組込み開発におけるテスト駆動開発事例

～C言語での適用事例～

07.7.25

XPJUG関西 細谷泰夫

XPJUG-Kansai
日本XPユーザグループ関西
http://www.xpjug.jp/

テスト駆動開発とは？

Test Driven Development

略して

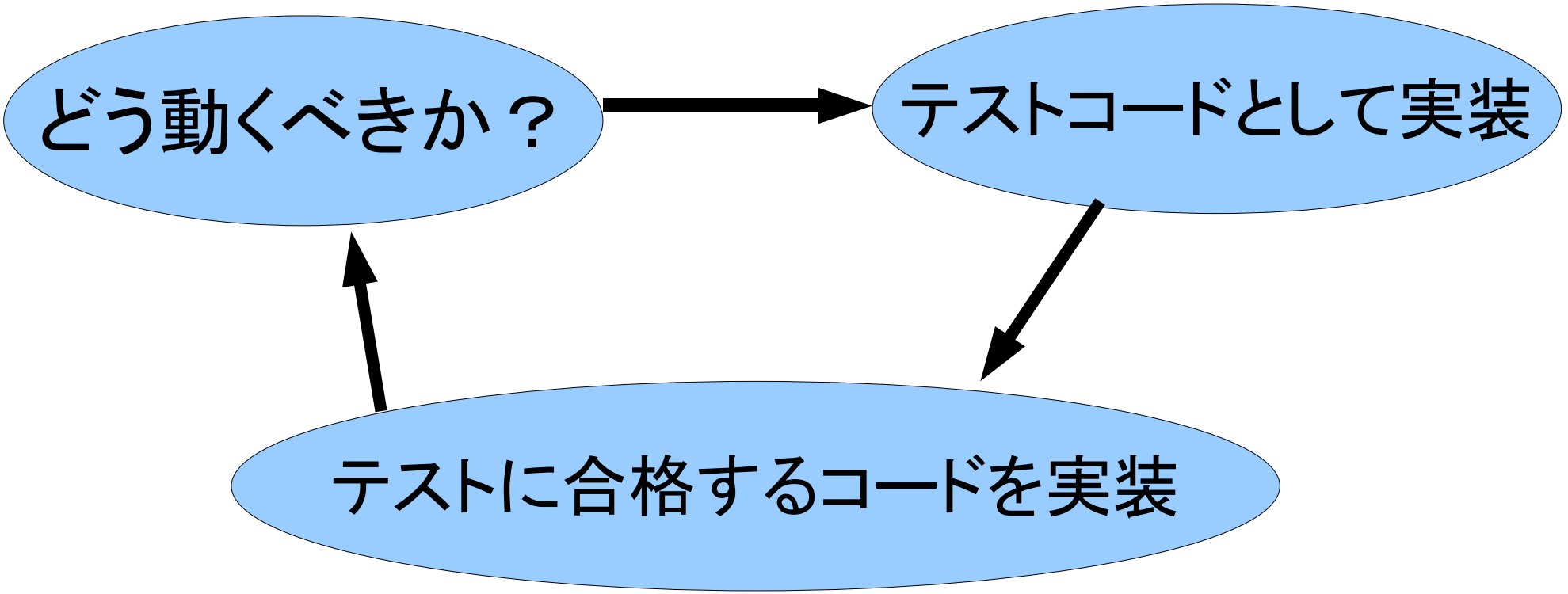
TDD

実装コード
より先に

テストコードを書く

つまり・・

テスト駆動開発(TDD)とは？

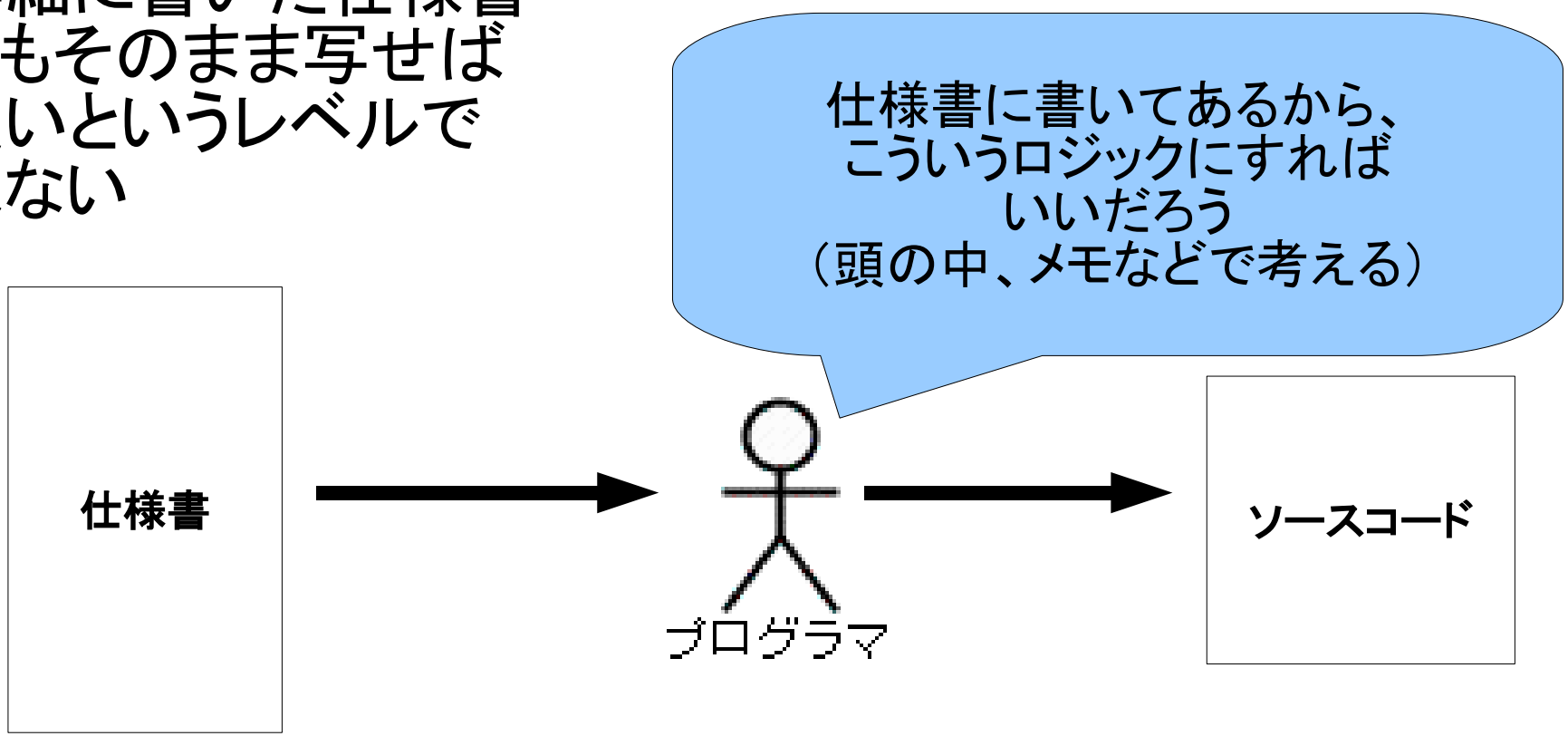


このサイクルを繰り返す開発手法

テストコード
ってなんだろう?

