ソフトウエア工学：第２回

掛下 哲郎
kake@is.saga-u.ac.jp

第2回の内容

プロセスモデルとは？

Software Lifecycle Process (SLCP)

代表的なプロセスモデル

- ウォーターフォールモデル
- スパイラルモデル
- テスト駆動開発（TDD）
- エクストリームプログラミング（XP）

高品質ソフトウェアは高品質のプロセスによって作り出す

工学の基本

共通フレーム2013

ソフトウエア開発工程とその目的

- 企画、要件定義、基本設計、詳細設計、コーディング、ソフトウェアテスト、運用・保守

スラムダンクモデル

- 分析や設計を行なわず、直接プログラミングを行なう。
- その結果
  - 低品質プログラムの開発
  - プログラマ間の分業ができない。
  - 予算や納期を満たせないソフトウエア開発
    - 作業のやり直しがしばしば発生

プロセスモデル:
ソフトウェア開発の基本手順

プロセスモデルの重要性

プロセスモデルは「戦略」

- 戦略のないプロジェクトは緊急事態に忙殺される。
- 戦略のあるプロジェクトは先を読んで対応できる。

プロセスモデルは「ソフトウエア開発計画」

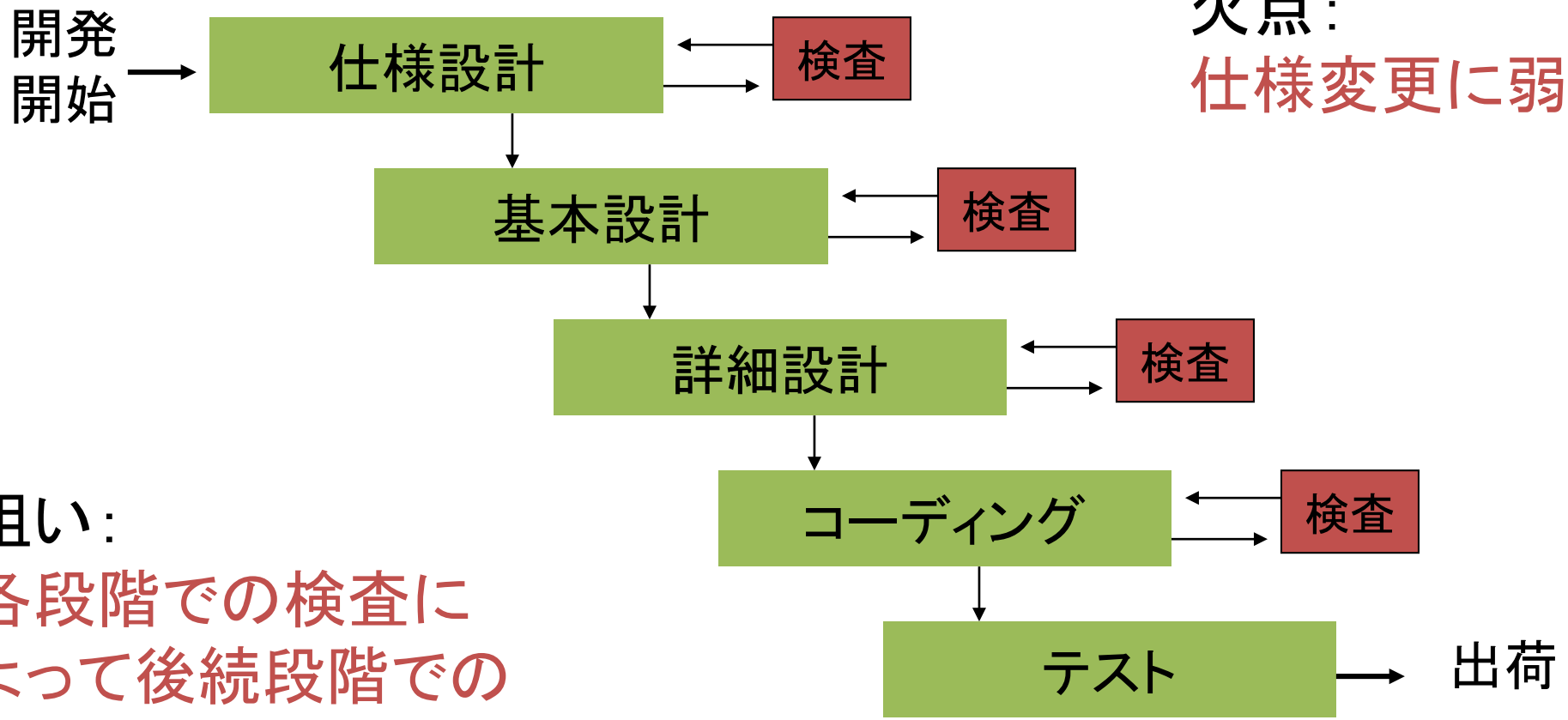
- 計画がないプロジェクトは管理できない。
- 管理されたプロジェクトは守れないような納期と費用を約束しない。
- 管理されたプロジェクトは品質を保証できる。

プロセスモデルは開発プロジェクトの「基本原則」

- 原則がないと、状況の変化や緊急事態に対処できない。
- 優れた原則は、メンバーの行動に対する共通指針を与える。
- 優れた原則は、他のプロジェクトにも繰り返し適用できる。

