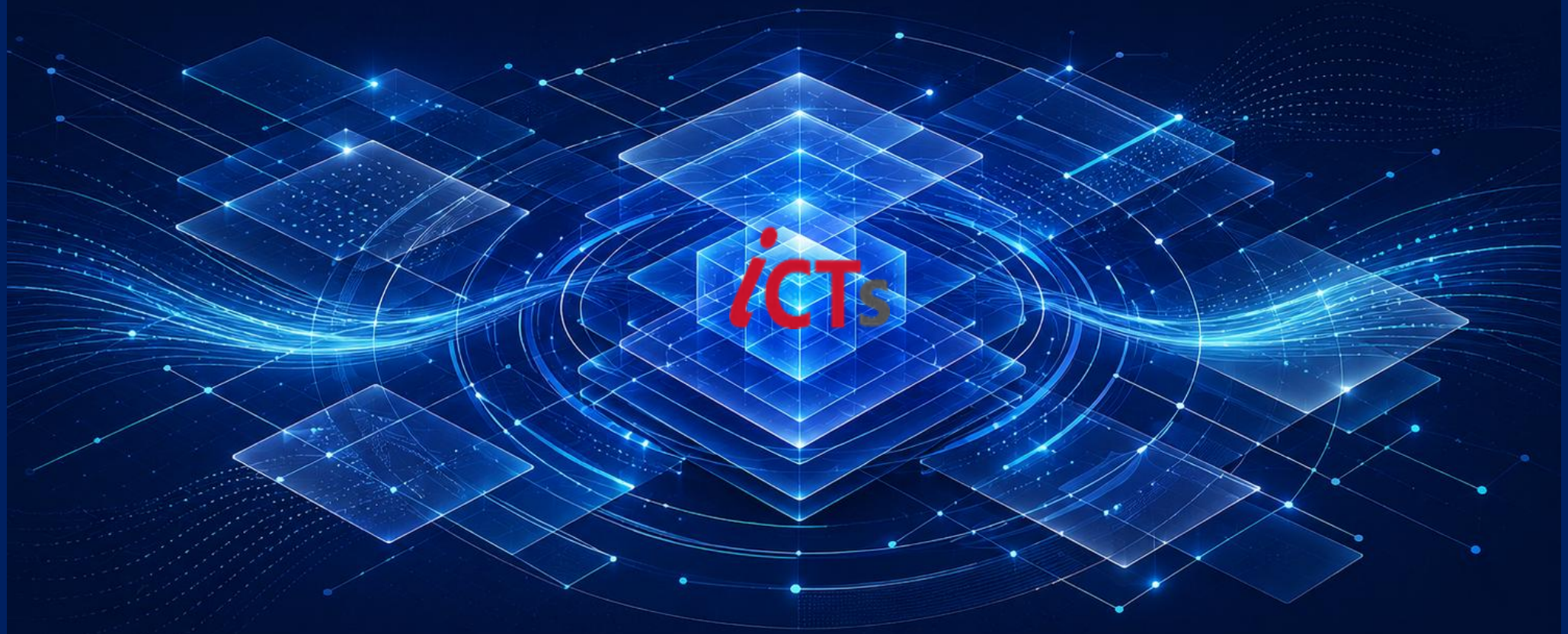
ICTシステム開発

システム設計資料



From Knowledge to Wisdom—
Built for the Real World.

NASAソフトウェア設計基準の適用構造

ディジタル空中写真測量システムの詳細設計に適用する確定ルール

上位要求
NPR 7150.2D / SWE-058

→

公式ガイダンス
NASA-HDBK-2203 Version D

→

詳細設計の到達点
コード化・コンパイル・
単体試験ができる設計

上位モジュール構成を変更せず、下位の設計単位まで具体化・記録・維持する

設計から実装まで

正式モジュール構成

↓

モジュール詳細設計

↓

クラス・関数・データ

↓

ソースコード

↓

単体・接続・
システム試験

各モジュールの設計内容

入力
名称・型・形式
単位・範囲・供給元

→

処理
数式・アルゴリズム
分岐・反復・判定

→

出力
名称・型・形式
状態・送信先

● 目的・識別・位置	● 上流・下流依存
● データ構造	● インターフェース
● 実行順序・制御フロー	● 異常・例外・ログ
● 性能・資源・制約	● 試験項目・合格条件
● 設計判断と理由	● 前提・制限・未確定事項

すべてを実装・試験可能な粒度で明記

横断管理

SwDD	構造・処理・設計判断
IDD	I/O・互換性・異常処理
SDD	型・単位・範囲・NoData
SWE-052	双方向追跡
SWE-080〜084	版・変更・正本管理
SWE-087〜089	レビュー・指摘解決

文書・コード・試験を一体管理

設計書の合格条件

- 曖昧さがない
- 直接実装できる
- 試験で確認できる
- 根拠を追跡できる

**正本のモジュール構成にない入力・処理・出力は、
確認なく追加・省略・変更しない**

適用基準：NPR 7150.2D / NASA-HDBK-2203 Version D

From Knowledge to Wisdom—
Built for the Real World.