従来の開発では・・・

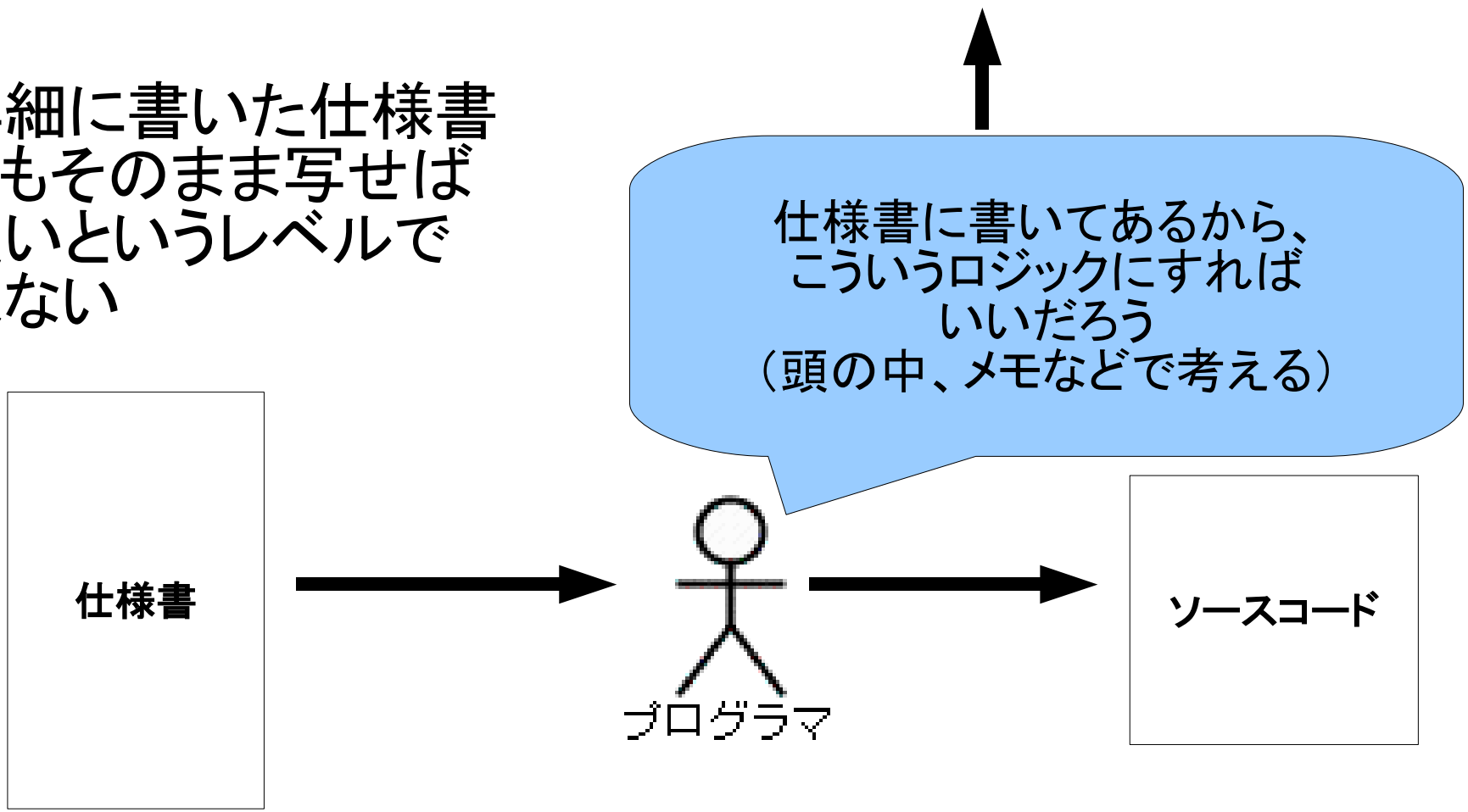
詳細に書いた仕様書
でもそのまま写せば
良いというレベルで
はない



従来の開発では・・・

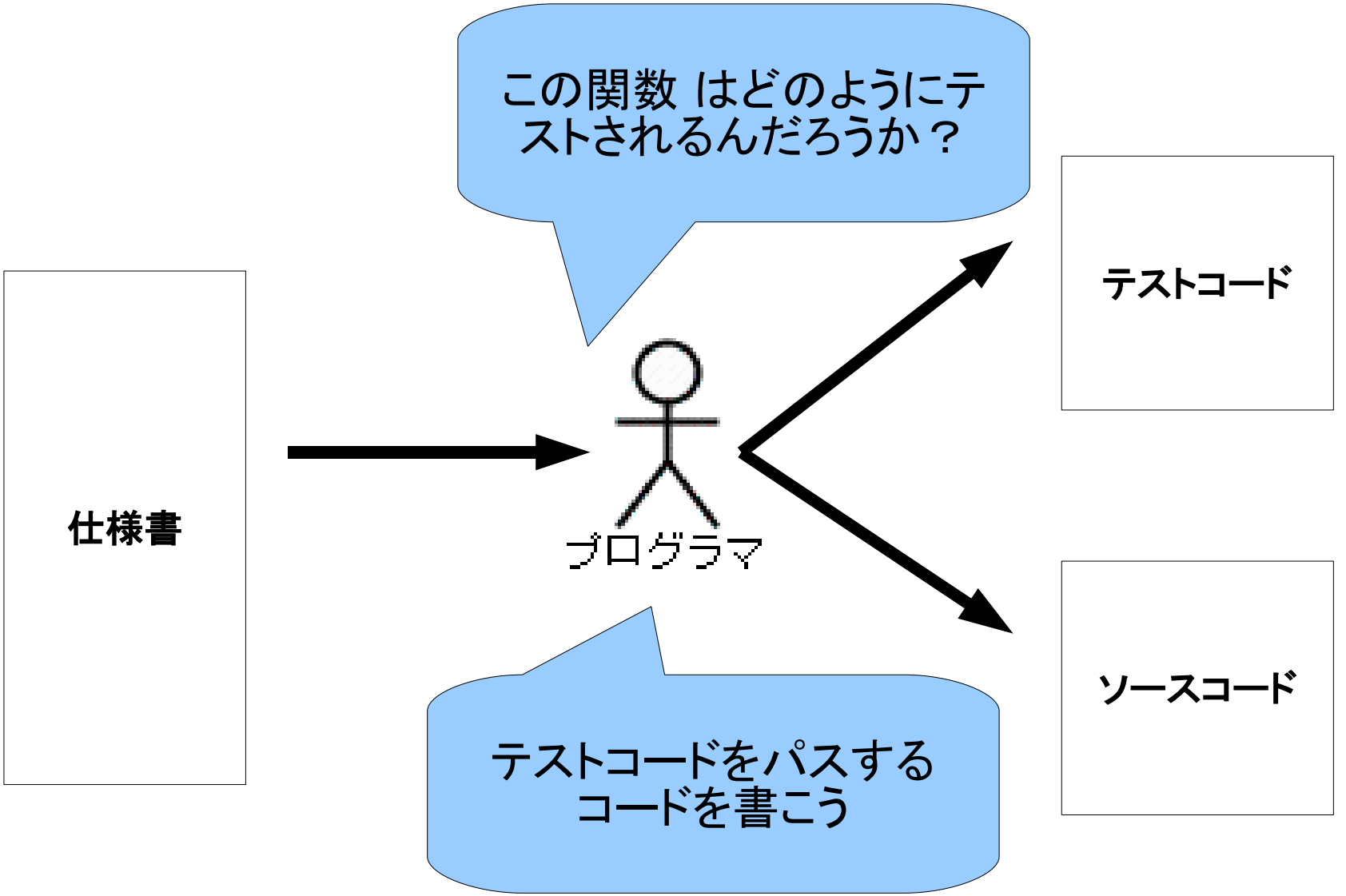
この部分がTDDのテストコード

詳細に書いた仕様書
でもそのまま写せば
良いというレベルで
はない



今まで頭の中やメモ書きで行っていた「最後の設計」をテストコードの実装として明確に実施する

TDDの場合



テストコードはプログラマの「テストの観点」です。

適用事例

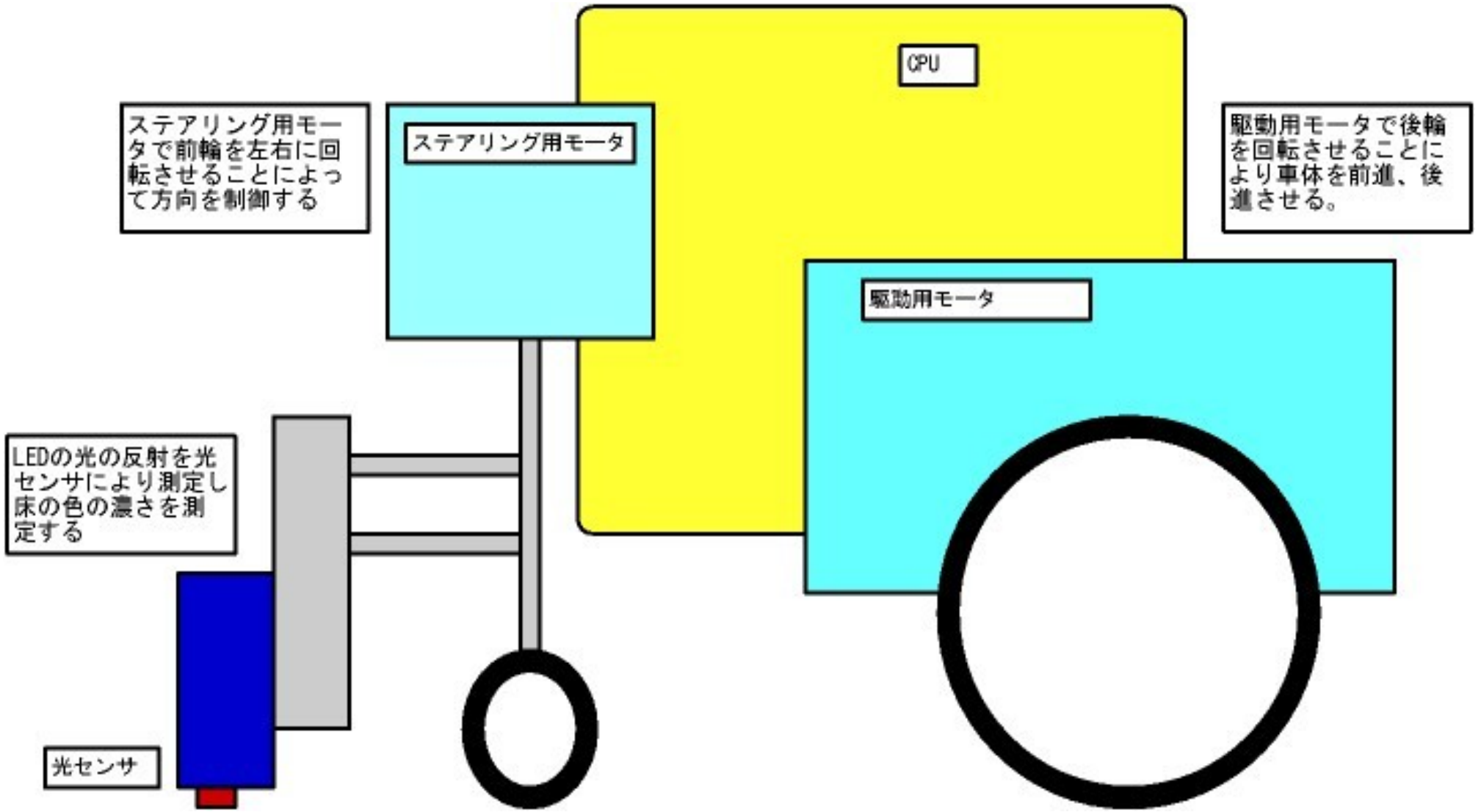
対象: ETロボコンにおけるライントレーサの開発

開発言語: C言語

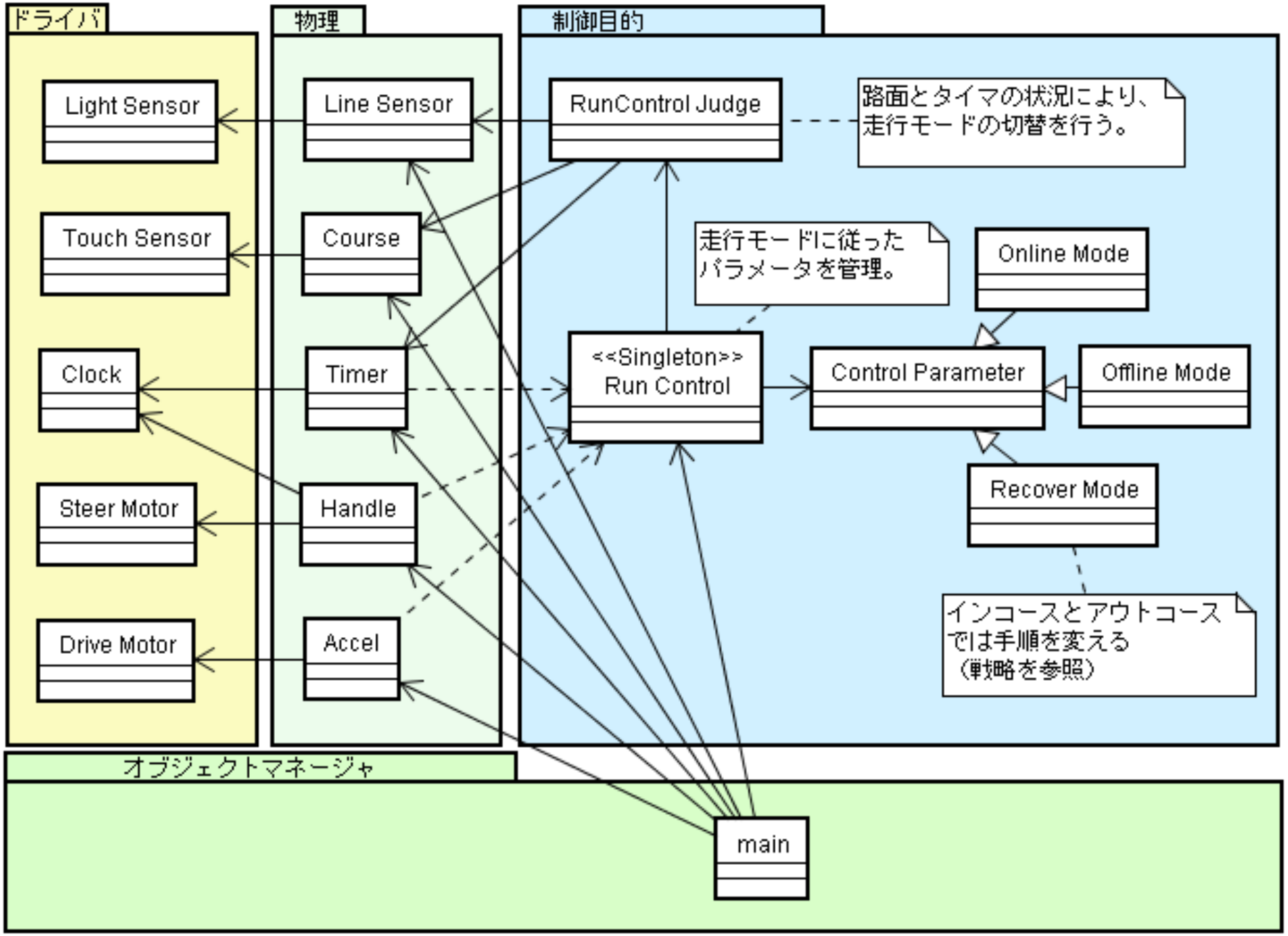
