ソフトウェア工学 118−20
（1998. 3. 11）

ソフトウェア開発クリーンルーム手法の方法論

西 橋 幹 俊

日本アイ・ビー・エム（株）

本稿では、ソフトウェア開発クリーンルーム手法に関連する後続の発表に先立ち、当手法の包括的解説を行う。特に、より技術的な議論や対比への導入となるように、クリーンルーム手法の概要に加えて、その方法論が実感できるように、一開発過程例を通じて各要素技術の適用場面を概観するほか、これまでに見えてきている問題点、課題、方向性等を整理して提示する。これらは筆者の私見に基づく側面もあるが、出来るだけ見解の多様性を列挙しつつ、後続の発表への複数の視点の提供や問題提起となることを目指す。

Methodologies in Cleanroom Software Engineering

Mikitoshi Nishihashi

IBM Japan Co. Ltd.

This article is to show a comprehensive introduction for Cleanroom Software Engineering (CRSE), before the following presentations in the seminar related to CRSE. Especially it includes an overview of actual development situations which apply each component technology in CRSE and list up issues, themes, and directions etc. which have been observed and found so far in addition to the overview of CRSE itself. They are intended to provide audience with multiple views to the following presentations in the seminar with various opinions, although they may include my personal ones to some extent.

1．クリーンルーム手法とは何か

表題に関してはいくつかの場で解説しているが、ひとつの言い方は、次の四つの要素技術からなるソフトウェア開発のための複合手法だということである。

1．プロジェクト管理技術としてのインクリメンタル開発
2．ボックス構造分析を用いた段階的詳細化
3．関数的等価性に基づく検証
4．ユーザーシナリオに基づく統計的品質保証

これらの四つの要素技術からなる複合手法、クリーンルーム手法の根底には幾つかの思想が横たわっている。そのうち、大きなものとして次の三つを挙げておこう。

A．開発者が管理可能な形で、開発対象を順次小さくしていく。

B．開発対象を関数とみなし、段階的詳細化も検証も「関数として」行う。
C．開発対象の「正しさ」を、人力（知力）と計算機（実稼動）とを場合に応じて使い分けて検証する。

この両群の関係を示すと、次のようになる。
1：A
2：A＋B
3：B＋C
4：C

以上をまず心にとどめておいて、クリーンルーム手法を概観してみよう。クリーンルーム手法を用いたソフトウェア開発の概要の流れは次の通りである。

まず措定されたユーザー要件から仕様を措定し、それを幾つかのインクリメントに分割する。それぞれのインクリメントについて、開発チームはボックス構造分析を用いて段階的詳細化を行いコードを開発していくと同時に関数的等価性に基づく検証により開発されたコードが正しいことを検証する。一方、品質保証チームはこのインクリメントに対応するユーザーの使用状況をユーザーシナリオとしてまとめ、それに基づいてテストケースを生成しておく。開発チームからコードを受け取ってそれを実稼動させ、どれくらい稼動し続けるかを測定する。問題が生じた場合はその内容を整理して開発チームへわたす。開発チームはそれを解決して品質保証チームへわたす。通常、このやりとりは、高々数回で終わり、品質保証チームは、このインクリメント［まで］のコードが目標通りの時間稼動し続けることを確認した後、開発チームへ次のインクリメントのコードを受け取る用意のあることを伝える（そうでなければこのインクリメントのやり直しであり、コードは開発チームへ突き返される）。すべてのインクリメントでこの作業が完了すれば、両チームの前には、事実上無欠陥のコードが出来上がっていることとなる。

なお、各インクリメントの完了時に、それまでに蓄積してきた認識の深まりや、時間の経過に伴う社会やユーザーの変化を反映して、仕様の変更や詳細化がなされうる。

このあと、２－４で、各要素技術や記述法等を概観したあと、５で、この流れを簡単な開発過程例を通じて具体的に見ていくことにしよう。

２．マクロな要素技術としてのインクリメンタル開発と統計的品質保証

インクリメンタル開発は、開発しようとするソフトウェアを、まず骨格を作り、重要なところから肉付けし、更に詳細部分をつめていこうとする、開発のやりかたであり、その一単位をインクリメントと呼ぶ。各インクリメントの内部は滝型開発のミニチュア版である。当該インクリメントの仕様を、開発チームが後述するミクロな要素技術であるボックス構造分析と検証とを用いて詳細化・検証し、品質保証チームがユーザーシナリオに基づく統計的テストを実稼動によって行い、望むＭＴＴＦ

