの分岐が実行されたか、など）は考慮せず、あくまで実行結果のみで確認を行う
ホワイトボックステスト	モジュールの内部構造に着目したテスト 全ての命令文や分岐が正しく実行されることを確認する
トップダウンテスト	単体テストがすでに完了したモジュールのうち、上位のプログラムから順番に結合させてテストを行う テスト対象となるモジュールが呼び出す下位モジュールはまだテストが完了していないため、スタブ（下位モジュールの代わりとなるテスト用プログラム）を準備する必要がある
ボトムアップテスト	単体テストがすでに完了したモジュールのうち、下位モジュールから順番に結合させてテストを行う テスト対象となるモジュールを呼び出す上位モジュールはまだテストが完了していないため、ドライバ（上位モジュールの代わりとなるテスト用プログラム）を準備する必要がある

3.2.2　単体テストの品質管理

(1) 品質目標値の設定

テストを実施するにあたり、最初に潜在する不具合の予測値を設定します。この予測値は、テスト対象のソースコードStep数から算出されます。

具体的には、過去の類似システムや経験値から導き出されたStep当たりの不具合発生件数を使用し算出します。

(2) 実績の管理

発見した不具合数をモジュール単位で管理します。もし、実績値が目標値を大幅に下回る、もしくは上回る場合は、その原因を分析し目標値やテスト方法の見直し、追加テストなどを行います。

(3) 品質改善

テスト結果を評価し、次工程で注意すべき点など申し送り事項を整理します。

(4) 回帰テスト

モジュールを修正・変更した場合に、すでにテスト済みのモジュール、機能に影響がないこと確認するために、一度実施したテストを再実施することを回帰テストといいます。モジュールの修正・変更が行われる度に回帰テストを実施することにより、品質を保つことができます。

単体テストを計画する際に、回帰テストの実施を検討するとよいでしょう。回帰テストの実行にコストがかからないよう、通常はxUnitなどを使用してテストを自動化します。

3.2.3　単体テストの実施手順

(1) 単体テストの観点、内容、およびデータの作成

決定したテスト方式に合わせて、単体テストを行う観点、内容、およびデータを作成します。

テストを行う観点として以下を検討します。

観　点	概　要
網羅性	モジュールの構造に対してどの部分をどれだけ実行するか 以下の観点がある ・命令網羅(C0) すべての命令文を実行する ・分岐網羅(C1) すべての分岐（真偽両者の分岐）を実行する ・条件網羅(C2) 複数条件で起こりうる分岐の組み合わせをすべて実行する
境界値	分岐における境界点に対して、境界をまたぐ部分をテストする 例えば x < 0 という分岐については、x = 1, 0, -1でテストする
限界値	モジュールの入力に可能な最小値、最大値、最小値-1、最大値+1でテストする

テストの内容はテスト仕様書としてまとめます。テスト仕様書には、テストを行う条件、テスト手順、実行後の期待する結果が書かれており、テスト実施者はテスト仕様書に沿ってテストを行います。

テスト観点、内容、データについては、必ずレビューを行います。テストの内容やテストデータに誤りがあると、正確なテストを行うことができず、不具合検出を見逃す可能性があります。

(2) テスト環境、ツールの準備

実際にテスト対象モジュールを動かす環境を準備します。また、テストに必要となるスタブ、ドライバ、またはxUnitで使用するテストコードを準備します。

(3) テストの実施および結果の確認

テストスケジュールに沿ってテストを実施します。テスト結果が期待する結果と異なる場合、その原因を調査しプログラムに不具合がある場合は修正後、再テストを行います。

また、テスト実行時のログ、網羅性（カバレッジ）検証結果、検出した不具合の記録を保存します。