第６章
内部設計とプログラミング

内部設計工程とは

- 作成しようとしているシステムが外部仕様を満たしているかを検証し、確認をとる工程が内部設計である。プログラム自身の設計ミスを未然に防ぐのもこの工程である
- 内部設計では、お客様との合意が得られた外部設計書をもとに、ソフトウェアの具体的な実現方法について検討を行なう
- 内部データやクラスを定義し、サブシステムをモジュールへと分解し、機能仕様を満たすようにモジュールを設計する
- 自動車の設計では、部品を設計する段階に該当する。生産性を高め、コストを抑えるために、プログラミングでも部品を共通化したり、部品数を削減したりするが、その設計を行なうのも内部設計である

図6.1

内部設計がなぜ重要なのか

- モジュールを組み合わせた後に訂正を行なうと多大なやり直しが発生し、ムダな労力とコストを要するため、内部設計をしっかり行なうことが全体のコスト低減につながる
- ソフトウェア（プログラムやドキュメント）は複数のメンバーで構築することが一般的であるが、個人が勝手に作成したモジュールを組み合わせることでは、効率的にソフトウェアを作成できない。このため、どのような方針や規範で作るかを内部設計の作業開始時に厳密に定義し、意識合わせを行なうことが重要である

図6.2

内部設計書の作成手順

1. 内部設計書の作成手順と方法を意識合わせする
   用語、命名規則、各種ドキュメントの作成方法（プログラミング規約など）を定める
2. 内部設計書を作成する
   共通モジュールの洗い出しから行なう。機能の実現方法について検討を行なう
3. 内部設計書のチーム内デザインレビュを行ない、レビュでの指摘事項を記録する(2回目以降のレビュでは訂正部分を中心にレビュを行なう)
4. レビュでの指摘事項をもとに内部設計書を訂正する
5. 訂正がなくなるまで、２～４を何度も繰り返す

図6.3

内部設計書の構成

- **内部設計書作成方針**：作成するものは何か、どのような基準、環境、ツール、手順で作成するのかを定める
- **モジュール設計書**：機能を実現するためのモジュール構成図、各モジュールの処理手順（フローチャートなど）を設計する
- **内部データ、クラス設計書**：内部データの形式、モジュール間でのデータの受け渡し方法などを定める
- **通信プロトコル設計書**：ネットワーク経由で端末を接続する場合、データの交換方式（プロトコル）を定める
- **テーブル設計書**：データベースのテーブル設計書。データの正規化を行なう
- その他：必要に応じてその他の設計書（セキュリティ設計書など）を作成する

図6.4

プログラミング工程の進め方

- 工程の目的
  この工程では内部設計で作成された仕様にもとづいてソースコードを作成する
- 工程の概要
  この工程では、ソースコードの作成、コンパイルエラーの除去、単体テストの実施とバグ対処を行なう。作業自体は個人作業となる。このためソフトウェア業界では他の会社に発注（外注）する場合もある

図6.6

ソースコードの作成手順

1. ソースコードの作成手順と方法を意識合わせする
   用語、命名規則、各種ドキュメントの作成方法（プログラミング規約など）を定める
2. ソースコードを作成する
   共通モジュールの洗い出しから行なう。機能の実現方法について検討を行なう
3. ソースコードの部内レビュを行ない、レビュでの指摘事項を記録する（2回目以降のレビュでは訂正部分を中心にレビュを行なう）
4. レビュでの指摘事項をもとにソースコードを訂正する
5. 訂正がなくなるまで、２～４を繰り返す

図6.7

プログラミングでの品質作りこみの工夫

- 内部設計に携わった人がソースコードの作成にも参画する場合には、別の人が作成した詳細設計を担当させることでクロスチェックができる
- 同様に単体テストにおいても、テスト項目の作成、テストの実施、バグ対処（ソースコードの作成者）をそれぞれ別の人が行なう場合がある
- 業界ではこの工程の作業を外注会社に委託する場合も多く、作業の効率化を図るために準備（環境構築など）が重要である