(Mean Time To Failure) が得られることを確認する。

インクリメンタル開発の意義は、総じて言えば、「仕様のサブセットを知的管理可能な形で順次実装出来る」ことであると言えるが、より細かくは、つぎのようなメリットが享受できることであると言えよう。

A．開発対象ソフトウェアとインターフェースを持つシステムの他のコンポーネントとの整合性を開発のかなり早い時期から実稼動させつつ確認することが出来る（整合性の早期稼動確認）。

B．新しい領域での開発であるとか、ユーザーの要件が未だ確定しない状況の下で、最も確からしい仕様を措定して開発し必要に応じて変更を加えることが、最小限の再作業で可能である（自然なプロトタイピングの場を提供）。

C．開発対象そのものではなくテストのための環境作りに要する作業を最小化することが出来る（ テスト環境作成作業の最小化）。

D．使用頻度の高い機能や、誤動作時に影響の大きい機能を、頻度多くテストすることが出来る（ 重要機能の高頻度テスト）。

E．ユーザーによっては詳細機能まで含んだシステムを求めるよりは、早期に基本機能を含むシステムを受取ることを望む場合があるが、各インクリメントの完了時はそこまでの機能の開発の完了時であるので、そこで配布することが可能である（稼動物配布時の融通性）。

インクリメンタル開発は以上のようなメリットを提供しうるのであるが、これを実現するのは必ずしも容易ではない。事実、上記のメリットは場合によっては相互に競合しうるものであり、それぞれの場合に応じて的確にトレードオフを判断する必要がある。いくつかのケースについては、ノウハウの蓄積がなされつつあるが、このようないかにインクリメントをきるかという、インクリメンタル開発計画の立てかたについては、なお研究の余地のあるところである。

ここではソフトウェアの「開発容易性」という概念を、ハードウェアの 製造容易性とのアナロジーで提起しておこう。ハードウェアの製造に際しては、その組立工程の立てかたによって製造が容易にも不可能にもなりうるが、開発段階でそのことを考慮に入れておけば状況の改善に役立つ。同様なことがソフトウェアの開発に際してもありえてよいと思われるが、インクリメンタル開発計画の立てかたの共通項として、ソフトウェアの開発容易性は出てこないだろうか。これは、ソフトウェアアーキテクチャーへの一要件を提示するように思われる。

インクリメンタル開発の各インクリメントの最後の工程は、ユーザーシナリオに基づく統計的品質保証であった。その意義は、量質両面において効率的にテストを行うことであると言えよう。ユーザーが、開発されたソフトウェアを含むシステム全体をどのように使うかを、簡単に言えば、その使い方

のメニューとそれぞれの使用頻度とを記述する。それに対応してテストケースを自動的に生成することができる。これを実行すれば、ユーザーの下で現実に起こるであろう障害をほぼ発生頻度順に生起せしめ、つぶしていくことができる。ひとつのサーベイによれば、このような荷重テストは単なるじゅうたん爆撃テストに比べて２０倍も効率のよいことが示されている。また、必ずしも使用頻度は高くはないが誤動作をしたときのインパクトが大きいという機能もある。そのような機能のテストの比重を高めることも可能である。

