半導体製造装置の FA 間通信ソフトウェア開発
～オブジェクト指向リファクタリングにおけるテストプロセス～

M2007MM035 山内健
指導教員 野呂昌満

1　はじめに

我々は OJL(On the Learning) として，半導体製造装置の FA(Factory Automation) 間通信ソフトウェア開発を行った．半導体製造装置の FA 間通信ソフトウェアは，手続き指向言語で作成されており，作成され始めてから 10 年以上経過し，その間，場当たり的な機能追加・変更によって，修正することは容易ではなくなっている．
本 OJL では，ソフトウェアの再利用性，保守性を向上させる手法の一つであるオブジェクト指向 [1] リファクタリング [2] を選択し，構造の整理を行った．リファクタリングは，システムの振舞いが変化していないことを保証する必要する．既存コード, 設計書をレビューし，分離が必要なコードの修正を行う．修正後のコードに対して振舞いが変化していないことを保証する．本 OJL で行うリファクタリングはソフトウェアアーキテクチャを中心に既存コードを対応付けし，一般のリファクタリングとはアプローチが異なり，システムの振舞いを保証できない．
本 OJL の目的は，ソフトウェアアーキテクチャに基づく開発プロセスを提案することである．筆者の担当であるテストプロセスでは，システムの振舞いを保証するテストプロセスの提案である．本 OJL のリファクタリングにおけるテストでは，システムの機能に対して保証するが，システムの機能を確認するためのテストケースを全て網羅することは困難である．本 OJL では，テストプロセスを決定し，システムの機能を保証でき，テストケースを絞り込んだ機能テストを行う．また，繰り返し機能復元を行うリファクタリングであることから，復元された機能と関連する機能に限定した機能テストを行い，テストにかかる労力を軽減する．
OJL では，問題を解決するための基底技術である「ベース技術」の蓄積し，ベース技術を選択・合成し，即応できる技術である「メタ技術」を修得する．本 OJL のベース技術は，ソフトウェア開発プロセスとして，オブジェクト指向リファクタリング，ソフトウェアレビュー，ソフトウェアテスト，ソフトウェアアーキテクチャとして，オブジェクト指向アーキテクチャスタイル，デザインパターンである．メタ技術は，ソフトウェア開発プロセスにおけるテストプロセスである．

2　提案する開発プロセス

本 OJL で提案する，ソフトウェアアーキテクチャに基づくソフトウェア再開発プロセスを図 1 に示す．
一般に行われるリファクタリングは，既存コードを修正し，オブジェクト指向コードを実現する．単体テストにおいてコードの振舞いが変化していないことを確認する．本 OJL のリファクタリングは，既存コードをソフトウェアアーキテクチャに対応付けする．対応付けするさいに，

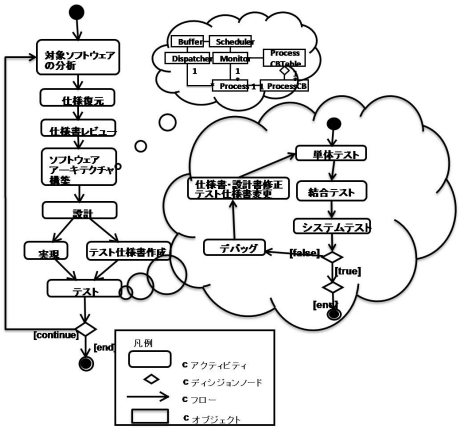

図 1　再開発プロセス

新たなコードの作成が必要であり，コードの振舞いを保証するテストではシステムの振舞いの保証が行えない．
本 OJL では，システムに求められる仕様である機能をテストすることで，システムの振舞いを保証する．機能保証を行うために，単体テストにおいて，内部構造におけるバグの検出，動作保証を行い，結合テストで，システムの機能を保証する．システムテストにおいて，構造を大規模に変更することから，システムの非機能要求を保証する．

3　背景技術

3.1　オブジェクト指向リファクタリング

オブジェクト指向リファクタリングは，データ抽象化と多相性の概念 [1] を背景としてソフトウェアの振舞いを変更せずに構造を変更し，オブジェクト指向コードに既存コードをマッピングする．オブジェクト指向リファクタリングを行うことで，機能追加・変更に対する柔軟な構造を実現する．
一般に行われるリファクタリングは，既存コードから修正したオブジェクト指向コードに対して，変更前のコードと変更後のコードに対して振舞いが変化していないことをテストする．本 OJL のリファクタリングは，既存コードをソフトウェアアーキテクチャに対応付けするさいに新たにコードを作成する．
二つのリファクタリングでは，既存コードとオブジェクト指向コードの対応付けが異なる．本 OJL におけるリファクタリングのテストは，システムに求められる仕様である機能をテストすることでシステムの保証をする．