図6.8

プログラミング工程の作業項目

- ソースコード作成における取り決めを作成
- 単体テスト実施手順書の作成
- ソースコード作成環境およびテスト環境の構築
- バグ管理の手順や報告様式を整え、管理ツールを導入
- 内部設計書をもとにソースコードを作成
- コンパイルエラーの除去
- ソースコードレビュの実施
- テストデータの作成
- スタブ、ドライバの作成
- 単体テストの実施と障害処理票の起票
- バグの分析

図6.9

プログラミングでの取り決めの例

- プログラミング規約（またはコーディング規約）
  - 変数名、クラス名、ファイル名などの命名規則
  - コメント、ヘッダ、著作権表示の記述方法
  - インデント、空白文字の入れ方　など
- 共通的なモジュールやクラスの設計
- クラスやメソッドとデータとの関係
- 共有フォルダの更新方法（CVSなどを用いる）
- テスト項目の選定方法、テスト実施方法
- 障害処理票の起票方法

図6.10

ソースコードレビュの実施方法

- ソースコードレビュには、ウォークスルーとインスペクションとがある
- ウォークスルー：ソースコードの起動から終了までの流れを追い、処理手順に誤りがないかを確認する方法。
  - 一般的にはソースコードレビュはウォークスルーにより行なう
- インスペクション：ソースコードの特定の部分を綿密にチェックし、決められた規約に基づいて作成されているかを確認する方法。

※　テストとは欠陥を発見する活動であり、デバグとは欠陥の原因を特定し、修復する活動であり、このため、テストとデバグとは本来区別すべきである

図6.11

ウォークスルーの実施方法

- レビュ参加者には事前にソースコードを配布しておく。参加者は、事前に配布されたソースコードに目を通しておく
- レビュ開始前に、レビュ記録表の記入者、レビュ時間を決める。レビュ時間は２時間以内とする
- レビュ参加者には、上司を含めないほうがよい。また、レビュ結果を人事評価に反映することは好ましくない
- ソースコードを作成した人がソースコードの説明をし、他の人からの指摘や質問を受ける

図6.12

単体テストのテスト項目の作成方法

- 単体テストはホワイトボックステストで行なう
- テストの項目は、内部設計書（詳細設計書）から作成する
- 作成されたモジュール（クラス）が、内部設計で記述された機能を満たしているかどうかをチェックする
- ソースコードのすべての分岐を通るようにテスト項目を作成する。判定条件を満たしているかをチェックする
- 項目数の目安は、経験的におよそソースコードの行数の5分の１程度である

図6.13

ホワイトボックステストとブラックボックステスト

- プログラムの制御構造を意識してテストすることを『ホワイトボックステスト』と言う。ホワイトボックステストではすべての分岐、すべての命令を実行するようにテスト項目を作成する
- 制御構造を意識せず、設計書の仕様にもとづいてテストを実施することを『ブラックボックステスト』と言う。ブラックボックステストでは、入力に対して設計書で定義された出力が得られるかどうかの確認を行なう

図6.14

テ　ス　ト　項　目　表

テスト区分　□単体テスト　□結合テスト　□総合テスト　□移行テスト　□運用テスト	管理番号　－　－　－	通番　／	作成者

テスト対象名(システム名、モジュール名)

通番	テスト予定日 担当者	テ　ス　ト　内　容	テスト実施日 担当者	テ　ス　ト　結　果	対　処　内　容

図6.15

スタブ と ドライバ

- スタブは以下のことを行なう
  - 呼び出されたことを表示する
  - 引数の内容を表示する
  - 戻値をセットしてプログラムを終了する
- ドライバは以下のことを行なう
  - モジュール実行に必要な引数をセットする
  - モジュールを実行する
  - 戻値の内容を表示する

