

製造工程とプログラミング

60151039 東 裕之輔

資料

- ▶ ずっと受けたかったソフトウェアエンジニアリングの新人研修 第7章「製造」
- ▶ 監修
宇治則考
- ▶ 著者
 - ・大森久美子
 - ・岡崎義勝
 - ・西原 琢夫

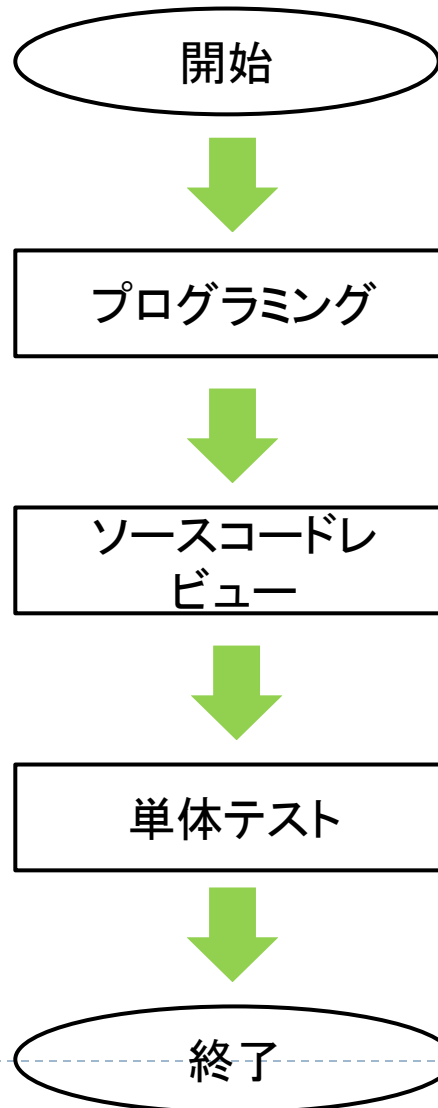


製造工程とプログラミング

- ▶ 設計が終われば次は実際にプログラムを作ります。
- ▶ プログラムを作る工程を「製造工程」と呼ぶ。
- ▶ 製造工程では内部設計書の内容に沿ってそれを実現するプログラムを書き、記述内容をチェックし、内部設計書どおりに動作するか確認する。



製造工程の作業手順



プログラミング

最初のプログラミングは、文字通りプログラムを書く作業。

内部設計書に従ってCやJavaで書く。

- ▶ 関数
- ▶ プログラムの名前
- ▶ 受け取るデータの数と種類、内部で行うべき処理、
- ▶ 呼び出し元へ返す処理結果

これらはすでに内部設計の定義に合致するように書く必要がある。



ソースコードレビュー

- ▶ 参加者全員でプログラムのソースコードを読み、次の点をチェックする。
 - ▶ (1)規約に沿って書いてあるかどうか
 - ▶ (2)正しいロジックで書いてあるかどうか
- ▶ (1)はコーディング規約のことであり、後に説明する。
- ▶ (2)はプログラムの流れを追うことで、機能が実現されているかどうか。



ソースコードレビュー

- ▶ ちなみに、プログラムの流れを追いながら、頭の中で動作をシュミレーションすることをトレースという。
- ▶ 他人が書いたプログラムをレビュー、トレースする、。自分でプログラムを書く以上に、プログラミングスキルが身につくと言われている。
- ▶ レビューはそのための良い機会なので、積極的に参加しよう。



コーディング規約

- ▶ コーディング規約とは、プログラマ全員が守るべき、プログラムの書き方を定めたものである。
- ▶ プログラミング言語ごとに定めるのが一般的。
- ▶ プログラムは文芸作品ではないので、個性は必要ない。
- ▶ コーディング規約を遵守することで、複数のプログラマが手分けしてソースコードを書く場合でも、記述形式が統一され、プログラムの保守性が高まる。



実際のコーディング規約

- ▶ 実際の規約では次のようなルールを定める。

- ・プログラミングの冒頭に付けるコメントの記述ルール
- ・変数名の付け方や宣言方法に関するルール
- ・プログラムロジックの記述ルール
- ・変数の型のルール