大規模プロジェクトには系統的なプロセスが不可欠

ウォーターフォールモデル



欠点:
仕様変更に弱い。

狙い:
各段階での検査によって後続段階でのコスト増を抑える。

エラー除去コストの比較

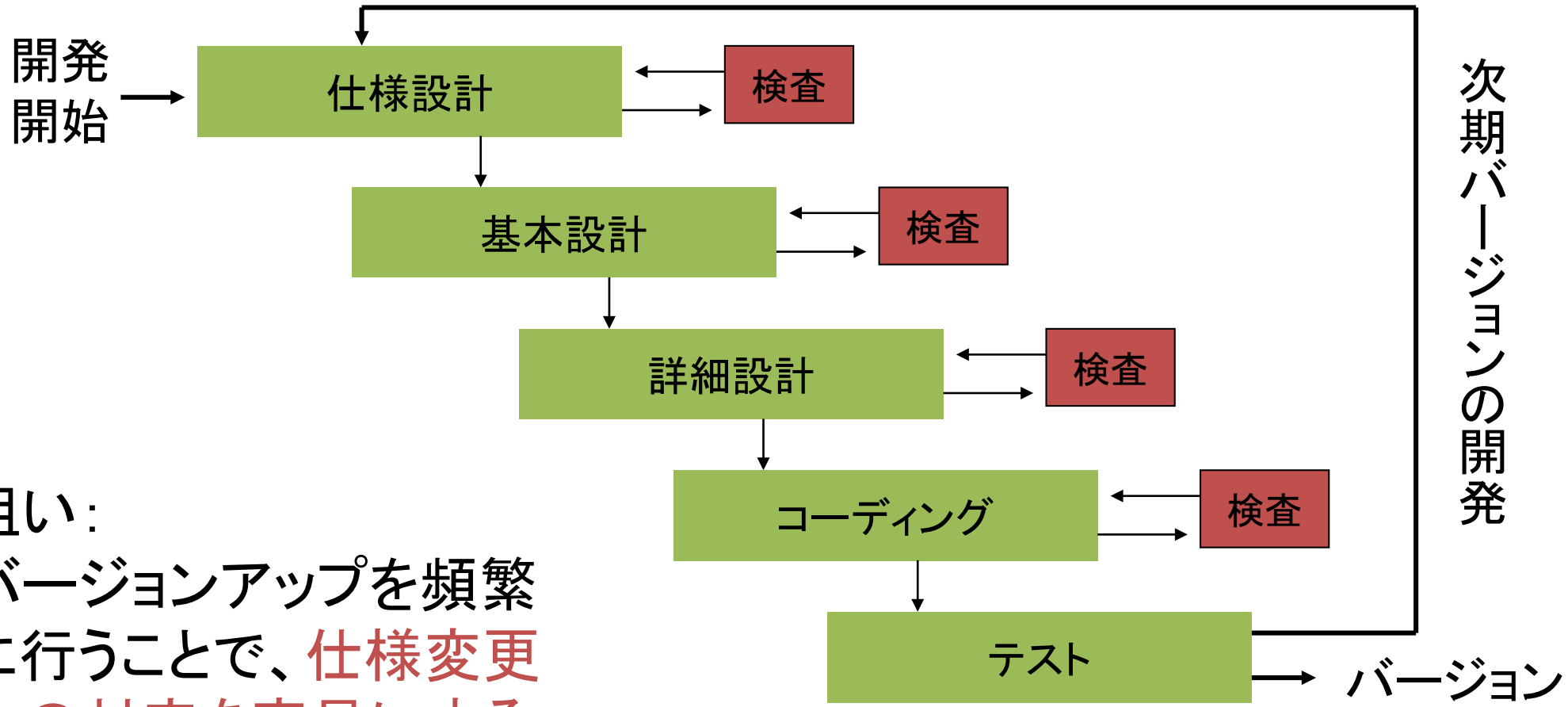
		バグ混入時点		
		仕様分析	設計	コーディング
バグ発見時点	仕様分析	1		
	設計	2	1	
	動作テスト	5	2	1
	構造テスト	15	5	2
	機能テスト	25	10	5

鉄則：バグは早期発見、早期対処

ウォーターフォールモデルの特徴

- 最も基本的なプロセスモデル
- 実プロジェクトで最も多く採用されている。
- 大規模プロジェクトにおける実績が多い。
  - プロジェクト管理が比較的容易
  - 有効性が実証済み
- 問題点
  - 障害復旧に対する柔軟性が乏しい
  - 顧客要求の変化への対応が困難

