ソフトウェアテストシンポジウム 2006
JaSST'06: Japan Symposium on Software Testing 2006

TDD を加速する擬似クラス生成方法
― Simulated-Class by Tests with KikainekoMocker ―

伊尾木 将之† 河野 広伸‡ 合田 英二‡

†‡日本アイ・ビー・エム株式会社 〒103-8510 中央区日本橋箱崎町 19-21
E-mail: †e33052@jp.ibm.com, ‡{hkono, egohda}@jp.ibm.com

あらまし 本論文では，TDD を加速する擬似クラス生成方法を提案する．近年ソフトウェア開発に求められる品質が非常に高まっている中で，品質向上の技術としてテスト駆動開発と擬似クラスを使用する手法が注目を集めている．これらの手法は非常に有用ではあるが，擬似クラスの作成が困難であるなど技術的な問題点も存在する．そこで，これらの手法の課題を緩和する手法として「テストから擬似クラスを生成する方法」を提唱する．筆者らはこの手法に「SCT(Simulated-Class by Tests)」と名づけた．SCT がどのように TDD と擬似クラスの持つ技術的困難性を解決し，どのような発展の可能性を持つのかを示す．

キーワード テスト駆動開発，モック，スタブ

A Method of Simulated-Class Generation Accelerating TDD
― Simulated-Class by Tests with KikainekoMocker ―

Masayuki Ioki† Hironobu Kohno‡ and Eiji Gohda‡

†‡IBM Japan, Ltd. 19-21,Hakozakicho,Nihonbashi,Chuo-ku,Tokyo 103-8510 Japan
E-mail: †e33052@jp.ibm.com, ‡{hkono, egohda}@jp.ibm.com

Abstract In this paper, we propose a method of simulated-class generation to accelerate TDD. In recent software development, the need for quality has increased, and test-driven development and the usage of simulated-class are receiving attention for quality improvement. Although these methods are effective, there are technical problems ; e.g. the difficulty to create the simulated-classes. So, we propose a method of simulated-class generation from test codes. (We named this method "SCT(Simulated-Class by Tests)".) We provide an explanation of how SCT solves the technical problems which TDD and simulated-class have, and SCT's evolvability.

Keyword Test-Driven Development, Mock, Stub

1. はじめに

近年ソフトウェアの開発はますます複雑多様化し，これに伴い開発自体が非常に高度な技術を必要とする作業となった．

また，ソフトウェアに求められる品質も非常に高まっているが，高い品質のソフトウェアを開発することは容易なことではない．そのため品質向上に向けた様々な手法が提案されている．その中の一つとしてテストを開発の中心に置いたテスト駆動型開発(Test-Driven Development：以下 TDD と略す)[1,2]と擬似クラスの活用法があり，注目を集めている．

TDD は画期的な手法として品質向上に貢献できると期待される一方，後述するように TDD が技術的困難さを持たないわけではない．そこで TDD の持つこの課題を緩和する手法として，スタブやモックなどの擬似クラスの活用方法が注目されている[3]．しかしこの提案もまた技術的な困難さを保持している．

そこで筆者らは，これらの課題を克服する手法として「テストコードから擬似クラスを生成する手法」(SCT（Simulated-Class by Tests）)を提案する．そして，この SCT を実装したツールとして，筆者らが Java で開発を行った擬似クラス生成ツール「KikainekoMocker」[4]を紹介する．

TDD の具体的な解説は書籍などに譲るが，本論文ではまず TDD と TDD の持つ技術的な困難さを解説する．次に擬似クラスを活用する手法を紹介し，またその技術的な困難さも示す．そしてそれらの課題を克服する手法として，筆者らが提案する SCT の具体的な説明を行う．その際に先述の「KikainekoMocker」を用いて，どのようにしてテストから擬似クラスを生成するかを示す．最後に SCT がどのように問題を解決するかを示し，この手法の発展の可能性を示す．

2. TDD

2.1. TDD とは

TDD とは XP 提唱者として有名な Kent Beck 氏によって提唱された設計・実装のための手法である．TDD はテストを開発の中心に持ってくることによってソフトウェアの品質向上を図ろうとする．TDD では技術的な核として「テスト・ファースト」と「リファクタリング」を持っている．そのため TDD では，実装に先立ちテスト作成から始められる．