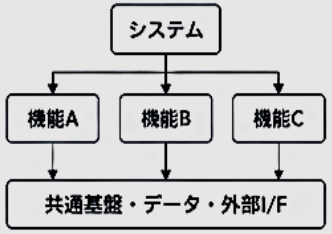
システム設計の５段階と責務（簡潔まとめ）

要求を価値ある製品として実現するための流れと、それぞれの責務を明確にする

① アーキテクチャ

システム全体を
どう構成するかを決める

（全体ブロック構成の例）



責務（何を担当するか）

- 目的を満たすための全体構造を決める
- 大きな機能ブロックを定義する
- ブロック間の関係・データ/制御の流れを決める
- 外部インターフェースを定める
- 品質属性（性能・信頼性・保守性など）の基本方針を決める

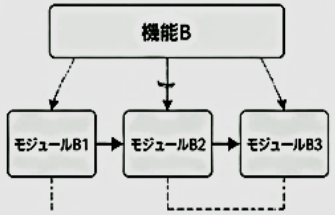
成果物（例）

- アーキテクチャ図（ブロック図）
- データフロー図（レベル1）
- インターフェース定義（外部/内部）
- 設計方針・制約・品質目標

② モジュール

各機能ブロックを
どのように担当させるかを決める

（モジュール構成の例）



責務（何を担当するか）

- 機能ブロックを実現するモジュールに分割する
- 各モジュールの入力→処理→出力を定義する
- モジュール間のインターフェースを定義する
- 例外処理・基本アルゴリズムを定義する

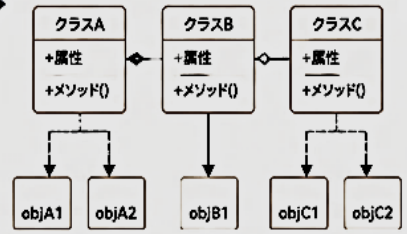
成果物（例）

- モジュール構成図
- モジュール仕様書（入出力・処理・I/F）
- 処理フロー図
- 例外処理・エラーハンドリング定義

③ クラス／オブジェクト

プログラムをどの構造・動きで
実現するかを決める

（クラス構成とオブジェクトの例）



責務（何を担当するか）

- モジュールを実現するクラスを設計する
- クラスの属性・メソッド・関係を定義する
- オブジェクトの生成・所有・ライフサイクルを設計する
- オブジェクト間の呼出し・連携の順序を設計する
- データ構造・状態遷移を定義する

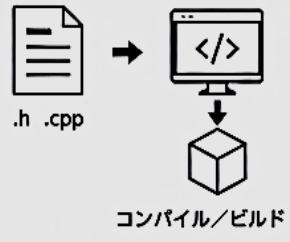
成果物（例）

- クラス図
- シーケンス図／コミュニケーション図
- オブジェクト構成図・ライフサイクル図
- データ定義書・状態遷移図

④ 実装

設計どおりに
コードを作る

（実装のイメージ）



責務（何を担当するか）

- 詳細設計を正確にコードへ変換する
- コーディング規約に従って実装する
- ビルド・単体試験を行い、設計どおりに動作することを確認する
- バグ修正・リファクタリングを行う

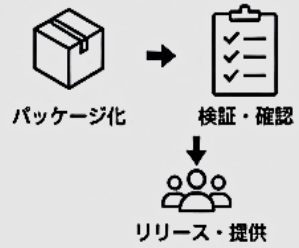
成果物（例）

- ソースコード（.h/.cpp 等）
- ビルド成果物（ライブラリ/実行ファイル）
- 単体試験結果・カバレッジレポート
- 静的解析・コードレビュー結果

⑤ 製品化

利用者に価値を届ける
製品として成立させる

（製品化のイメージ）



責務（何を担当するか）

- システム統合を行い、要求を満たすことを確認する
- システム試験・受入試験を実施する
- 製品としての品質を確保する
- リリース・配布・導入を行う
- 運用・保守の体制とドキュメントを整える

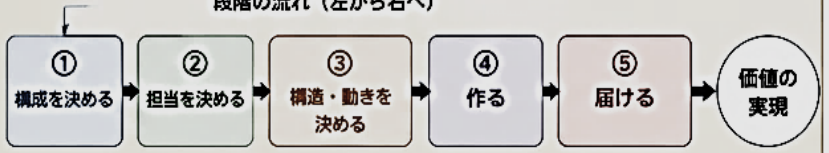
成果物（例）

- 統合版実行ファイル／パッケージ
- 試験報告書（統合・システム・受入）
- ユーザマニュアル・運用手順書
- リリースノート・保守計画書

ポイント（全体まとめ）

- 上位の段階ほど「何を実現するか」（目的・構造）を決め、下位の段階ほど「どう実現するか」（具体・詳細）を決める。
- 設計（①〜③）で「やることの全体像と実現方法」を決め切り、実装（④）は設計どおりに正確に作る。
- 製品化（⑤）は「動くソフト」を「使える製品」に仕上げ、利用者に価値を届ける責務を担う。

段階の流れ（左から右へ）



From Knowledge to Wisdom — Built for the Real World.