スパイラルモデル



狙い：
バージョンアップを頻繁に行うことで、仕様変更への対応を容易にする。

スパイラルモデルの特徴

代表的なプロジェクトの進行と一致する。

緻密なプロジェクト管理が必要

- プロジェクトが反復を終える際にドキュメンテーション類が一貫していなければならない。

スパイラルの反復回数

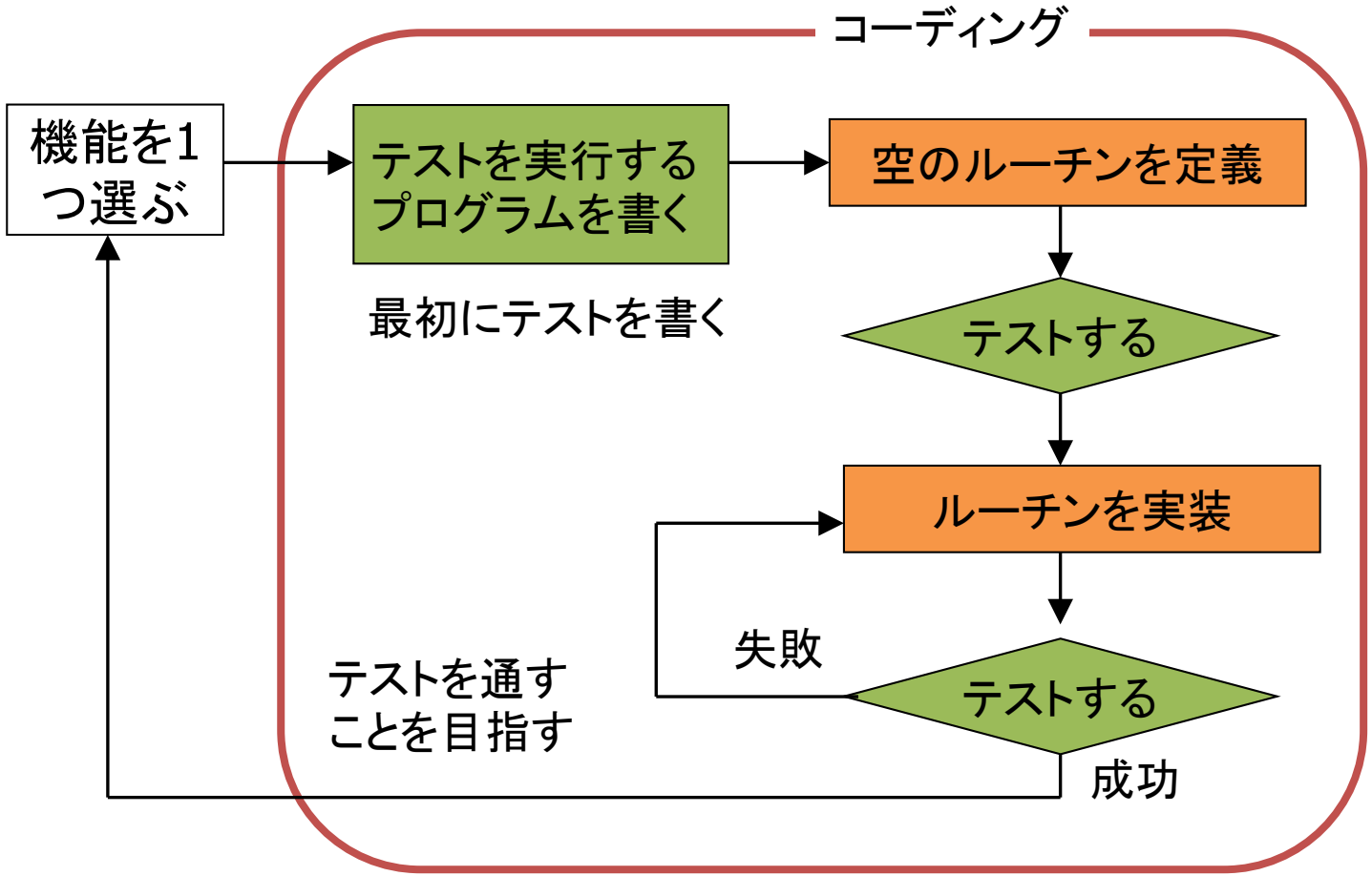
- 目安：12人月規模で2〜3回

応用：インクリメンタルな開発

- 各反復でわずかな機能のみを追加。反復回数の増加。
- 例：Microsoft社（プロジェクトをパーツに分解。パーツ毎にコードとドキュメンテーションを毎日、所定の時間に更新。定期的にパーツを組み合わせる）