2.2. TDD の持つ技術的困難さ

TDD では実装に先立ちテストから作成され，常にテストを実行することによって品質を確保しようする．しかしテストの作成・実行が困難な場合には，TDD は上手く機能できない．このようなテストの作成・実行を困難にする原因として主に，

1. 未実装のクラスへの依存
2. 環境への依存

があげられる．以下この 2 点に関して解説する．

2.2.1. 未実装のクラスへの依存

未実装のクラスへの依存とは，開発対象クラスが未実装の(つまりはまだ存在しない)クラスへの依存関係を持つことを指している．この場合開発対象クラスのテストを実施することは不可能であり，TDD を活用することが出来ない．特にチーム開発において，他チームが開発を行う予定のクラスに依存している場合，開発はさらに困難をきたす．

2.2.2. 環境への依存

環境への依存とは開発対象クラスが DB やコンテナなどの環境に依存することを指している．環境に依存したクラスに対するテストは実行にコストがかかってしまいやすく，また環境独自の制約のために十分なテストが行えない場合が多い．結果として TDD を上手く活用できない．

3. 擬似クラス

3.1. 擬似クラスの手法

擬似クラスとは，正しい実装ではないが要求される振る舞いを見かけ上持つクラスのことを指し，スタブやモックなどがその例にあたる．TDD の技術的困難さを緩和する手法の一つして現在注目されている．

TDD の課題はいずれも開発対象の依存関係が原因であった．擬似クラスを使用するとこの依存関係を絶つことができ，TDD の困難さを緩和させることができる．具体的には未実装クラスや DAO(Data Access Objects)のような環境に関するクラスを擬似クラスで代用する．これによってテストを実行可能にし，また環境構築などのコストを回避することが可能になる．

3.2. 擬似クラスの問題点

擬似クラスの使用は TDD の困難さを緩和するが，しかし擬似クラスにはまた以下のような問題が存在する．

1. 擬似クラス作成の困難さ
2. 擬似クラスの定義の難解さ
3. 擬似クラス定義コードの廃棄

これらに関してそれぞれ解説を行う．

3.2.1. 擬似クラス作成の困難さ

擬似クラスは見かけ上の処理しか行わないが，しかしその作成に大きなコストが発生しやすい．これには擬似クラスに求められる入出力が一般に複雑な場合が多いことや，擬似とは言え一つのクラスを実装すること自体がコストがかかってしまう作業であるからだ．

この擬似クラスの作成のコストを抑えるために，いくつかの擬似クラス生成ツールが存在するが，しかし上で挙げた他の 2 つの問題点はまだ残る．

3.2.2. 擬似クラスの定義の難解さ

擬似クラスを作成するには，擬似クラスに要求されるインターフェースやメソッドの呼び出し順などの入出力に関する定義を行わなくてはいけない．しかしこの入出力が複雑な場合，定義を記述すること自体にコストがかかってしまいやすい．

また擬似クラスを生成するツールを用いた場合の多くは，そのツールの API に依存した定義を記述しなくてはならず，学習のコストが発生してしまう．さらに生成ツールの API を理解していたとしても，擬似クラスに求められる定義が複雑な場合，擬似クラス定義コード自体が非常に難解になりやすく，直感的な理解が困難になってしまう．

3.2.3. 擬似クラス定義コードの廃棄

擬似クラスに対する定義コードは，擬似クラスを生成するという目的以外に使用されないため，擬似クラスが本物のクラスに置き換わった時点で，破棄されてしまう．つまりは擬似クラスのために余分なコストを払っていることになる．これは低コスト・短納期が厳

しく求められる現状では見逃すことができない．

4. テストから擬似クラス生成という提案

TDD と擬似クラスは良い手法ではあるが，それぞれに技術的な困難さがある．そこで筆者らは SCT(Simulated-Class by Tests)と名づけた「テストから擬似クラスを生成する手法」を提案する．