システム設計における モジュール・クラス・オブジェクト

モジュールは「何を実現するか」、クラスは「どう実現するか」、オブジェクトは「実際に処理する」という役割の違いがあります。
設計時（静的）と実行時（動的）を区別して考えることで、モジュールごとの責任が明確になります。

ポイント
- モジュールとクラスは設計時の構造（静的に分解する）
- オブジェクトは実行時の実体（動的に生成・破棄される）
- 責務を明確に分けることで、保守しやすく、再利用性の高い設計になる

モジュール
「何を実現するか」

- システムの機能をまとめた設計単位
- 複数のクラスを含む
- 外部とのインターフェースを持つ
- 他のモジュールと依存関係を持つ

例 **画像入力・検査モジュール**
入力された画像を受け取り、検査を行い、結果を出力する。

クラス
「どう実現するか」

- データ（状態）と処理（メソッド）を定義する型
- モジュール内の処理を具体化
- 継承・集約・関連などで他のクラスとつながる

例 **ImageValidator クラス**
画像データを検査し、検査結果を生成する。

オブジェクト
「実際に処理する」

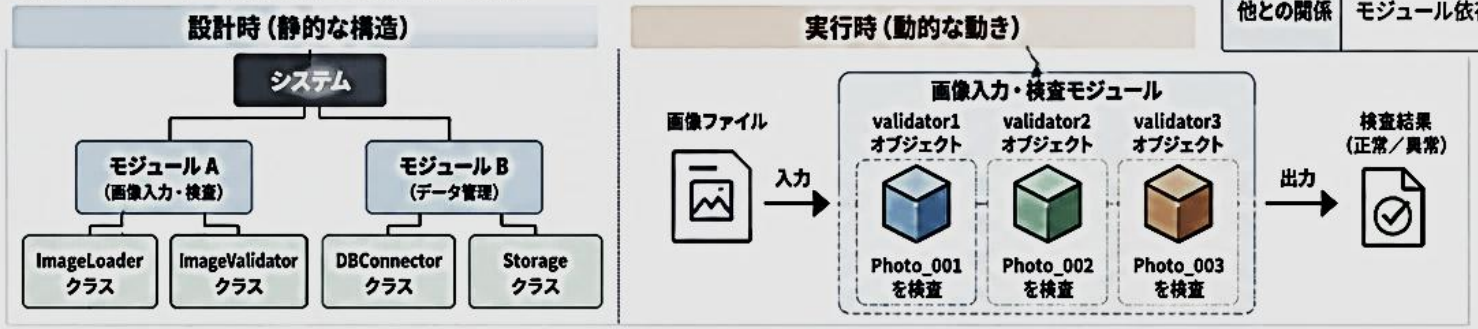
- クラスから生成された実体
- それぞれが状態（状態値）を持つ
- 実行時に生成され、使い終わると破棄される
- 同じクラスからも別の状態を持つ複数のオブジェクトを生成できる

例 **validator1、validator2**
それぞれ異なる画像ファイルを検査している。

モジュール・クラス・オブジェクトの比較

項目	モジュール	クラス	オブジェクト
主な位置	アーキテクチャ／詳細設計	詳細設計／実装設計	実行時
意味	機能をまとめた設計単位	データと処理を定義する型	クラスから生成された実体
主な責務	システム機能上の何を担当するか	モジュール責務をどのように実現するか	実際の状態を保持し処理を実行すること
入出力	モジュールI/F（入力・出力）	メソッド（引数・戻り値）	実際の値・状態（例：ファイル名、画像データ）
状態	原則として抽象的	状態の構造を定義	実際の状態値を保持
寿命	システム設計上存在	プログラム定義として存在	生成から破棄まで
他との関係	モジュール依存関係	継承・集約などの関係・依存など	呼出し・メッセージ・参照・イベントなど

関係のイメージ（静的構造と実行時の動き）



設計から実装までの順序（責務を整理して設計）

1. **モジュール責務を決める**
   このモジュールは何を入力し、何を行い、何を保証して出力するかを確定する。
2. **モジュール内部の処理を分解**
   入力 → 処理 → 出力、分岐、異常処理、データフローを設計する。
3. **処理責務をクラスへ割り当てる**
   一つのクラスに無関係な責務を集めず、「何を知っているか」「何を行うか」を明確にする。
4. **クラス間インターフェースを決める**
   メソッド、引数、戻り値、データ構造、エラー、所有関係を定義する。
5. **オブジェクトの動きを設計する**
   どのオブジェクトが生成され、誰が所有し、誰を呼び、いつ破棄されるかを設計する。

A モジュール ＝ 機能を実現する場 ／ クラス ＝ その機能を実現する具体的な設計 ／ オブジェクト ＝ 実行時の実体

より良い設計で、より保守しやすいシステムを

From Knowledge to Wisdom — Built for the Real World.