楽な方に流れることでテストしやすいコードが生まれる

テスト駆動開発で組み込みソフトウェアの品質を上げる

システムの記事　関連データ

細谷泰夫

テスト駆動開発（TDD：test driven development）は，eXtreme Programing（XP）という開発手法で提唱されているプラクティス（実現手段）の一つです．そう聞くと，「Javaなどには適用できても，組み込みソフトウェア開発には向いていないのでは？」と思う方がいるかもしれませんが，テスト駆動開発は，組み込みソフトウェア開発にも非常に有効な開発手法です．筆者自身も通信システムにおける組み込みソフトウェア開発にテスト駆動開発を適用し，大きな効果を実感しました．ここでは，テスト駆動開発の実践方法を具体的に紹介します．（筆者）

テスト駆動開発（TDD：test driven development）は，テストを実装の中心に据えたプログラム開発手法です．具体的には，実装対象であるプログラムのコードを書く前にテスト・プログラムのコードを実装し，テスト・プログラムに合格するように実装対象プログラムのコードを実装していきます（**図1**）．テスト・プログラムを少し書いては，対応する実装対象プログラムのコードを書くという作業を繰り返し，プログラムをあるべき姿に徐々に近づけていきます．

テスト駆動開発はよくテスト手法と間違えられますが，そうではなく，あくまでも開発手法です．テスト駆動開発で行う「テスト」は，コードを実装する過程において実装者がどのように考えたのかを整理した結果であり，いわゆる「単体テスト」とは別のものです．このことを理解した上で，単体テストを効率良く実施するための方法としてテスト駆動開発を活用することは可能です．なぜなら，単体テストとテスト・コードは多くの場合，ほとんど同じ内容となることが多いからです．テスト駆動開発のテスト・コードをレビューしたり，カバレッジ・ツールを活用してテスト・コードの網羅性を保証したり，テスト・コードを補完するテストを追加することなどによって，テスト駆動開発を単体テストに代えることも可能です．

まずはテスト駆動開発を実践し，有用性を理解してみてください．そうすれば活用方法はおのずと見えてくるでしょう．

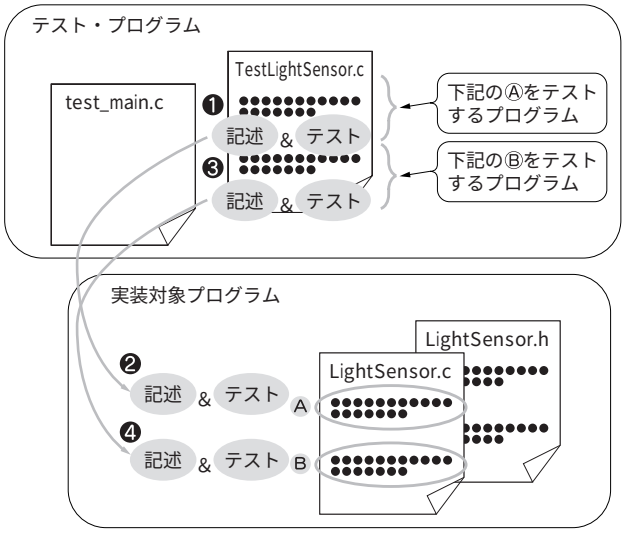

図1　テスト駆動開発の概要
まずテスト・プログラムのコードを実装し，そのテストに合格するように実装対象プログラムのコードを実装する，という作業を繰り返す．

● テスト駆動開発のために必要な環境

C言語でテスト駆動開発を実践するためのツールとして

KeyWord　TDD，開発手法，テスト，CUnit for Mr.Ando，リファクタリング，オブジェクト指向，ETロボコン

は，「CUnit」や「CUnit for Mr.Ando」[(1)(2)]というテスト・フレームワークが用意されています．CUnitのほうが有名で多機能，CUnit for Mr.Andoのほうがシンプルで導入しやすい，という特徴があります．本稿では，CUnit for Mr.Andoに多少変更を加えたものを使用します．

CUnit for Mr.Andoは，テスト用の関数testRunnerを含んだソース・ファイルとヘッダ・ファイルを備えています(**図2**)．テスト用のソース・ファイル(test_main.cやtestLightSensor.cなど)にコードを記述し，そのテストに合格するように実装ファイル(LightSensor.cなど)を記述していきます．