4.1. SCT とは

SCT とは，あるクラスに対するテストをそのクラスの入出力に対する定義であると解釈し，その定義から擬似クラスを自動生成する手法のことである．従来の擬似クラス生成と比較したものを図 1 に示す．

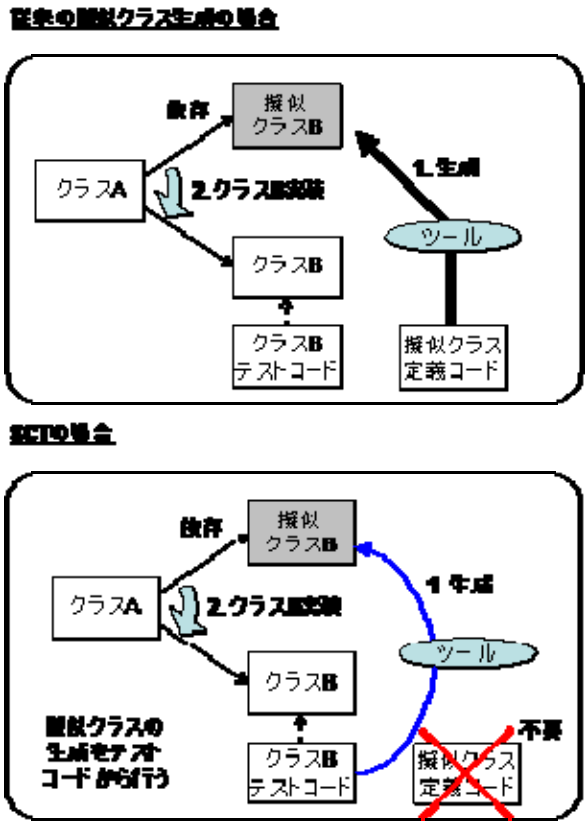

図 1　従来と SCT の比較

テストコードには，テスト対象のクラスが持つべき外面的な振る舞いが情報として含まれている．SCT ではこの情報を利用して，テストから擬似クラスを生成する．つまり，テストさえ存在すれば擬似とは言え動作するクラスを取得できるという提案である．

筆者らはこの実例を示すために，KikainekoMocker という擬似クラス生成ツールを実装した．後に具体例でも示すが KikainekoMocker は，JUnit のテストコードを読み込み，そこから擬似クラスに対する定義を解釈することによって，そのテストをパスする擬似クラスを生成する．また KikainekoMocker はオープンソースとして公開している．

4.2. SCT がどのように機能するか

以下に SCT がどのように機能し，先に挙げた問題を解決するかを示す．まず擬似クラスの困難さに関してどのように機能するかを示し，次に TDD に対してどのように機能するかを示す．

4.2.1. 擬似クラスに対して

擬似クラスの作成の困難さに関して，SCT では TDD で用いられるテストコードをそのまま擬似クラスに関する定義であると解釈し，そこから自動生成することによって作成に関するコストを抑えようとする．

また TDD で一般に用いられるテスティングフレームワークである xUnit 系のテストの記述は比較的容易であり，擬似クラスへの定義が直感的になりやすい．このため擬似クラスに対する定義が比較的容易になる．また一般的な擬似クラス生成ツールの知名度よりも JUnit などのほうが知名度は圧倒的に高いために，JUnit のテストコードに対する学習コストはかなり低いと言える．

さらに擬似クラスが本物のクラスに置き換わったとしても，擬似クラス定義コード自体がテストであるため，そのテストコードは本物のクラスに対して有効であり，このテストコードを有効に活用できる．

4.2.2. TDD に対して

先に述べたように TDD で上手く扱えなかった未実装クラスと環境依存クラスの問題に対して擬似クラスは有効に機能する．そして上で示したように SCT は擬似クラスをより活用しやすくすることが可能である．その結果として SCT は TDD を促進させることができる．

そして TDD であれば，まずテストを書くことから始め，SCT ではそのテストを活用する．このため TDD の自然なリズムの中で SCT を活用することができるのである．