テスト駆動開発

TDD：Test Driven Development



テスト駆動開発の特徴

仕様が明確になる。

- テストプログラムを作成するには仕様の理解が不可欠
- 具体的な実行例を考える機会を与える

統合テストが楽

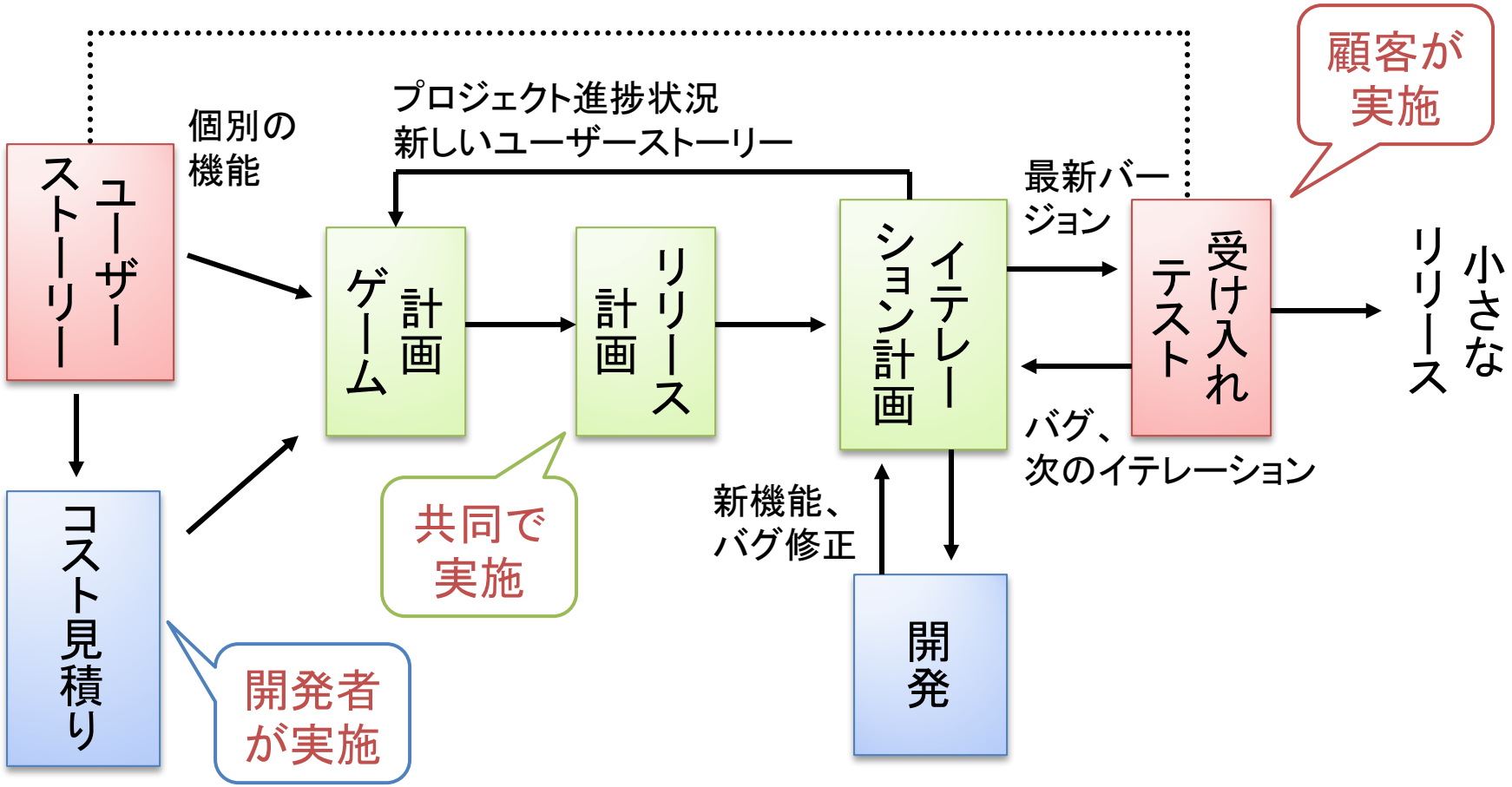
- 開発の過程でテストケースが作成される

開発がリズミカルに進められる

- 個別の機能毎に確認しながら開発
- テストを通じた動作の確認

コーディングとテストのみをカバー

Extreme Programming (XP)