図6.16

単体テストの実施手順

ソースコード
の作成者

①ソースコードの作成
②コンパイルエラーの除去

テスト項目の
作成者

③詳細設計書の内容から
テスト項目表の作成

テストの
実施者

④スタブ、ドライバの作成
⑤テストの実施
⑥障害処理票の起票
⑦障害管理簿に追記

障害対応
担当者

⑧障害内容の原因を調査
⑨障害処理票の対処内容を記載
⑩障害対処（ソースコードの修正など）
⑪障害管理簿を更新

管理者

⑫障害管理簿の内容を分析

図6.17

単体テストでのドキュメントの流れ

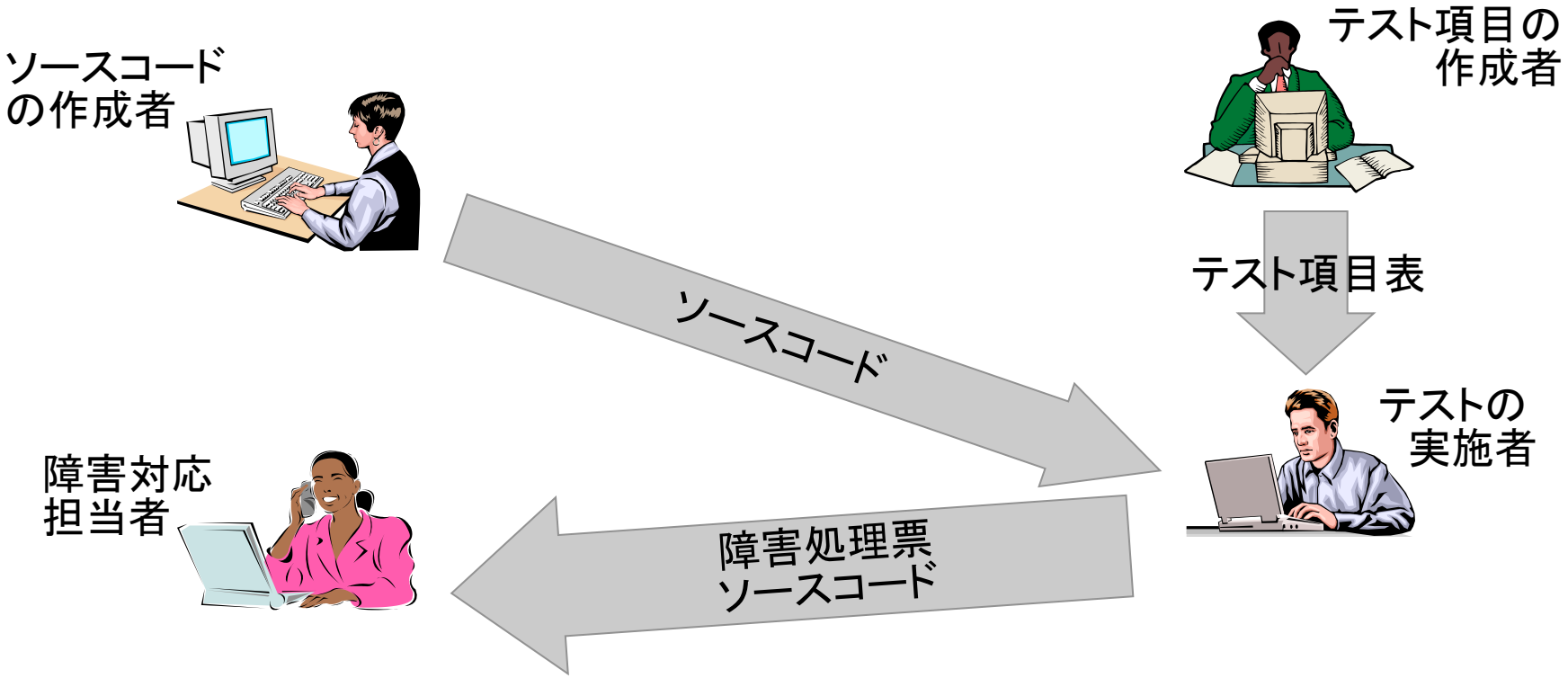


図6.18

障害処理票の起票方法

1. テスト実施者は、障害処理票を起票し、これをもとに、障害管理簿に新しい行を追加する
2. 障害対応担当者に障害処理票を配布する
3. 障害対応担当者が調査し、原因や判明したり、対応が決定したら、管理者に報告する
4. 管理者が障害管理簿に原因や対応内容を記入する
5. 障害対応担当者が障害処理票の障害処理内容を記入して、管理者に提出する
6. 定期的な会議で障害管理簿の障害状況を確認する

障　害　処　理　票

障害発生時刻	年　月　日(　)　時　分	管理番号	- - -
テスト工程 □単体テスト □結合テスト □総合テスト □移行 □運用		テスト項目管理番号	通番
システム名	モジュール名	起票者	

