テスト駆動開発におけるソースコードの循環的複雑度の計測

細谷 泰夫† 森崎 修司‡ 松本 健一‡
†三菱電機株式会社 ‡奈良先端科学技術大学院大学

1 はじめに

アジャイルプロセスは，ユーザのニーズに適合した高品質なソフトウェアを効率的に開発するための開発プロセスとして知られているが，品質保証の方法が明確に定義されていないことが普及の妨げの一因となっている．従来のウォータフォールプロセスと同様の品質保証をアジャイルプロセスに適用しようとしても，開発プロセスの差異が大きくうまく適用することができない．アジャイルプロセスにおいて品質保証を行うには，アジャイルプロセスがどのように品質を確保しようとしているかを分析し，品質確保のための手順が適切に実行されていることを計測，評価する必要がある．

本論文では，アジャイルプロセスの一つである eXtreme Programming における品質確保に寄与するプラクティス「テスト駆動開発（TDD:Test Driven Development)」に着目し，TDD が適切に実行されていることを評価するためにどのような計測を行えば良いかについて考察する．

2 TDD 評価の観点

TDD では，プログラム自体の実装の前に，期待される実行結果を定義し，それをテスト用プログラムとして作成する [1]．テスト用実装が期待どおりとなっているかどうかを確認することができる [1]．TDD が適切に実施されていることを評価するには，表 1 の観点で計測を行う必要がある．本論文では，「TDD で得られるべき効果が発揮されていること」を計測する方法について論じる．

3 TDD の効果

文献 [2] では，TDD の効果として，「プログラミングが楽しくなる」，「プログラムを変更する勇気が出る」，「テストしやすいコードを書くようになる」が挙げられている．「プログラミングが楽しくなる」については，定量的な計測が難しいと考える．「プログラムを変更する勇気が出る」については，この勇気によりコードを改善するリファクタリングを躊躇なく実行することができる．コードがシンプルさを維持していることを計測することでその効果を評価することができると考える．「テストしやすいコードを書くようになる」については，テストを先に書くことにより，単機能の規模がテストケースを作成可能な範囲に抑制されるという効果が期待される．つまり，単機能の規模を計測することによりその効果を評価することができると考える．コードのシンプルさの維持と単機能の規模の抑制は，共に循環的複雑度の抑制に強く関連している．「プログラムを変更する勇気が出る」，「テストしやすいコードを書くようになる」の 2 つの効果を，循環的複雑度を計測することにより評価することが可能であると考えた．

表 1: 計測の観点

網羅性	テストコードが TDD 適用対象の仕様に対して漏れなく作成されていること
妥当性	テストコードの内容が妥当であること
効果	TDD で得られるべき効果が発揮されていること

4 循環的複雑度の計測

本実験では，共通の H/W で動作するライントレーサ制御ソフトを開発し，ソフトウェアのモデルと走行タイムを競う Embedded Technology Software Design Robot Contest のために開発されたソフトウェアを計測対象として TDD を実施した場合と TDD を実施しなかった場合について循環的複雑度の計測を実施した．計測の概要を表 2 示す．

表 2: 計測の概要

計測対象の指標	コード行数，循環的経路複雑度
計測の最少単位	関数
計測回数	開発の経過に合わせて 3 回

Yasuo Hosotani†, Shuji Morisaki, Ken-ichi Matsumoto‡
†Mitsubishi Electric Corporation, ‡Graduate School of Information Science, Nara Institute of Science and Technology

5 計測結果と考察

本実験によって計測した関数当たりのコード行数と循環的経路複雑度の分布を図 **1**，図 **2** に示す．図 **1**，図 **2** の点は一つの関数に対応し，横軸は関数の規模(コード行数)，縦軸は循環的経路複雑度を示す．図 **1** は TDD を適用したもの，図 **2** は TDD を適用していないものである．図 **1** では，関数当たりのコード行数は 30 以下，循環的経路複雑度は 10 以下に全ての関数が収まっている．一方，図 **2** では関数当たりのコード行数は 100 以下，循環的経路複雑度は 13 以下の範囲でバラツキが見られる．