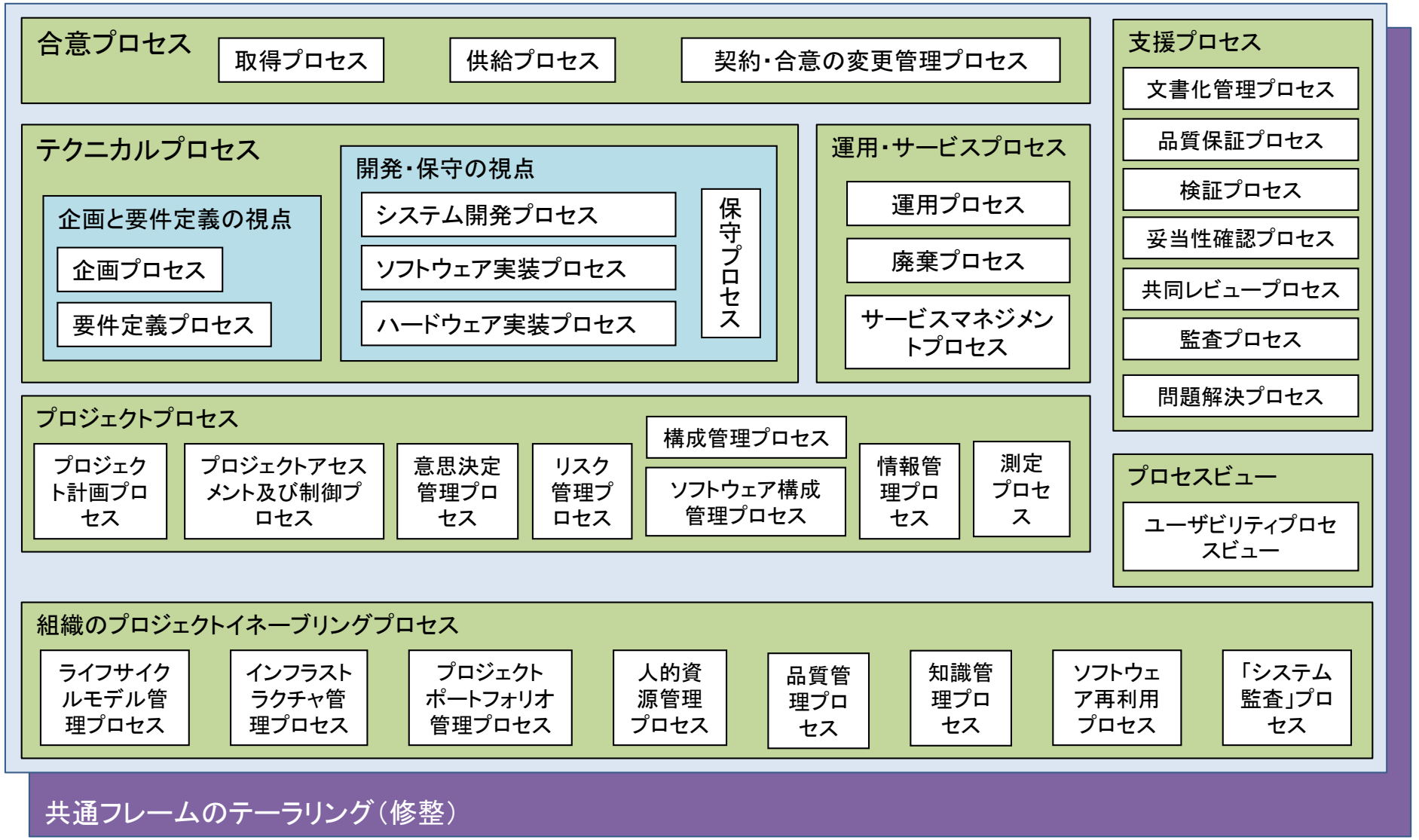
XPの主要な特徴（実践項目）

- 開発プロセス全体
  - 顧客が開発の現場におり、プログラマやマネージャーと直接打ち合わせる。
  - プログラミングの前にテストを定義する。
  - 2～3ヶ月毎の小さなリリース
- プログラミング
  - コーディング標準
  - ペアプログラミングによる学習・相乗効果
  - 設計の単純化（最低限の機能実装、リファクタリング）
  - コードの共同所有、頻繁な結合/実行、週40時間労働

共通フレーム2013

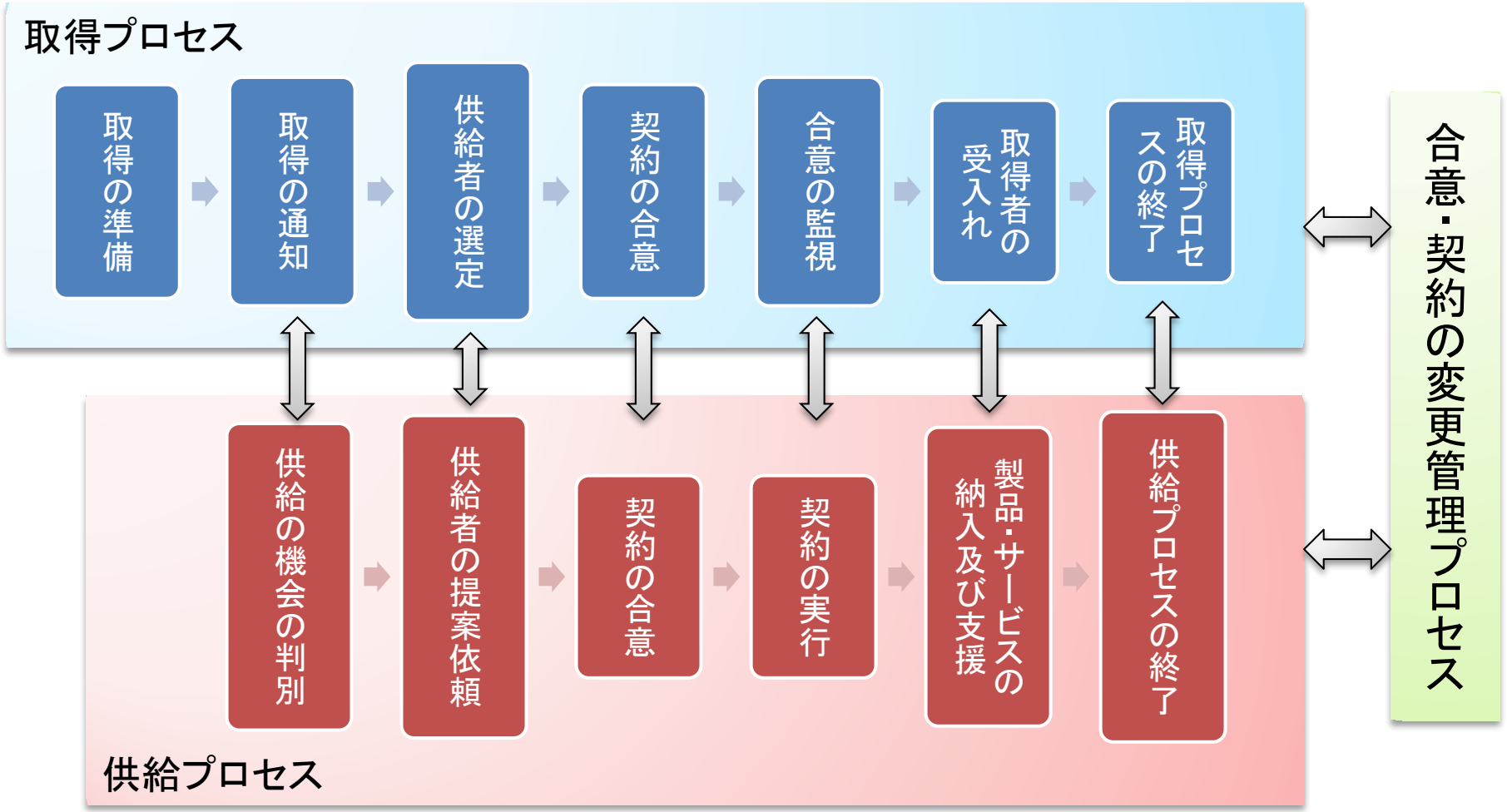
- ITシステム開発の作業規定
  - ソフトウェアの発注者，開発者，運用者が行うべき作業を明確化
  - ソフトウェア開発に関係する人々が「同じ言葉で話す」ことができるようにする
  - IT紛争を解決する際にも参照される
- ISO標準規格を踏まえて国が策定
  - ISO/IEC 12207:2017（JIS X0160:2012）⇒ ソフトウェア工学
  - ISO/IEC 15288:2023（JIS X0170:2013）⇒ システム工学
  - ISO/IEC/IEEE 29148:2018 ⇒ 要求工学
- ウォーターフォール，スパイラル，アジャイルなどすべての開発方法論に共通