開発手法: 自律オブジェクト指向

開発規模: 3k step+テストコード3k step

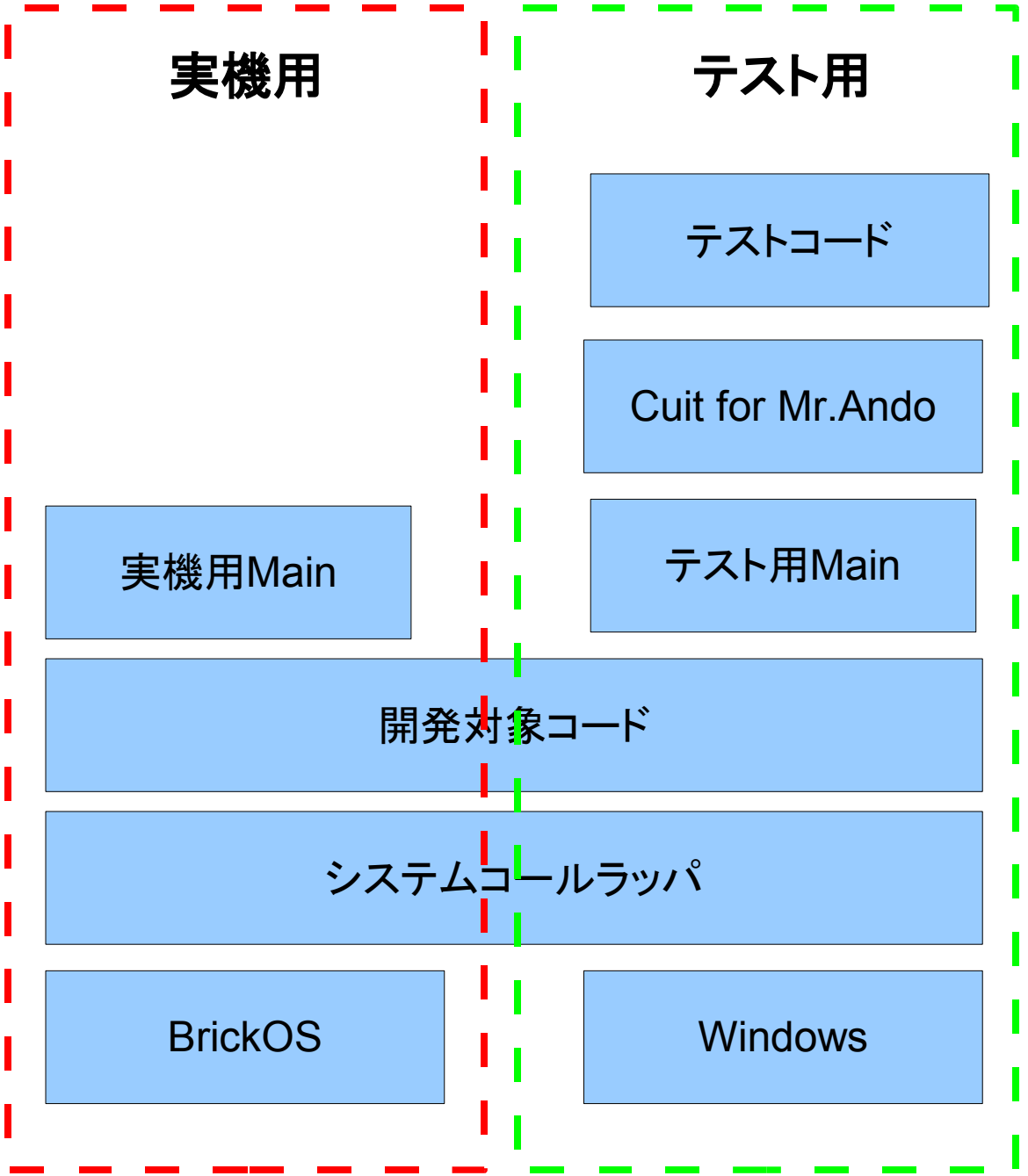
ライントレーサ（ETロボコン仕様）概要



クラス図


ドライバ
物理
制御目的
Light Sensor
Line Sensor
RunControl Judge
路面とタイマの状況により、
走行モードの切替を行う。
Touch Sensor
Course
走行モードに従った
パラメータを管理。
Online Mode
Clock
Timer
<<Singleton>>
Run Control
Control Parameter
Offline Mode
Steer Motor
Handle
Recover Mode
Drive Motor
Accel
インコースとアウトコース