さて統計的品質保証のこのような意義は頭では理解できるのであるが、現実的場面での適用には壁があるほか、これを適用する以前に解決しておくべき課題もあるように思える。

現実的場面での統計的品質保証の適用における問題点としては、経験不足、データ獲得の難しさ、道具立ての精度と複雑さのトレードオフ、といった三点がまず挙げられる。
第一の経験不足とは、次のようにして起こる。クリーンルーム手法の適用は、まず幾つかの要素技術を選択的に使用する部分的適用から始まることが普通である。インクリメンタル開発と早めの文書化とで大きな成果を収めた例があるし、関数的等価性に基づく検証こそクリーンルーム手法の真髄として、ボックス構造分析と検証とをまず適用するケースも多い。いずれにしても、統計的品質保証の適用は後回しとなる。さらに、仮に統計的品質保証を適用したとしてもその結果の公表には、クリーンルーム手法の要素技術一般が抱える困難さが待ちかまえている。基本的にある種の手法や技術の成果の公表は、その使用の前後でのよさ加減の対比としてなされる。しかし品質データの公表には微妙さがつきまとう。こうして、一開発グループとしても、クリーンルーム手法適用社会全体としても「経験不足」に陥らざるを得なくなるのである。
第二のデータ獲得の難しさとは、ユーザーシナリオにおける各シナリオへの重み付けデータを獲得することの難しさである。新しい領域での難しさは想像に難くないであろうが、既存の領域であってもその獲得は決して「安価」ではない。
第三の問題点とは、次のようなことである。これまで統計的品質保証を行うためにユーザーシナリオを記述する方法のひとつとして文脈自由文法に確率を付随させて記述する方法が使われてきた。形式言語理論を学んだ人にとっては何でもない道具立てであり、全く知らないという人が習得するのもそんなに困難ではないように思える。しかし、これをも難しすぎるという声が一面である。一方、形式言語理論を学んだ人には明らかなように、この文法の表現能力には制約がある。テストしたいケースがこの文法の制約を超えた範囲にある場合にこの道具立てでは不十分であることが起こりうる。この両面を現実の場面でいかに調和させるかが、第三の問題点の内容である。

統計的品質保証の現実的適用における問題点は以上のとおりであるが、統計的品質保証の適用には次のようなより基本的な問題点があるように思える。つまり、実稼動システムの一コンポーネントとしてのソフトウェアの開発には、ボックス構造分析と検証とを通じて事実上無欠陥を達成することを目指すのであるが、他のコンポーネントはいったんは考慮外におき、統計的テストにおいて、「システム全体として効率よくテストする」ことを目指すわけである。他のコンポーネントにたちの悪いバグが入っている場合、この目標は達成し得ないことも十分考えられる。他のコンポーネントもそれぞれ十分な品質を達成していて初めて「統計的品質保証」はその力を十分に発揮出来るであろう。ソフト

ウェアの部分ではこのような結合が可能なことが期待される。それぞれのコンポーネントをクリーンルーム手法のミクロな技術を使うことにより事実上無欠陥に作り上げ、その結合したシステムは事実上無欠陥が期待されるのであるが、それを確認するために統計的品質保証を使うという構図である。そのような結合を提供することが、ソフトウェアアーキテクチャーに期待される一要件であるように思える。

以上、クリーンルーム手法のマクロな要素技術としてのインクリメンタル開発と統計的品質保証とを概説すると共に、それらの抱えている課題、ソフトウェアの開発容易性や結合可能性を、ソフトウェアアーキテクチャーへの要件として提示した。

3．ミクロな要素技術としてのボックス構造分析と検証

インクリメンタル開発計画が策定され、各インクリメント内での開発で使われる要素技術は、ボックス構造分析と検証とである。これらを確実に用いれば、事実上無欠陥のコードを生産性よく開発することができる。その根底には、ソフトウェアを関数として捉えること、詳細化とその検証との作業の中で、入力データの領域全体を視野に入れて判断していることが理論的なベースとして存在するが、通常の実践の中では明示的に「関数」ということを意識しなくても進む。

ボックス構造分析とは、開発しようとするソフトウェアの外的動作、それを実現するために必要な内部データ、それを用いた手続きにそれぞれ焦点を当てたブラックボックス、ステートボックス、クリアボックスとして、開発対象を順次詳細化していき、更にクリアボックス内に残る一段低次のブラックボックスに対して同様の詳細化を繰り返すことにより、最終的にはターゲット言語で書かれたコードに至るやり方である。各段階の詳細化の前後がきっちりと文書化される。