共通フレーム2013の基本構成

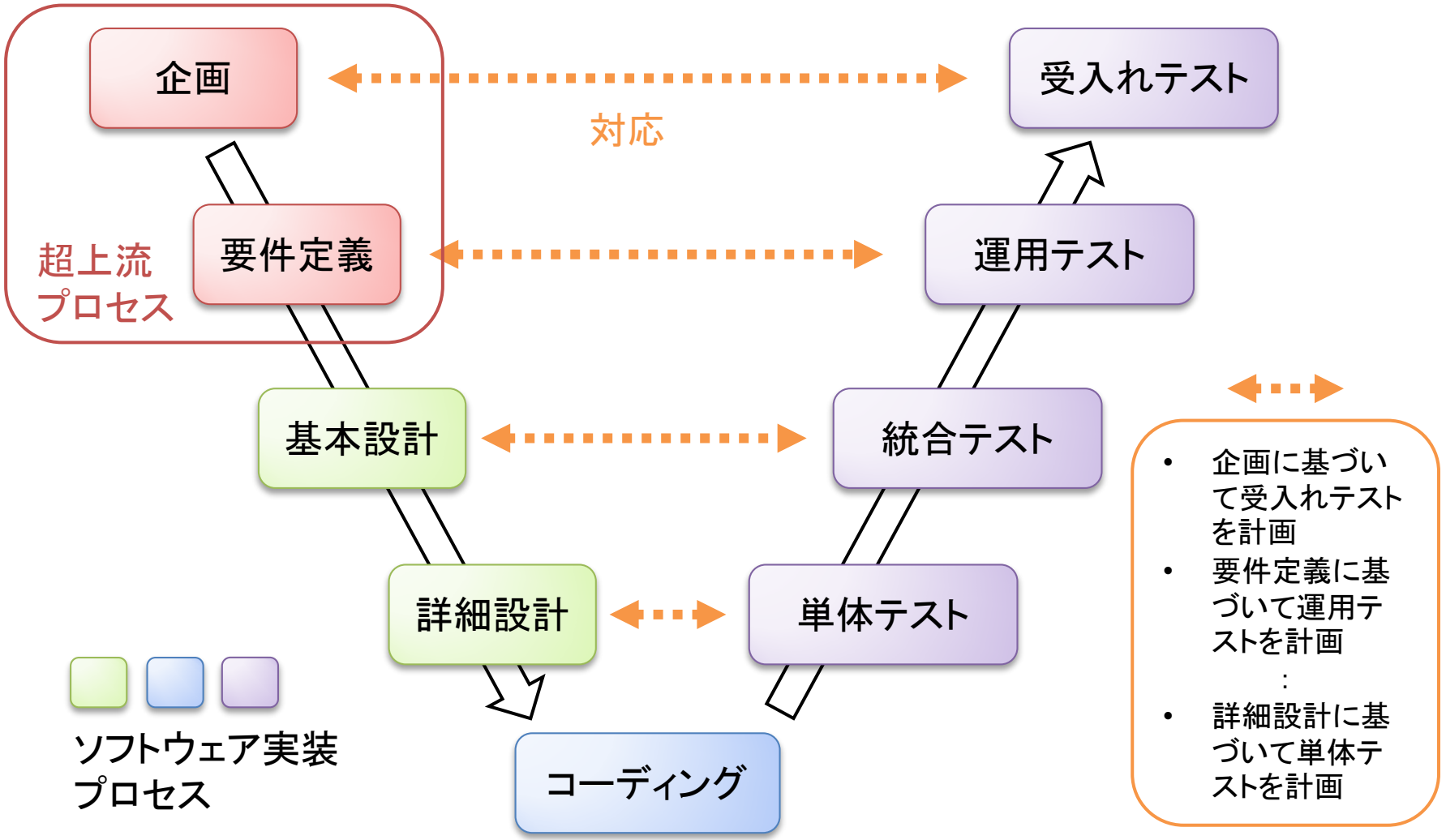


共通フレーム2013・主ライフサイクルプロセス

情報システムの取得と供給



ソフトウェア開発のV字モデル



企画

- 顧客の現状を分析する。
- 顧客の抱える問題点を理解する。
  - 顧客ニーズ(必要度):この問題が解決できないと何が困るか？
  - 顧客ウォンツ(欲求):この問題が解決できると何が嬉しいか？
- 顧客の問題を解決するための可能な手段を列挙する。
- 列挙された手段の中から最も適切なものを選択する。
  - 顧客ニーズに基づく選択
  - 開発者の強みに基づく選択
  - 開発コスト(費用・納期)に基づく選択

要件定義

- ソフトウエアに対する要求事項を文書化する。
  - 原則として仕様書に書かれている事項だけを実装する。
- 仕様書の記載に漏れがないようにする。
- 仕様書の記載に矛盾がないようにする。
- 開発コストを見積もる。

プロジェクトが大失敗する原因

見積もりミス

- 開発者が関与しない見積もり
- 仕様確定前の見積もり
- 仕様変更後に再見積もりを実施しない。

仕様が確定できないこと

- 仕様の抜けは仕様関係の不具合の中で修正が最も難しい。

基本設計

ソフトウエアをいくつかの部分に分割する。

- 機能に基づいた分割　:構造化設計
- データに基づいた分割:オブジェクト指向設計

モジュール毎に、提供する機能(ルーチン)とその仕様を決める。

- モジュール名、モジュール間の相互関連
- ルーチン名、引数・戻り値の名前とデータ構造

ソフトウエアの基本構造を決める重要なステップ

詳細設計

- モジュールの各機能を実現する手順（アルゴリズム）の決定
- アルゴリズムが必要とするデータ構造の決定
- アルゴリズム設計までは、プログラミング言語とは独立して自然言語で行う.