時間推移による循環的経路複雑度の傾向を，図 **3**，図 **4** に示す．図 **3**，図 **4** は，開発のある 3 つの時点における循環的経路複雑度の傾向を示している．循環的経路複雑度の最高値，最低値，中間値，関数の 75 %が属している範囲を箱ひげ図を用いて示した．図 **3** は TDD を適用したもの，図 **4** を適用していないものである．図 **3** では，循環的経路複雑度の推移は，最高値，75 %の範囲共に変化が少なく，最高値は 10 以下に収まっている．一方，図 **4** では，時間推移による変動が見られた．また最高値については，開発途中では 16 を超える関数も見られた．

これらの計測結果から，TDD の実施有無と循環的経路複雑度の関係を考察する．図 **1** により，TDD を実施することにより循環的経路複雑度および関数当たりのコード行数が一定の範囲に抑制されると読み取れる．また，図 **1** では，関数当たりのコード行数に応じて循環的経路複雑度が増加する傾向が見られるが，図 **2** では関数当たりのコード行数が少ないにも関わらず，循環的経路複雑度が突出している関数が存在する．

図 **3** からは，TDD を実施することにより，循環的経路複雑度が時間推移に関わらず低い値で安定していることが読み取れる．一方，図 **4** では，最終的に循環的経路複雑度は低減されているが，開発途中で循環的経路複雑度が増大している．すなわち，TDD を実施することにより，関数当たりの循環的経路複雑度，コード行数は低い水準でバラツキが小さくなると共に，時間推移によらず安定する効果が得られる可能性がある．関数当たりの循環的経路複雑度，コード行数が時間推移によらず抑制されることにより，時間推移によらずプログラムの保守性を高く維持することができると考える．保守性を高く維持することにより，新しい機能の追加にかかるコストを低減することが可能と考える．

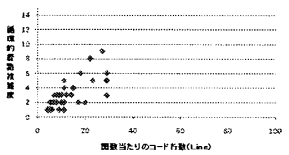

図 1: 循環的経路複雑度の分布 (TDD あり)

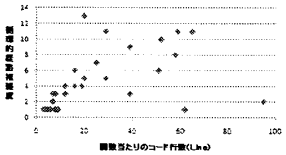

図 2: 循環的経路複雑度の分布 (TDD なし)

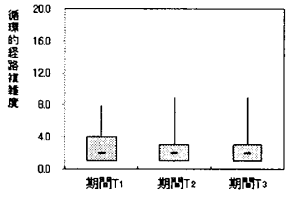

図 3: 時間推移による循環的経路複雑度の傾向 (TDD あり)

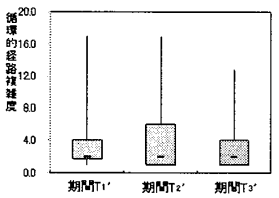

図 4: 時間推移による循環的経路複雑度の傾向 (TDD なし)

6 結論

本評価により，TDD の実施により循環的経路複雑度が一定の範囲内に収まる結果を得た．TDD の効果を評価するためには，関数当たりの循環的経路複雑度の平均を計測，評価するのではなく，関数当たりのコード行数，循環的経路複雑度の分布が一定範囲に安定しているかどうかを計測することが有効であると考える．また，本評価では，TDD の効果が発揮されているかどうかを評価するために関数当たりの規模，循環的経路複雑度を計測したが，残る評価の観点である網羅性，妥当性については異なる計測方法が必要であると考える．今後の課題として，これらの計測方法の検討および TDD 以外のプラクティスについてもプラクティスが適切に実行されていることを計測する手段の検討が挙げられる．

参考文献

[1] Beck,K.:*Embracing change with extreme programming.*Computer.IEEE computer society.Oct.1999

[2] 大月美佳:Test-Driven Development(テスト駆動開発) 開発手法としてのテスト. 情報処理 vol.49. 情報処理学会.Feb. 2008