V	トラブル分類	V	バグ混入工程	V	検出すべき工程	V	検出遅延理由
	1.設計バグ		1.要求分析		1.要求分析		1.設計レビュー漏れ
	2.製造バグ		2.システム提案		2.システム提案		2.仕様書訂正漏れ
	3.改造バグ		3.外部設計		3.外部設計		3.仕様書解釈誤り
	4.DB、OSバグ		4.内部設計		4.内部設計		4.ソースコードレビュー漏れ
	5.環境、HWバグ		5.製造		5.製造		5.テスト漏れ
	6.手順バグ		6.単体テスト		6.単体テスト		6.改修誤り
	7.提供データ誤り		7.結合テスト		7.結合テスト		7.ユーザー調整漏れ
	8.誤操作		8.総合テスト		8.総合テスト		8.ユーザー調整誤り
	9.		9.移行		9.移行		9.仕様変更
	10.		10.運用		10.運用		10.杜撰なバグ管理
	11.その他		11.その他		11.その他		11.その他

障害状況

障害対処内容

対処者

対処日時　月　日　時　分

確認者

確認日時　月　日　時　分

備考(関連項目に対する対策、次工程への申し送りなど)

図6.19

第7章
結合テスト、総合テスト、品質保証の実施方法

テスト工程の進め方

- テスト工程では、『結合テスト』と『総合テスト』を行なう
- 結合テストでは、単体テストが終了しているモジュールを組み合わせ、外部設計書で定義されている機能が実現できているかどうかを確認する
- 総合テストでは、システム提案書や基本設計で定義された、性能、障害回復機能、過負荷時のふるまいなどが当初の設計どおりであるかどうかを確認する