設計の重要性

- ソフトウエア設計は、ソフトウエアの保守性や再利用性に大きく影響する。
- ソフトウエア設計において、ベストの解が1つしかないことは、通常ありえない。
- ソフトウエアの設計は、複雑で反復が必要なプロセスである。
  - 最初に考えついた設計方式が間違っている可能性は高い。
- ソフトウエアの処理効率には、良いコーディングよりも良い設計が大きく影響する。

コーディング

- コーディングとは？
  - 開発に用いるプログラミング言語の決定
  - モジュール設計及び詳細設計に従って，ソフトウェアをプログラミング言語で実現する.
- コーディングの手順
  - プログラムの宣言部を書く。
  - アルゴリズムの各行を高水準のコメントとする。
  - それぞれのコメントの下に対応するコードを埋め込む。
  - アルゴリズムとコードが一貫しているか確認する。
  - 以上の作業を確認する。

コーディングの重要性

- 素直なコーディングはデバッグ・保守に要するコストを最小化する。
  - ソフトウエア開発のライフサイクルでは、エラー除去に最も時間がかかる。
- 素直なコーディングは高速化をもたらす。
- 高水準プログラミング言語、最適化コンパイラ、最新ハードウエアの組み合わせは、アセンブリ言語プログラムよりも高速

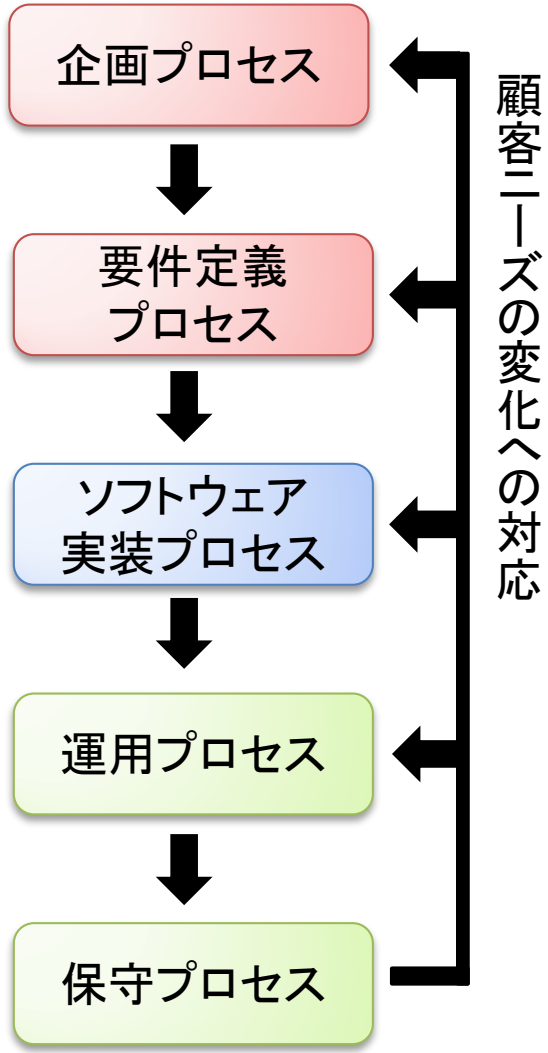
ソフトウエアテストの目的

- ソフトウェアテストとは？
  - ソフトウェアに入力データ（テストケース）を与えて実行する． → 実行結果A
  - 仕様書に基づく実行結果 → 実行結果B
  - 実行結果AとBを比較 → 一致しない場合，バグ発見
- 意味のあるソフトウエアは必ずバグを含む。
  - ソフトウエアにバグ（誤り）がないことは、一般には示せない。
- 運用前にできるだけ多くのバグを発見して修正するためにテスト等を行なう。
  - ソフトウエアにバグがないことを示すために行なうのではない。