では手順を変える
（戦略を参照）
オブジェクトマネージャ
main

開発環境



- 実機用はクロスコンパイラでビルド
- xUnitはCUnit for Mr.Andoを使用
- 開発対象コードは共通
- システムコールをラップしてプラットホームの差異を吸収
- テストプログラムはWindows上で実行

C言語におけるオブジェクト指向的実装

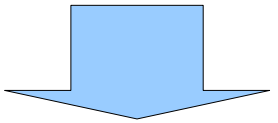
オブジェクト指向実装のメリット

データのカプセル化：　保守性、可読性の高いコード

クラス：データと振る舞いを1つの概念とみなす
→単体テストの単位として適切

今回採用した手法

ファイルアクセス構造体とファイル操作関数をお手本



クラスの型は構造体で定義
属性は構造体のメンバとして定義

外部から操作関数を介してメンバを操作する

コンストラクタ、デストラクタのテストコード例

テスト側

```
unsigned int CTestLineSensor_Setup(void)
{
    m_CTestLineSensor_p=CLineSensor_Constructor();
    TEST_ASSERT(NULL != m_CTestLineSensor_p);
    TEST_ASSERT(0 == m_CTestLineSensor_p->isOnline);

    return 0;
}
unsigned int CTestLineSensor_Teardown(void)
{
    CLineSensor_Destructor(&m_CTestLineSensor_p);
    TEST_ASSERT(NULL == m_CTestLineSensor_p);
    return 0;
}
```