C言語でテスト駆動開発を実施するとき，各クラスのテストごとにスタブ(テスト用の疑似関数)が競合するという問題が起こるので，CUnit for Mr.Andoなどでは各クラス(光センサ，ライン・センサなど)の単体テストごとにコンパイル環境(Makefile)を分けることを推奨しています．筆者はスタブを外部から設定するインターフェースを実装することにより，コンパイル環境を分けずに(一つのMakefileで)テストできるようにしました．また，今回扱う例はシンプルであるため，**図2**に示したファイルを一つのフォルダにまとめることもできます(スタブの競合については，下掲のコラム「C言語によるテスト駆動開発の問題点」を参照)．

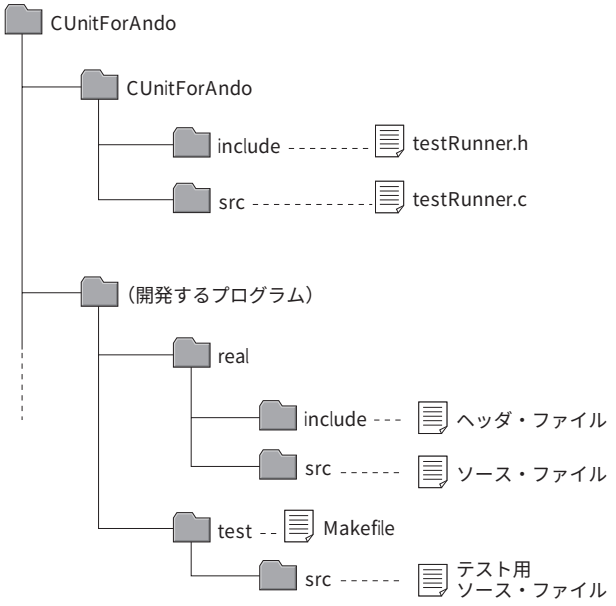

図2　CUnit for Mr.Andoのフォルダ構成例

テスト用ソース・ファイルに確認したい内容を記述する．テストそのものは，testRunner.cなどに定義されたテスト用の関数を利用して実施する．なお，本稿では，筆者がスタブを外部から設定するインターフェースを実装しており，コンパイル環境を分ける必要がないので，フォルダ構成はもっと単純にできる(例えば，すべてを一つのフォルダにまとめることも可能)．

● テスト駆動開発の基本的な流れ

まず，テスト・フレームワークを用いてテスト駆動開発を行うための基本手順について理解しましょう．ここでは，今から実装しようとしているクラスを「実装クラス」，実装クラスに対してテストを記述するクラスを「テスト・クラス」と呼びます．テスト駆動開発の基本的な流れは**図3**のようになります．

コンパイルが通らなかったり，テストが失敗したりする状態を「レッド」，テストがすべて成功している状態を「グリーン」，そしてプログラムの動作を変えずに構造を手直しすることを「リファクタリング」と呼びます．

通常のプログラミングでは，正しく動作しているコード

COLUMN - 1　C言語によるテスト駆動開発の問題点

C言語によるテスト駆動開発を可能とするテスト・フレームワークとしては，「CUnit」や「CUnit for Mr.Ando」があります．しかし，フレームワークはあるのに，C言語によるテスト駆動開発はあまり普及していません．

その理由の一つとして考えられるのは，言語として回帰テストを行うことが困難であることです．テスト駆動開発で，ある関数(クラスA)からある関数(クラスB)を呼ぶことをテストする場合，クラスAのテスト結果がクラスBの実装に影響されないように，クラスBの実体を呼び出すのではなくクラスBのダミーを用いることになります．JavaやC++などのオブジェクト指向言語では，クラスBをオーバライドすればよいのですが，C言語ではスタブ(テスト用の疑似関数)を用いてテストすることになります．このとき，スタブと実体の名称が2重定義となってしまうため，スタブと実体ではコンパイル環境を分ける必要が出てきます．

テスト駆動開発で重要なのは，「レッド－グリーン－リファクタリング」の軽快なリズムと，常に回帰テストが行えるからこそ可能なリファクタリングによるクラス設計の改善です．コンパイル環境がいくつにも分かれた状態では，これらを実現するのは難しいでしょう．

筆者が参画している日本XPユーザグループ(XPJUG)関西支部組込みTDD分科会は，C言語でテスト駆動開発を行うための具体的な手法を確立するために活動しています．スタブの競合に対しては，スタブを外部から設定するインターフェースを実装することにより，コンパイル環境を分けずに回帰テストができるようにする方法を提案しています[(3)]．この資料は本誌の付属CD-ROMにも収録しているので，参照してください．