ボックス構造分析による詳細化が、［各インクリメントの］トップレベルの仕様からプログラミング言語の一文に至る詳細化・実装の過程であるのに対して、関数的等価性に基づく検証は、その各段階の詳細化が正しく行われていることを確認する作業である。

このようにボックス構造分析と検証とは、「ボックス構造分析を行った上で検証」という明白な順序性があり、「方向として逆向きの」作業であるが、現実の開発チームの作業としてはこれらをほぼひとかたまりの作業として行う。そうでなければ実際的にはならない。具体的には次のような具合である。各インクリメントで実装すべきことを開発チームのメンバーは分担し、開発の早期から早め早めに「開発レビュー」を行う。ここでの眼目は、担当者が分担部分を説明した後、質疑応答によりチームメンバー内の疑問を解消する一方、代替案の提示等をする中で設計の単純化等よりよい設計にしていくことである。その中で、トリビアルな誤りは解消される。説明中に担当者自身が気が付くという形をとることもあるし、他のメンバーからの疑問や異論の提示がトリガーになることもある。この作業をきっちりと行った結果、開発チームの各メンバーが、自分の担当部分以外の詳細化に対してもその正しさが「独立に」判断できるようになっていることが重要である。ある程度詳細化がまとまったところで「検証レビュー」を行う。一段一段の詳細化が関数的に等価であるか否かの「公式かつ最終

的な」作業である。が通常は開発レビュー実施のおかげでさっさと進む。
ときどきメンバーのひとりまたは複数が疑問や異論を唱えることがある。ここが重要である。ほぼ正しくなされてきた詳細化に誤りを見つけた「かもしれない」からである。綿密に疑問や異論のもとを探る。結局、各人の判断過程は一致するだろう。そして、疑問や異論が誤りであったと判明することもあるだろうし、それが重要な見落としや判断ミスの発見の糸口となるかもしれない。後者の割合が小さく、例えば１％程度であったとしても大収穫であることを認識しよう。開発の後の局面まで残せばどれだけコストがかかるかわからない禍根をこの段階の作業で除去できたのだから。
この過程を厳密に実施した場合の精度を一例で示すと、一段一段の判断で誤りを犯す割合を１％以下に抑え得るメンバーが５人でやったとすれば、チーム全体としてはテンナインの精度で正しく判断できることになる。

４．記述内容と記述法

ボックス構造分析を進めるに当たっては、仕様や設計の内容を記述する必要がある。この記述を、これから詳細化したい意図を含んでいるということと関数であることとを込めて「意図関数」と呼ぶ。また、検証を進めるに当たっては、プログラムの動きを全体として記述する必要があり、これを「プログラム関数」と呼ぶ。

記述の仕方は、チーム内でコミュニケーションが取れる範囲内で自由であるが、ここでは、これまでに使われ洗練されてきているひとつの記述体系を紹介しよう。
同時割当文や条件付き同時割当文を用いて、出力が入力からどのように変化するかを表現する。
テーブルを用いても同様な表現が出来る。また、対象領域の概念を語彙として増やしていくために記述関数という形式を用いる。以上は意図関数やプログラム関数を表現するのに必須と言ってもよい。
このほか、ある種の繰返し構造を表現するのに［文脈自由］文法を用いると重宝する場合がある。

５．開発過程例：自動販売機のファームウェア開発

いま、コインを投入し欲しい飲み物を選択して、それ［と必要に応じておつりと］を受け取る、通常の自動販売機を制御するファームウェアを開発する場面を想定し、クリーンルーム手法適用の模擬体験をしよう。（本例に関しては、時間の許す限り当日ご紹介したい）

［投入されたコインに応じて、適切に現在投入金額を表示し、必要に応じて飲物、おつりを出す］
　［投入されたコインに応じて、現在投入金額を表示する］
　［投入されたコインに応じて、必要なら飲物を出す］
　［投入されたコインに応じて、必要ならおつりを出す］