POINT

(1)メンバ変数の初期値チェックテストクラスからはメンバ変数値を
テスト目的で参照しても良い

(2)ポインタ変数がNULLであることによりインスタンス削除を確認

(3)ポインタ変数がNULLで無いことによりインスタンス生成を確認

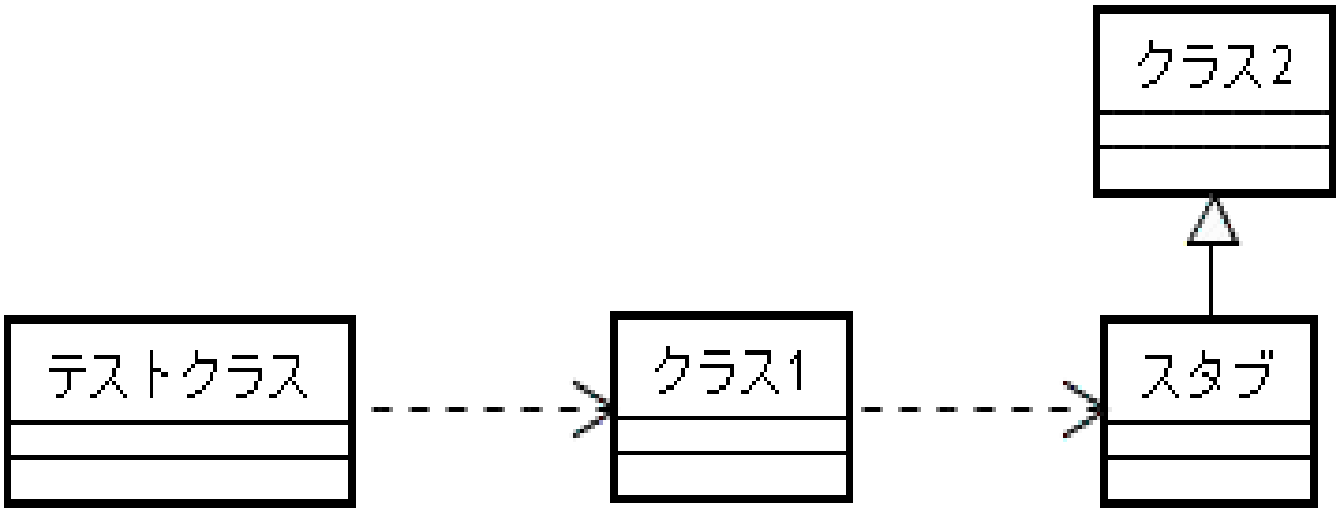
C言語でTDDを行う場合の問題点

スタブの競合

テスト対象のクラスが他クラスと関連付いている場合のテスト

C++、Javaの場合・・