- ▶ 読みやすいプログラム、分かりやすい名前、同じコードが重複していない、役割は一つであること、と初心者にもわかりやすいルールを具体的に定める。
-

コーディング規約の例

▶ Javaのコーディング規約の例: コメントに関して

- 処理に関するコメントは可能な限り詳細に記述する。またコメントに記述する内容は、プログラミング言語での記述内容を日本語に翻訳するのではなく、その処理が持つ意味を明確に記述する

○ `dataCount = 0; /* データ数カウンタを初期化する */`
× `dataCount = 0; /* dataCount を 0 にセットする */`

- クラスの冒頭にはそのクラスに関する概要を記述する。記述にあたっての形式や使用文字種は以下に従う

`/**`

`* クラスの説明 (日本語)`
`* @author 作成者名 (半角英字)`
`* @author 修正者名 (半角英字)`
`* @version バージョン番号 (半角英数字)`

`/*`

Javadoc

- ▶ Javaにはソースコードからリファレンスマニュアルを自動生成するJavadocというツールがあり、その書式でコメントを書いておくと、プログラムインターフェースに関する仕様書が自動生成される。
- ▶ このように使用するツールを意識しながらコーディング規約は定められていて、プログラムの保守性を高めている。
- ▶ 変数名の付け方に関しては、例えば次のような規約を設ける。



コーディング規約の例

▶ Javaのコーディング規約の例: 変数の命名に関して

- 変数名に使用する文字は、半角英字、数字、記号_ (アンダーバー)、記号- (マイナス) とする
- 変数名はその変数の持つ意味を英語で記述する。英字は小文字を使用するものとし、複数の英単語を連結する場合に限り、2 単語め以降の各単語の冒頭文字を大文字とする
 - `inputRecordCounter`
 - × `input_record_counter`
 - × `nyuuryokuRekoodoKaunta`
- 変数名の冒頭文字は必ず英小文字とする。数字や記号から始めてはならない
 - `firstField`
 - × `1stField`
 - × `_1st_field`

オフショア開発

- ▶ このように変数名などを英語で命名するのは、外国人との協働や。「オフショア開発」という海外でのシステム開発などを考慮しているため。
- ▶ この理屈でいけばコメントも英語のほうがいいかもしれないが、日本人の理解が不十分となる恐れがあるため、最小限、変数名だけに英語を用いる場合が多い。



単体テスト

- ▶ 単体テストとは関数や手続きといった最小単位のプログラムを単体で動かしてみて、内部設計書どおりに動くかどうかを確かめる製造工程の最後の工程である。
- ▶ 単体テストでは主にホワイトボックステストという形式で行われる。
- ▶ 内部構造が見えることを前提としたテストである。