図7.1

設計とテストとの関係

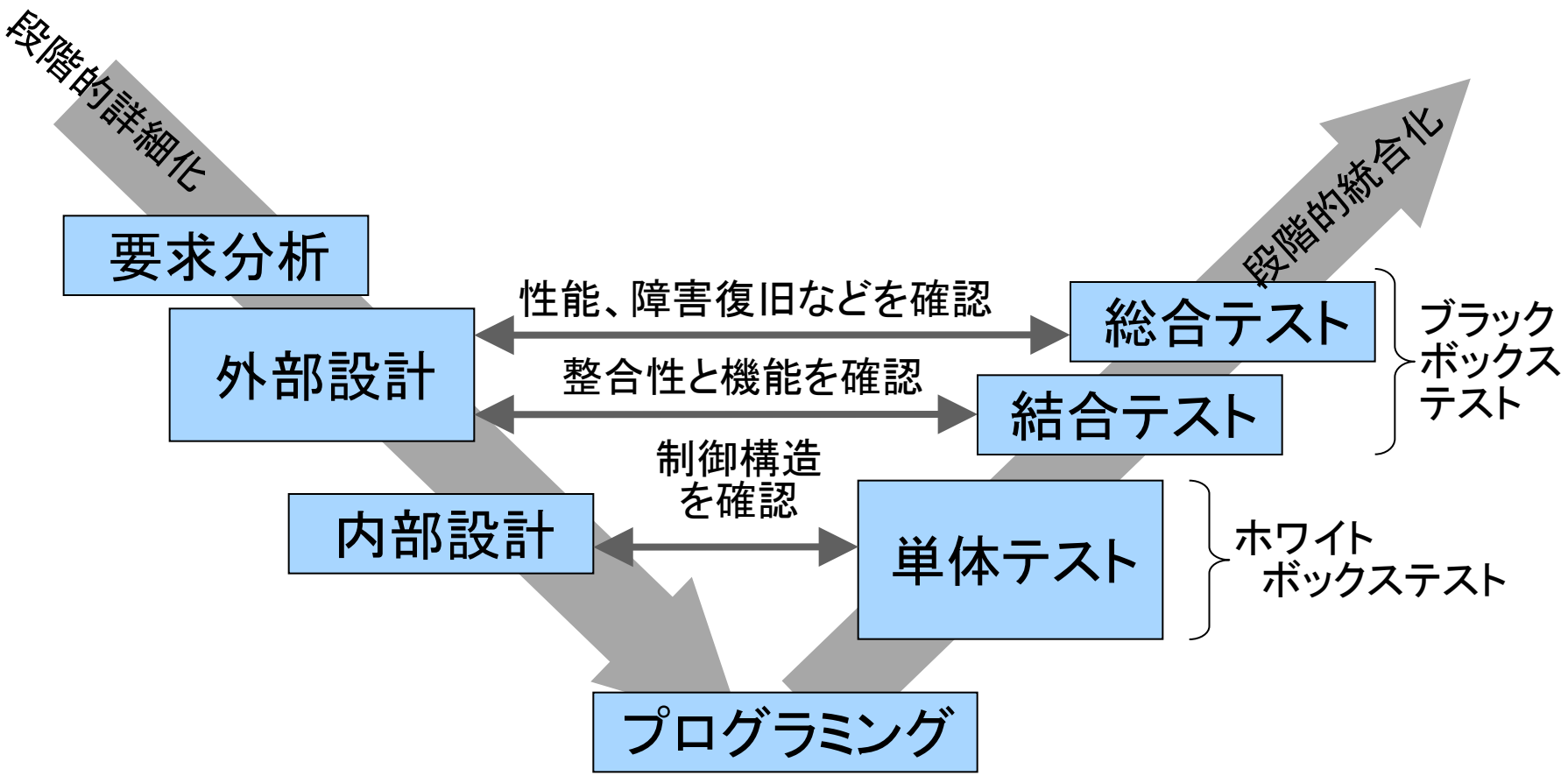


図7.2

結合テストの作業項目

- テスト計画書の作成
  - テストスケジュール、テスト体制、テスト手順
- テスト環境の準備
  - テスト用マシン、実マシン、擬似呼び出し発生装置　など
- テスト項目の作成
- スタブ、ドライバの作成
- テストの実施
- バグの除去、バグの分析

図7.3

トップダウンテストとボトムアップテスト

- トップダウンテストでは、最上位モジュールから結合テストを行なう。この場合、下位モジュール（図のsub1、sub2）の擬似的な動きをするためのスタブを用意する
- ボトムアップテストでは、最下位モジュールから結合テストを行なう。この場合、上位モジュール（図のmain）の擬似的な動きをするためのドライバを用意する

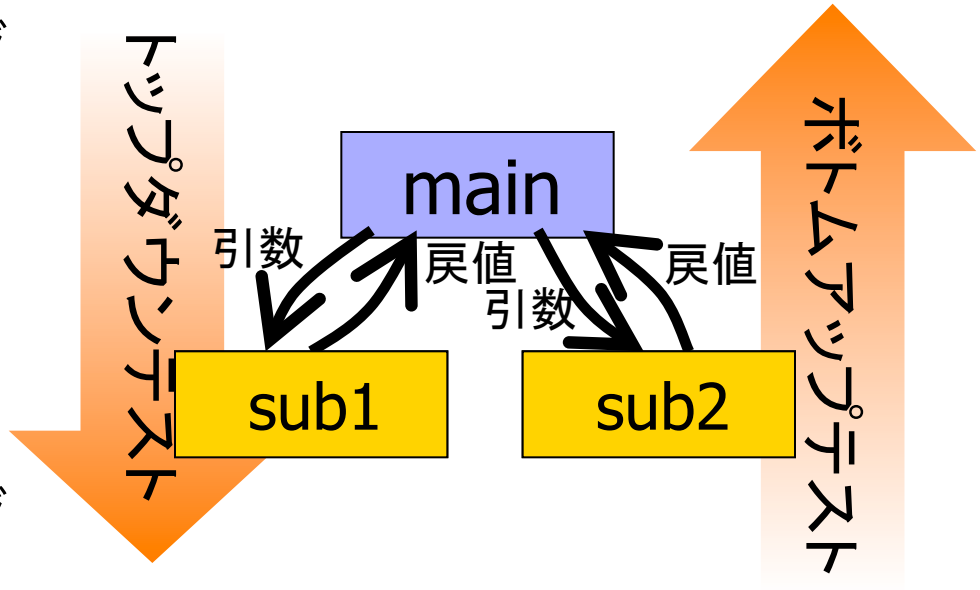


図7.4

品質作りこみの工夫

- 単体テストと同様に結合テストにおいても、テスト項目の作成、テストの実施、バグ対処（ソースコードの作成者）をそれぞれ別人が行なうようにすることでクロスチェックができる