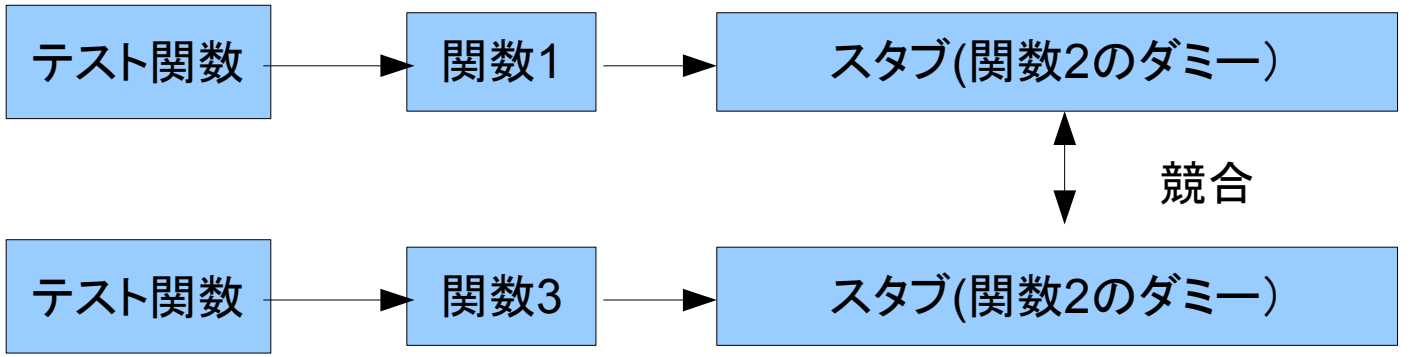
スタブ用のダミークラスを作成することにより単体テストが可能となる。



C言語でTDDを行う場合の問題点

C言語の場合‥

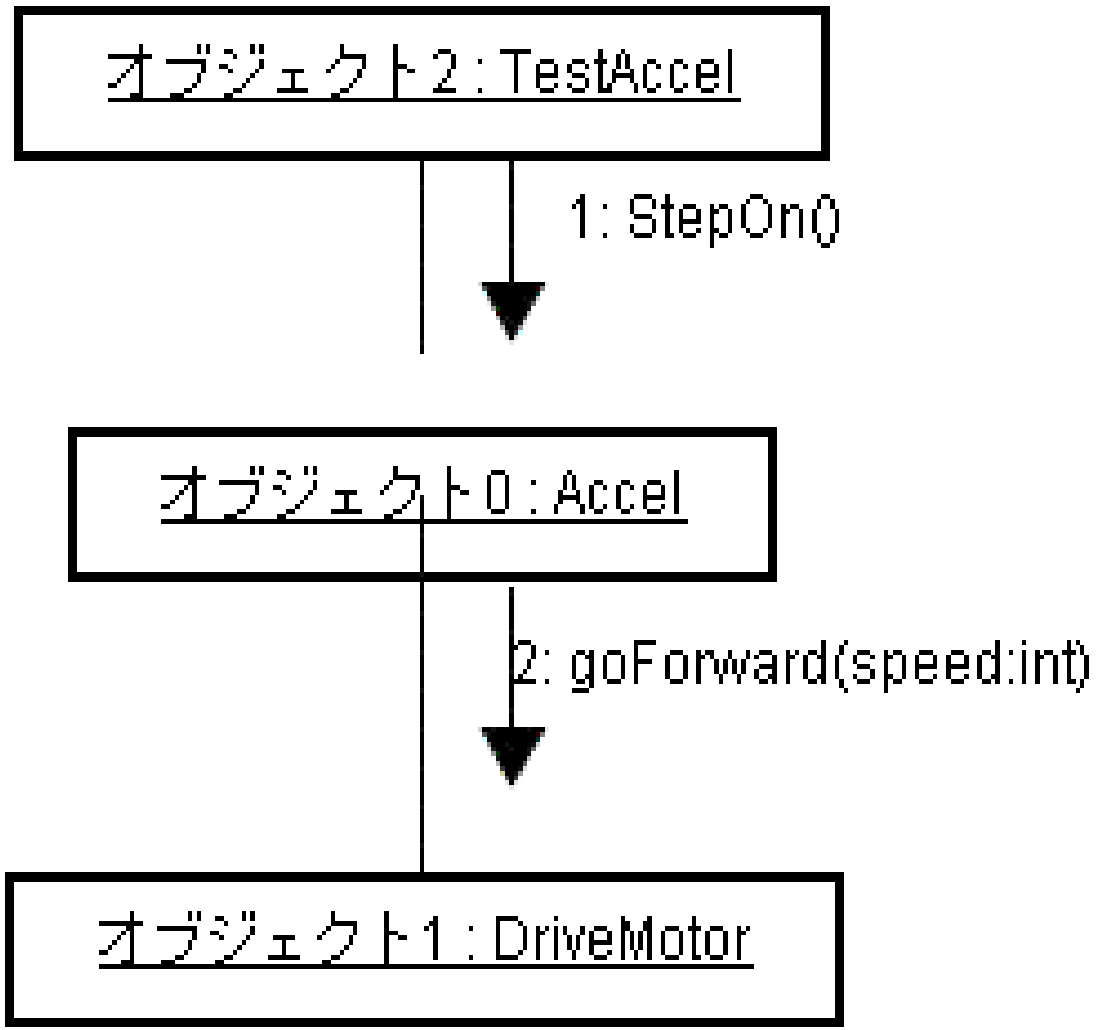
スタブ関数を用いて単体テストを行う



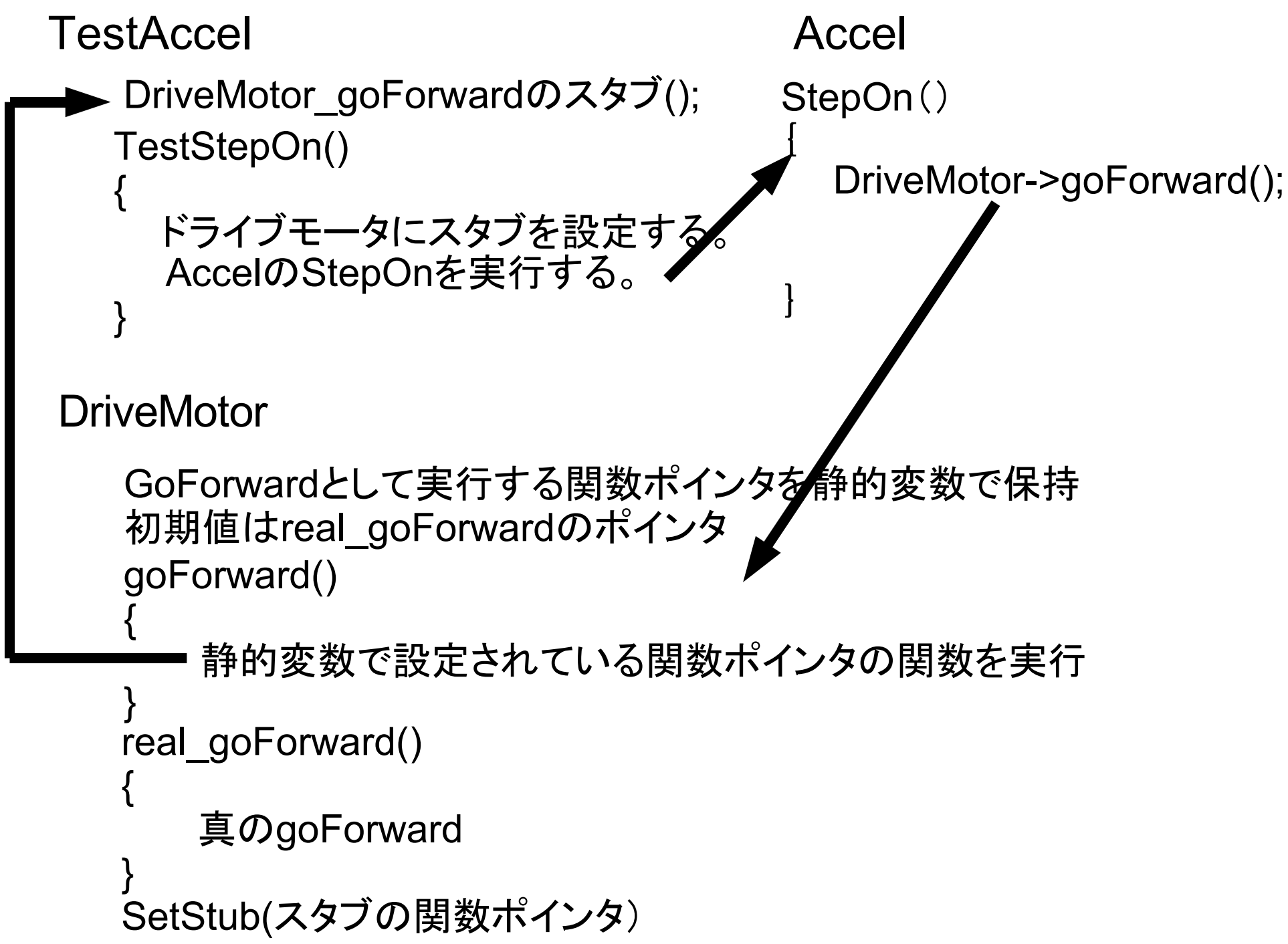
上図のように関数1と関数3から関数2が呼ばれる場合は関数2のダミーが競合する。
また、真の関数2とダミーの間でも競合が発生する。