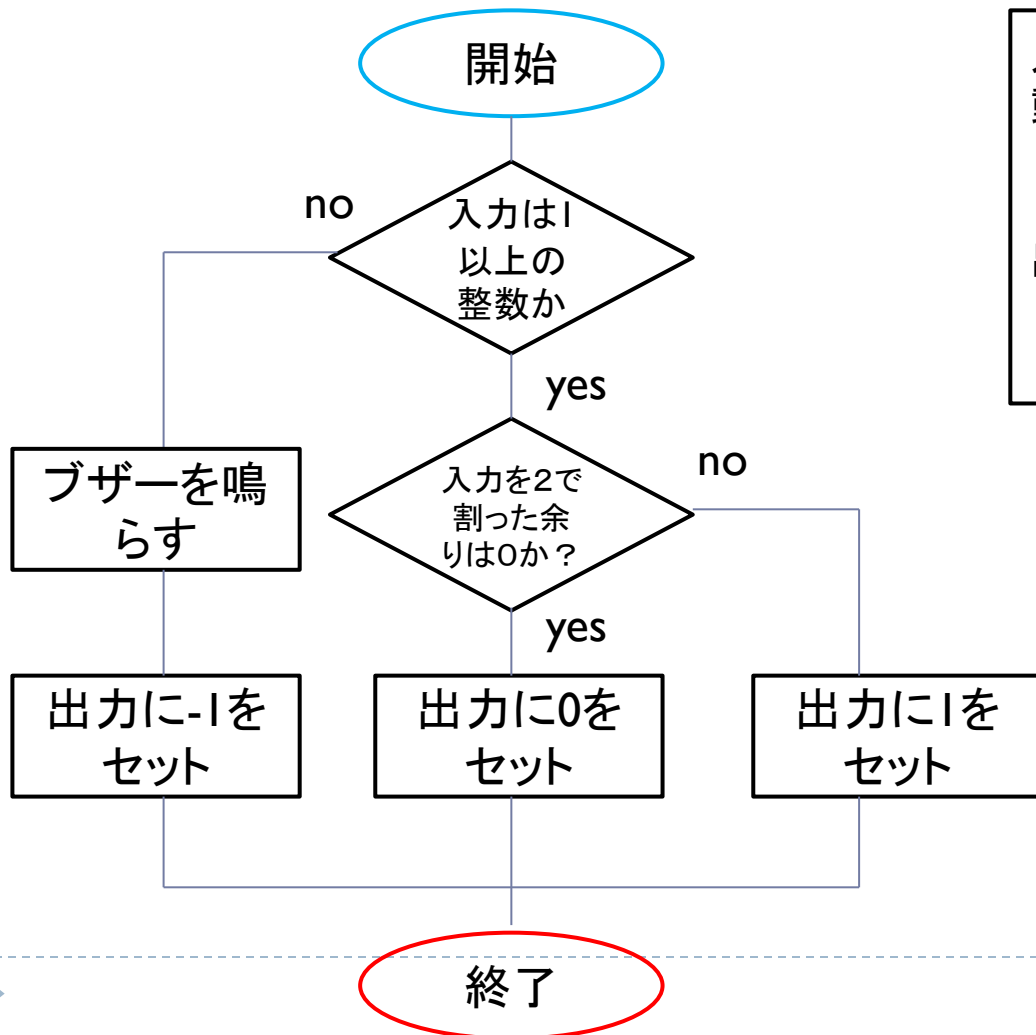
テストデータ

- ▶ まずテストを行う際、テスト項目を作成し、テスト項目に対応するテストデータを定める。年齢をチェックするテストでは20を入力するなら、20はテストデータになる。
- ▶ 単体テストはすべての分岐を通るようなテストデータを設定する。
- ▶ しかし全ての分岐パターンを探し出すのは困難
- ▶ フローチャートを基に探し出す。(フローグラフ図法)