4.3. 具体例

ここでは筆者らが開発した KikainekoMocker を用いて，擬似クラスの具体的な生成過程を示す．KikainekoMocker 自体は CUI での実行も可能であるが，Eclipse3 のプラグインとしても動作するため，ここでは Eclipse3 上での例を示す．

まずは手順を示す．

1. 擬似クラスに対するテストを書く
2. KikainekoMocker を使用して擬似コードを生成す

る

3. テストを実行して擬似クラスの振舞いを確認する

4.3.1. 擬似クラスに対するテストを書く

まずは擬似クラスに求められる要件をテストとして表現する．ここでは例として四則演算を行うクラス Calc.java の擬似コードを生成するための CalcTest.java という JUnit の TestCase を実装する．

図 2　CalcTest.java の配置

この時点では対象クラスである Calc.java はまだ存在していない．

CalcTest.java を図 3 のように実装する．

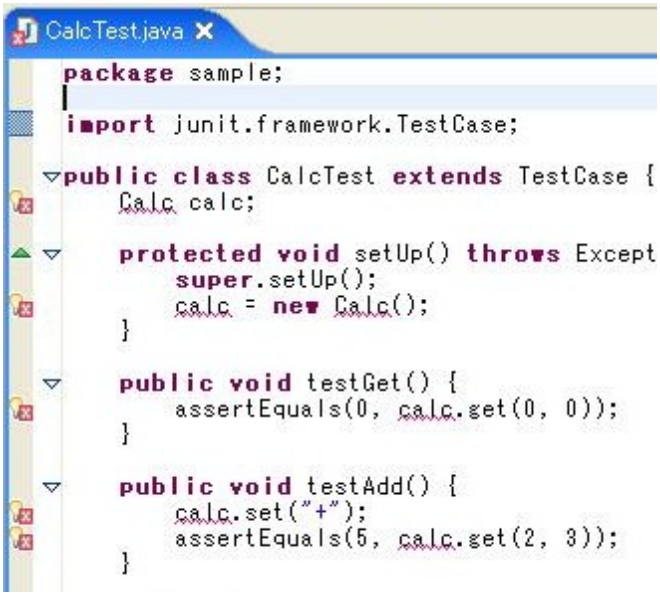

図 3　CalcTest.java

Calc.java がまだ存在しないために Eclipse でのコンパイルエラーの表示となっている．またテストコードも通常のテストコードと変わりがない．

4.3.2. KikainekoMocker を使用して擬似クラスを生成する

次に擬似コードを生成する．KikainekoMocker では TDD で重要視される開発のリズムを壊さないように非常に簡単に擬似コードを生成できるようにしている．図 4 のように右クリックのみで擬似クラスを生成する．

図 4　擬似クラスの生成

そしてこれを実行すると図 5 のように Calc.java が生成され，コンパイルエラーは消えている．但しこの Calc.java は擬似コードであり本当に要求される処理は実装されていない．

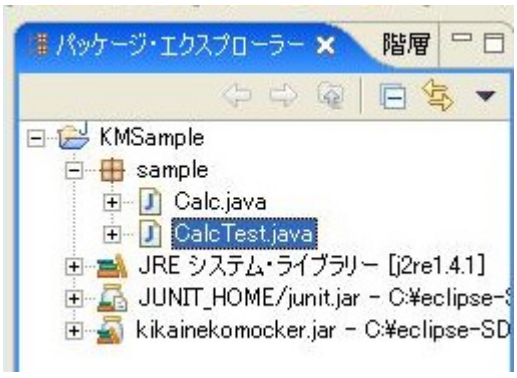

図 5　生成された擬似クラス(Calc.java)

4.3.3. テストを実行して擬似クラスの振舞いを確認する

生成された擬似クラスが求める振舞いを持っているかを確認するためさきほどの CalcTest.java を実行する．

図 6　JUnit 実行

JUnit がグリーンバーを示し, 擬似クラスがテストをパスしたことが分かる.

この例で分かるように KikainekoMocker はテスト以外の入力を必要とせずに, 擬似クラスを生成している. また KikainekoMocker を意識することはほとんどない. このため TDD の流れ, つまりテストを書き, それをパスさせるという流れの中で自然に使用することが可能なのである.

