IT パスポート試験合格講座 Strategy & Management 08

- システム開発技術

1. 機能要件
   システム開発で定義される、必ず満たさなければならない機能に関する条件。例えば、データの構造、種類、処理の内容、出力の方法や形式など。

2. 非機能要件(Non-Functional Requirement)
   システム開発で定義される機能要件以外の要件。例えば、可用性、性能・拡張性、運用・保守性、移行性、セキュリティ、環境・エコロジーなど。

3. 品質特性
   ソフトウェアの品質を評価する基準。ISO/IEC の規格ではソフトウェアの品質特性モデルとして、下記の 6 つの特性を挙げている。

   (1) 機能性(Functionality)
   必要な機能の実装程度。合目的性(Suitability)、正確性(Accuracy)、相互運用性(Interoperability)、機密性(Security)、標準適合性(Compliance)。
   (2) 信頼性(Reliability)
   ある状況がある時間続いたときにソフトウェアがどの程度機能するかに影響する特性。障害許容性(Fault Tolerance)、回復性(Recoverability)など。
   (3) 使用性(Usability)
   利用するのにかかる手間、個人の努力などに影響する特性。理解性(Understandability)、習得性(Learnability)、運用性(Operability)など。
   (4) 効率性(Efficiency)
   ソフトウェアの性能や要するリソース量の影響度。時間効率性(Time Behavior)、資源効率性(Resource Behavior)など。
   (5) 保守性(Maintainability)
   変更を加えるときに必要な労力の特性。解析性(Analyzability)、変更性(Changeability)、安定性(Stability)、試験性(Testability)など。
   (6) 移植性(Portability)
   別の環境への移植のしやすさ。環境適応性(Adaptability)、設置性(Installability)、共存性(Co-existence)、置換性(Replaceability)など。

4. 外部設計（基本設計、概要設計）
   要件定義に基づいて、ユーザーに提供する機能、ユーザインターフェース、レイアウト、フォーマットなどを決定する工程。

5. 内部設計（Internal Design）
   外部設計で決定された要件を、どのような構成で、どのようにシステムを動作させて実現するかを決定する工程。

6. プログラミング（Programming）
   プログラム言語によって、目的とする処理のためのソースコードを作成（コーディング）する工程。動作のテストやデバッグなどの作業を含む場合もある。

7. コーディング（Coding）
   プログラム言語、ハードウェア記述言語、マークアップ言語などで、コンピュータが処理できるソースコードを作成する工程。

8. コンパイラ（Compiler）
   高水準プログラミング言語のソースコードを、機械語のロードモジュール（実行ファイル）に変換する言語プロセッサ。この変換作業をコンパイル（Compile）と呼ぶ。

9. デバッグ（Debug）
   プログラムの欠陥を取り除く修正作業。プログラムの欠陥のことを、虫を意味するバグと呼ぶとことに由来している。

10. ブラックボックステスト
    プログラム仕様書通りに動作することを検証するテスト。システムへの入力と出力結果のみを検証するテストであり、黒箱試験とも呼ばれる。

11. ホワイトボックステスト
    内部設計での仕様書の通りに動作することを検証するテスト。プログラムや単体のモジュールにおける単体テストとして実施され、白箱試験とも呼ばれる。

    ✓ 分岐網羅
    プログラムの全ての分岐経路を少なくとも 1 回実行するようにテストケースを作成するテスト方法。

- ✓ 同値分割（Equivalence Partitioning）
  出力結果が同じであると考えられるデータを入力するテスト省力化の方法。
- ✓ 限界値分析（境界値分析、Boundary Value Analysis）
  「合格の 60 点、不合格の 59 点」など、出力結果が変化するデータの境界であると考えられるデータをテストケースとして入力するテスト省力化の方法。
- ✓ 原因結果グラフ（Cause-Effect Graphing）
  複数の入力データと出力データを視覚化された関係グラフを、組み合わせ論理回路としてテストケースに用いるテスト方法。

12. コードレビュー（Code Review）
    複数の開発者がソースコードの体系的な査読・評価を行う工程。

13. 単体テスト（Unit Test）
    プログラムの内部設計書に基づいて、プログラムがモジュール単位で正しく動作するかをホワイトボックステストによって確認する工程。

14. 結合テスト（Integration Testing）
    単体テストに合格した複数のモジュールを組み合わせて動作させ、モジュール間のインターフェースを確認する工程。ドライバを使って下位のモジュールから結合するボトムアップテスト、スタブを使って上位のモジュールから結合するトップダウンテスト、モジュールをすべて組み合わせるビッグバンテストなどがある。

    - ✓ ドライバ（サンプルドライバ／コントローラ）
      上位モジュールの代わりに用いる代用品の総称。
    - ✓ スタブ（Stub）
      モジュールが呼び出す下位モジュールの代わりに用いる代用品の総称。

15. システムテスト（System Test）
    要件定義で決められた機能や性能が漏れなく備わっているかを確認する工程。機能テスト、性能テスト、操作性テスト、サブシステムの結合テスト、耐障害性テスト、負荷テスト、耐久テスト、例外処理テストなどが行われる。

16. 運用テスト（Operations Test）
    システムの利用者が、実際の業務に沿った環境でテストを行うことで、実際の稼働か可能か否かを確認する工程。

17. 回帰テスト（退行テスト、リグレッションテスト、Regression Test）
システムやプログラムを変更・修正した場合に、その作業によって、今まで正常に機能していた部分に不具合などの影響が出ていないかを確認する工程。

18. 受入れテスト
完成したシステムが、必要な機能や要求した性能を実際に満たしているか否かを、システムの利用者が確認する工程。

19. ソフトウェア保守
システムの運用を開始した後に、構成の変更や不具合の修正、機能の追加やソフトウェアのバージョンアップなどに対応するための作業。

20. プログラムステップ法
ソフトウェアの見積り方法のひとつ。プログラムを記述するソースコードの行数を基に、ソフトウェアの開発規模を算出する手法。

21. ファンクションポイント法
ソフトウェアの見積り方法のひとつ。外部入出力や内部ファイルの数と難易度の高さから論理的にファンクションポイントを設けて、開発規模を算出する手法。

22. 類推見積法
ソフトウェアの見積り方法のひとつ。過去に経験した類似の開発における経験やデータを基にして、開発規模を算出する手法。