結合テストのテスト項目作成方法

- 結合テストはブラックボックステストで行なう
- テストの項目は、外部設計書から作成する
- 外部設計書で定義された機能を満たしているかどうかをチェックする
- 項目数の目安は、経験的におよそソースコードの行数の20分の１程度である

図7.5

総合テストの実施項目

- ストレステスト
  過負荷状態におけるシステムのふるまいを確認
- 性能テスト
  応答時間や処理速度が規定内かどうか
- 回復テスト
  障害が発生した場合の回復機能の動作確認
- 操作テスト
  オペレータの誤操作に対するふるまいや、操作マニュアルの通りに実施ができるかどうかの確認

図7.6

品質保証の方法
バグの数量に関する分析

- テスト工程で見つかったバグの数量を計測し、システムの品質を予測する
- 通常は、テストが進むにしたがってバグの発生が減少し、累積バグ数は成長曲線を描く。この曲線の状況を見て、品質が高まってきたかどうかを判定する
- 累積バグ数の曲線を成長曲線に当てはめることで、収束値、すなわち総バグ数と残存バグ数を予測することができる。

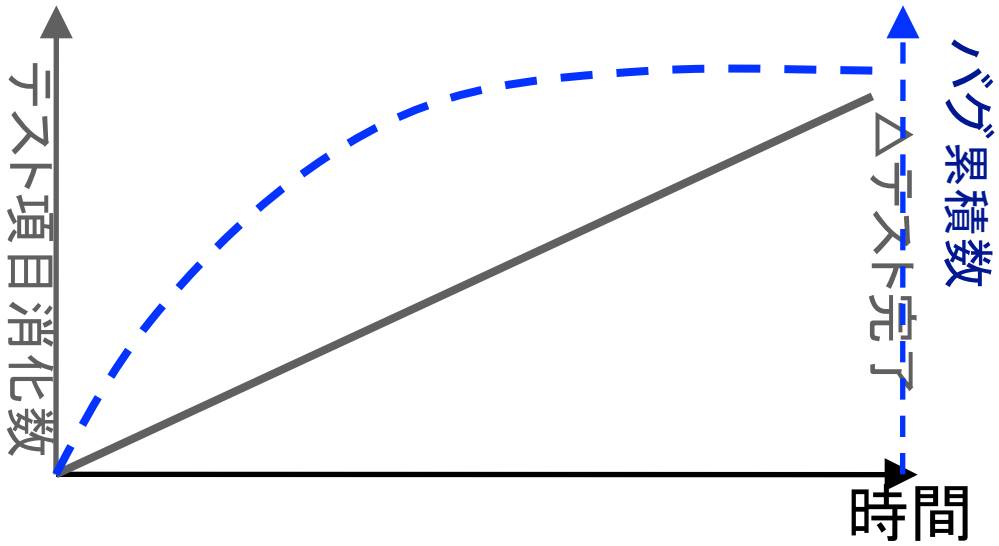
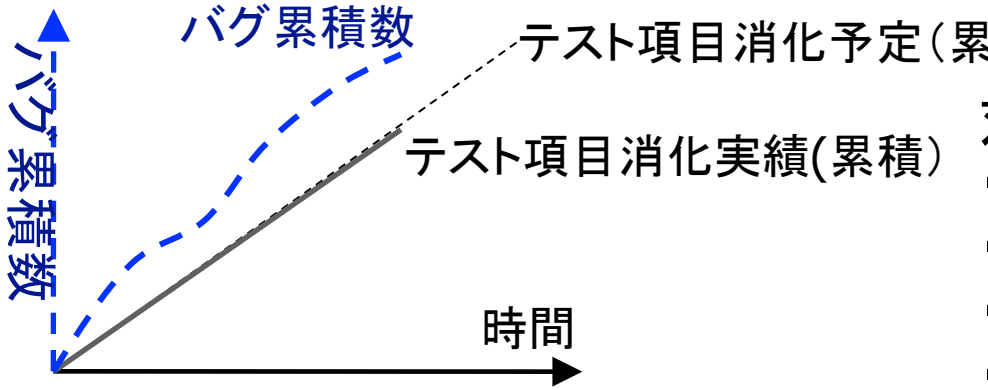