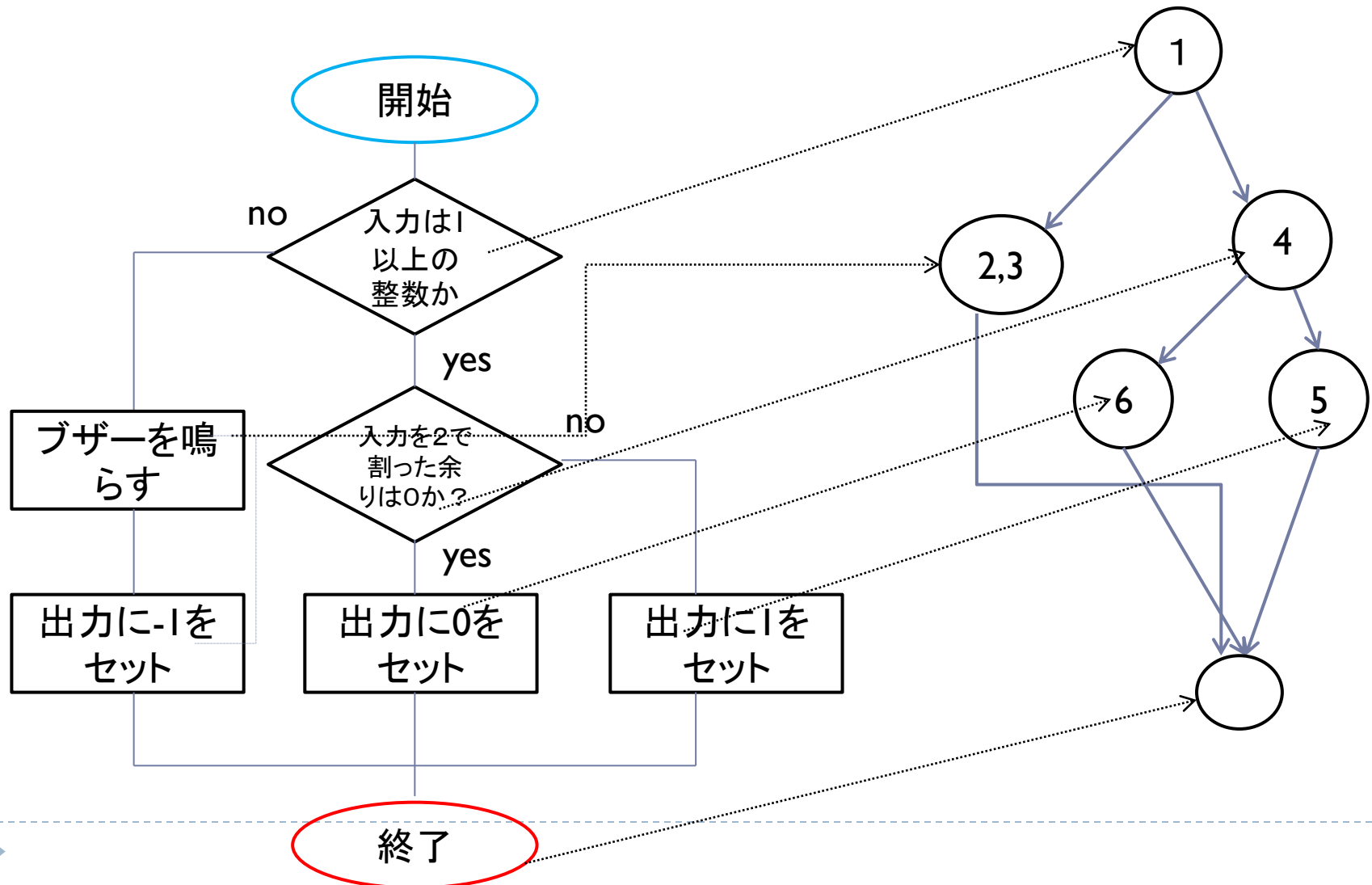
単体テストの対象プログラムのフローチャート

- ▶ 自然数(1以上の整数)の偶数か奇数かを判断するプログラム

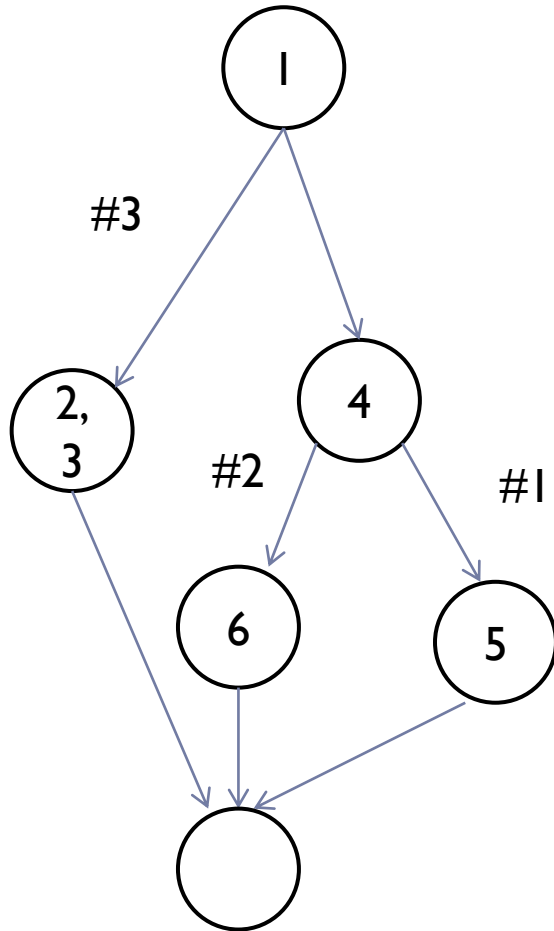


入力: 1つの整数。
動作: 自然数なら何も鳴らさない。
自然数でなければブザーを鳴らす。
出力: 偶数なら0。
奇数なら1。
自然数でなければ-1。

フローチャートからフローグラフへ変換



フローグラフからパス数を算定



$$\begin{aligned}\text{パスの数} &= \text{エッジ数} - \text{ノード数} + 2 \\ &= 7 - 6 + 2 \\ &= 3\end{aligned}$$

テストデータと通るパス

- ▶ パス数が3ならテストデータは3つあればよいことが分かる。

テストデータ	通るパス
1	#1・・・自然数かつ奇数
2	#2・・・自然数かつ偶数
0	#3・・・自然数ではない

- ▶ 言い換えればこの3つのデータに対してテストしてやれば、少なくともテストから漏れるような処理は発生しない。
 - ▶ 一般的にはテストデータの作成者とテスト実施者は分けたほうが正確なテストが実施できる。
-