5. SCT の可能性

SCT の持つ可能性を示す.

5.1. 試行錯誤の容易化

先の KikainekoMocker の例で分かるように, 必要な入力はテストだけである. そのため擬似クラスの振る舞いを変更したければ, テストコードを変更するだけでよい.

これは設計に良い影響を与える. 設計時に簡単にでもテストを書けば, 擬似とはいえ動くものが手に入り, またその振る舞いを簡単に変更できるため, 設計に対する試行錯誤に効果的である.

また, もし受け入れテストなどをテストコードで表現できるならば, 要求の探索としても効果的である.

5.2. コミュニケーションの容易化

チーム間でのコミュニケーションの失敗の多くは, 互いの認識のずれによって引き起こされている. この認識のずれによって仕様がチーム間で微妙にずれてしまい大きなバグとなってしまうことは珍しくない. この問題を SCT は緩和させる可能性がある. なぜならテストというドキュメントを積極的に使用できるからである.

例えば, チーム間での仕様に関するやりとりは従来自然言語で記述されたドキュメントと口頭での伝達に頼っていた. しかし SCT では, これをテストコードで置き換えやすくする. テストコードはプログラミング言語で記述されるために認識のずれは起こりにくく, また KikainekoMocker などを用いるとすぐに動くものが手に入るため確認が取りやすい. このように SCT ではテストでのコミュニケーションを促進する.

5.3. 設計の確認

シーケンス図などの設計書を正しく実装するのは複雑な作業だが, シーケンス図などの設計書からテストを起こすことはさほど難しい作業ではない. そして SCT ではテストから擬似クラスを自動生成するため, 設計と動作するシステムとの距離を縮めることが可能である. これによって, ドキュメント上のみでの設計の確認から具体的なフィードバックを伴った設計の確認が可能になる.

また他チームとの開発で SCT を用いた場合, 自チームの開発が終了する時点では, 擬似クラスに対して十分なテストが書かれていることになる. そのため他チームが開発を行った真のクラスの受け入れテストになり得る効果もある.

6. SCT の限界に関して

SCT はテストから動作する擬似クラスを取得できるが, 逆に言えば生成できるクラスの仕様はテストの持つ仕様の表現能力に限定されてしまう. 例えば KikainekoMocker は JUnit のテストケースの表現能力に依存している. JUnit は入出力などの振る舞いに関する仕様の表現能力は高いが, 非機能要件などの仕様の表現能力は低い. そのため KikainekoMocker で生成できる擬似クラスは範囲が限定されてしまう. これは SCT の適用範囲に限界があることを意味している.

また仮にテストで全ての要件が記述可能であったとしても, そこから全てのテストをパスする擬似クラスを完全に生成可能であるかどうかも明らかになっていない. 入出力などの振舞いに関する多くのテストに対して, SCT の手法が可能であることは KikainekoMocker が証明している. しかし, 例えば環境に依存したテストなどには, SCT が対応することは難しいと予想される.

しかしテストコードで表現が困難な非機能要件や環境に強く依存した要件などが, 擬似クラスに対して真に求められる要件であるかどうかは議論の余地がある.

7. おわりに

本論文では TDD と擬似クラスの橋渡しを行う手法として SCT を提案した. SCT は, テストさえ存在すれば動作するクラスを取得できるということを主張している. これはテストの新たな活用方法であり, また設計・実装の新たな手法となり得る発展の可能性を秘めている.

しかしこの手法による実績は現在のところまだほ

とんどない．この手法の経験を積むことによってこの手法の限界やあるいは発展の可能性を探りたい．

文　献

[1] ケント・ベック，テスト駆動開発入門，ピュアソン・エデュケーション，東京，2003.

[2] 川端光義，倉貫義人，兒玉督司，バグがないプログラムのつくり方 Java と Eclipse で学ぶ TDD テスト駆動開 Be agile!，翔泳社，東京，2004.

[3] ロバート・C・マーチン，アジャイルソフトウェア開発の奥義，ソフトバンクパブリッシング,東京,2004.

[4] KikainekoMocker ， http://kikainekomocker.hp.infoseek.co.jp/index.html , 2005.09.30