スタブ関数をテスト関数から設定する方式を採用しスタブを用いたテストを実現した。

スタブを用いたテスト方法



DriveMotorと関連するAccelをテストしたい！


TestAccel
DriveMotor_goForwardのスタブ();
TestStepOn()
{
ドライブモータにスタブを設定する。
AccelのStepOnを実行する。
}
Accel
StepOn()
{
DriveMotor->goForward();
}
DriveMotor
GoForwardとして実行する関数ポインタを静的変数で保持
初期値はreal_goForwardのポインタ
goForward()
{
静的変数で設定されている関数ポインタの関数を実行
}
real_goForward()
{
真のgoForward
}
SetStub(スタブの関数ポインタ)

考察：スタブの実現方法比較

今回使用した関数ポインタでスタブを設定する方式以外にも
コンパイル環境を分ける方式によりスタブを使用したテストを実現できる。
２つの実現方式を比較すると・・

方式	仕組みのシンプルさ	変更への対処
コンパイル環境を分ける方式	◎	△
スタブの関数ポインタを外から与える方法	△	◎

クラスの増減を伴うようなリファクタリングを継続的に行う場合は、
変更への対処に強い今回の方式が良い。
逆に設計段階でクラス構成を固めて大きな変更が発生しない
場合は、コンパイル環境を分ける方式がシンプルで良い。

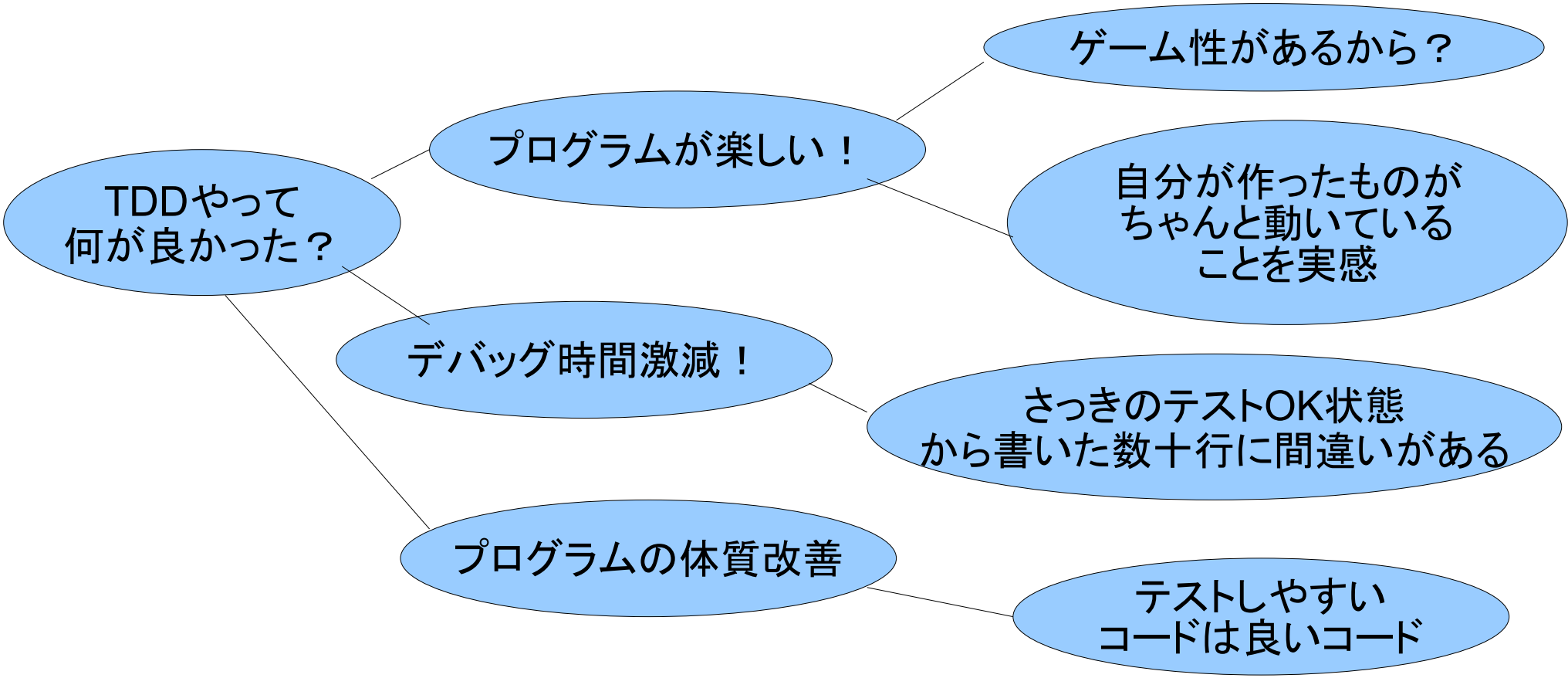
メトリクスで示される効果

あるコードメトリクス測定ツールにて静的解析を実施した結果
＊体験版の制限で10ファイルしか測定不可であるためライン数上位10ソースファイルを解析

Total Functions	72
Total LOC	663
Avg LOC	9.21
Avg Cyclomatic Comp. ..	1.81
Max LOC	74
Max Cyclomatic Comp. ..	13

自然にテストしやすいコードとなるため、コードのメトリクスが改善される。

効果の実感



TDDはPlan(テストコード)、Do(ターゲットコード)、
Check(テスト実行)、Act(修正、リファクタリング)
というPDCAサイクルを取り入れた効率の良い開発手法である

ふりかえり

1.TDDのテストコードは「プログラマのテスト観点」であることを説明しました。

2.ETロボコンでの開発事例の概要を説明しました。

3.C言語におけるスタブを用いたテストの実現方法について説明しました。

4.TDDが品質向上に対して効果があることを説明しました。