運用・保守

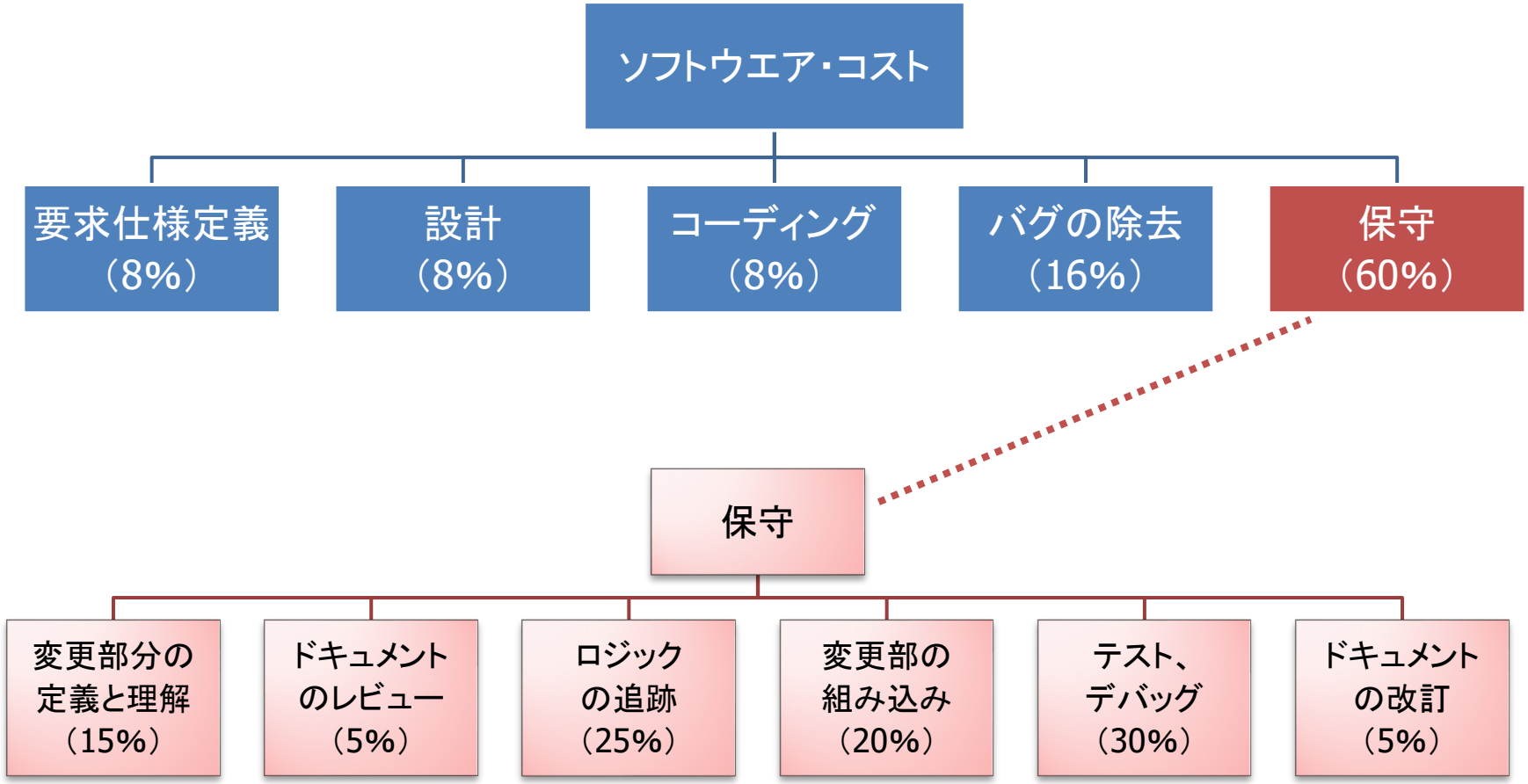
- 運用
  - 納入，インストール，初期教育等
  - システム稼動時のQ and A
- 保守
  - バグ修正
  - 仕様変更(機能拡張)への対応
  - バージョンアップ
  - 保守契約に基づいてサポートを実施する.

ITシステムは生きもの。作って終わりではない。顧客との取引が継続する限り、または事業や業務が続く限り(ITシステムを必要とする限り)、ライフサイクル全般を考慮して企画や要件定義を継続的に改善することが、結局は、適正コストで「使えるシステム」を実現できる。



出典:IPA SEC, 経営者が参画する要求品質の確保
～超上流から攻めるIT化の勘どころ～ 第2版, 2006.

ソフトウェア・コストの内訳



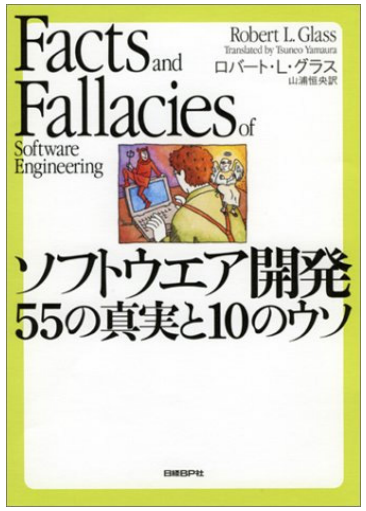
保守の重要性

- 保守にはソフトウエア・コストの40～80%（平均60%）が必要。
  - 保守はソフトウエア開発の最重要工程
- 保守の60%は機能拡張
  - 環境適合（18%）、バグ修正（17%）、その他（5%）
- 保守と開発の違いは既存プログラムの理解
  - 既存プログラムの理解には保守作業時間の約30%が必要
  - 理解しやすいシステム設計・コーディングが不可欠

参考図書

IPA SEC編，共通フレーム2013 ― 経営者、業務部門とともに取組む「使える」システムの実現，2013年3月，1500円

IPA SEC編，実務に活かすIT化の原理原則17ヶ条 ～プロジェクトを成功に導く超上流の勘どころ～，2010年10月，477円
無償ダウンロード可



ロバート・L・グラス著，山浦 恒央 訳，ソフトウエア開発：55の真実と10のウソ，日経BP社，2004年4月，2200円

IPA SEC編，経営者が参画する要求品質の確保 ～超上流から攻めるIT化の勘どころ～ 第2版，2006年5月，1714円
無償ダウンロード可