図7.7

バグ累積曲線の傾向と対策

事例1



対策
- プログラム品質の見直し
- テスト項目の追加
- テスト項目の見直し
- 再テストの実施

事例2

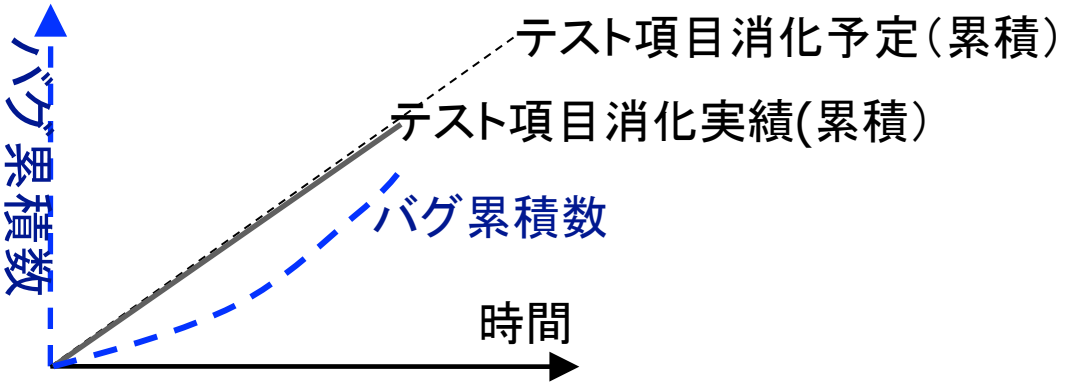
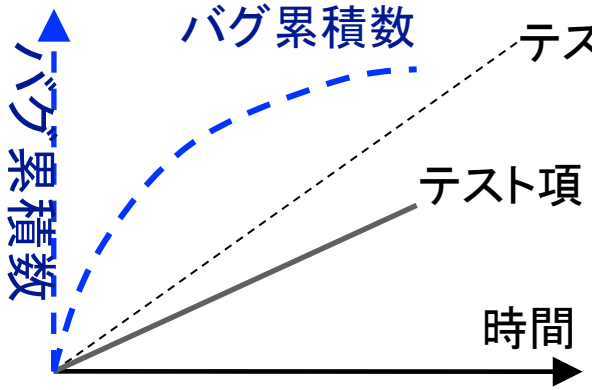


図7.8

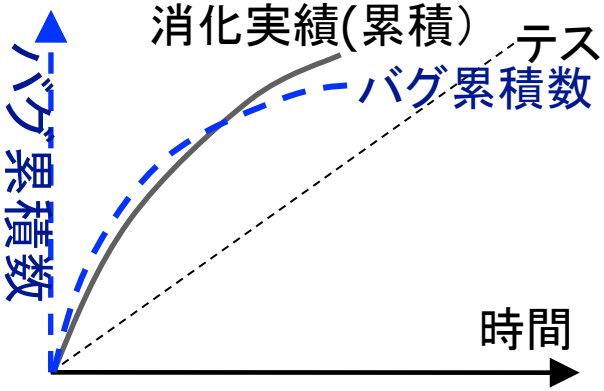
バグ累積曲線の傾向と対策

事例3



対策
- バグ予測値の再検証
- バグ内容の分析
- テストの続行

事例4



対策
- テスト項目の追加
- テスト項目の見直し
- 再テストの実施

図7.9

品質保証の方法
バグの質に関する分析

- テスト期間が短い場合やプロトタイプ設計などの場合、バグ累積曲線が成長曲線を描かない場合もある。この場合には、バグ数ではなくバグの内容により品質が良いかどうかを評価する
- バグの内容で、バグの混入工程、摘出すべき工程を分析し、前工程の作業が不十分だった場合には、前工程の作業の見直しなどを行なう

図7.10

品質見解の目的

- 品質見解の目的は、バグの傾向を分析し、構築したシステムの品質が高いことを証明することである
- 読者は、品質保証部門の人。品質保証部門は、出荷に向けての最終検査の際に品質見解を参照する

図7.11