3.2　ソフトウェアテスト

ソフトウェアテストは，各テストにおいてテスト手法・技法を選択し，ソフトウェアの動作・機能保証を行う．テスト技法は，ソフトウェアに対する保証すべき特徴に対し

て，バグの検出，動作保証を行う技術 [3] である．
制御フローテストは，プログラムから一連のパスに対してテストする構造テストの技法である．ユースケーステストは，システムの機能を表現するユースケース図を用いて，ユースケースのシナリオ通りの動作を行うことを確認する機能テストのテスト技法 [4] である．本 OJL では，既存コードをオブジェクト指向コードに対応付けを行うさい，新たにコードを作成する必要がある．単体テストにおいて，制御フローテストによって，オブジェクト指向コードに対するバグの検出・動作保証する．結合テストにおいて，ユースケーステストを用いて，シナリオ通りに動作することで機能保証しながら，テストケースを削減する．

3.2.1 本 OJL における機能テスト

機能テストは一般に，システムテストで行われる．しかし本 OJL では結合テストにおいて機能を保証する．対象ソフトウェアの特徴から，システムに対するイベントの入力によってサブシステムの関数の呼び出しを確認し，モジュール間の協調動作とシステムの機能を保証する．

3.2.2 インクリメンタルテスト手法

インクリメンタルテスト手法は，結合テスト手法の一つである．追加されるモジュールに対してテストを行い，追加されたモジュールを統合させて，機能を保証する [5]．本 OJL において，リファクタリング対象領域を拡大させ，機能復元する反復開発を行う．機能復元を行うさいにモジュールが追加・変更される．追加・変更されたモジュールを追加モジュールと捉え，テストを行い，モジュール統合時に機能復元された機能に対するテストを行う．

4 本 OJL における専用手法

本 OJL では，システムの振舞いを保証するために，システムの機能に対するテストを行う．半導体製造装置の FA 間通信ソフトウェアは，装置制御プロセスを管理するシステムである．システムに対してイベントを入力し，サブシステムの関数の呼び出しによって機能を確認する．機能をテストするための入力となるイベントの組合せは膨大であり，テストケースの削減が必要である．

4.1 結合テストにおけるユースケーステストの選択

ユースケーステストを用いることで，システムの機能を捉えたテストが可能になる．ユースケーステストは，テストケースの大幅な削減が可能であるが，バグが少ない状態でなくては適用できない．内部構造の動作保証，バグの検出を行うために，単体テストにおいてホワイトボックステストを選択することで，ユースケーステストを可能にする．

4.2 ユースケーステストに基づくインクリメンタルテスト

本 OJL では，リファクタリング対象領域を広げ，機能復元を繰り返す反復開発である．機能テストにおいて，テストケースを削減するために，ユースケーステストによる機能テストを選択している．ユースケーステストをベースとして，復元された機能に対するテストを実現するために，ユースケーステストに基づくインクリメンタルテストを行う．既に保証した機能のテストを削減するために，ソフトウェアアーキテクチャにおける追加・変更モジュールに対してバグの検出・動作保証する．追加・変更モジュールの動作保証を行うことから，モジュール統合時において追加された機能・関連する機能に限定したテストを実現する．

5 テストプロセス

本 OJL におけるテストは，システムの機能をテストすることで，システムの振舞いを保証する．テストプロセスの詳細を図 2 に示す．

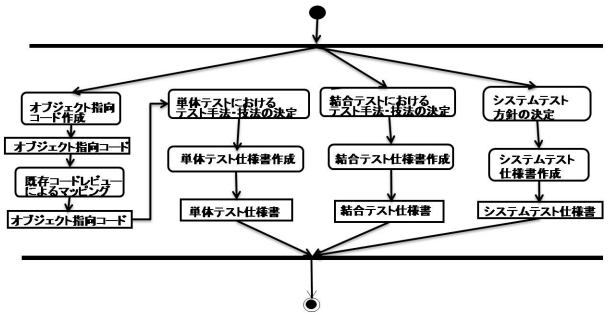

図 2 テストプロセス

装置制御プロセスを管理するシステムにおいて，機能を確認するために必要な入力イベントの組合せは膨大である．本 OJL では，内部構造に対してバグの検出・動作保証することで，システムの機能の動作を確認することで，機能テストを可能にし，テストケースを削減する．
テストプロセスとして，内部構造のバグの検出・動作保証を行うために，単体テストにおいて，ホワイトボックステストを行う．システムの機能を確認するために，結合テストにおいて，ユースケーステストを用いて，テストシナリオによる機能の動作を確認する．システムテストにおいて，システムの大規模な構造を変更したことから，非機能要求に対する保証を行う．
本 OJL では，機能復元を繰り返す反復開発であり，追加された機能を保証するために，ソフトウェアアーキテクチャの変更箇所を保証し，ユースケーステストに基づくインクリメンタルテストを行う．

5.1 単体テスト

単体テストは，一般のリファクタリングのテストでは，既存コードから分離するコードを修正し，修正されたコードに対して振舞いが変化していないことを保証する．
本 OJL のリファクタリングは，既存コードをソフトウェアアーキテクチャに対応付ける．対応付けを行うさいに，新たにコードを作成する必要がある．モジュールに対するバグを検出・動作保証する必要があり，コードに対するテストであるホワイトボックステストを行う．また，装置制御プロセスを管理するシステムは，イベントを扱うが，データであるイベントの書き換えはほとんと行われない．テスト技法に制御フローテストを選択することで，

コードの分岐条件を網羅し，コードカバレッジを全て満たすテストが可能であり，内部構造に対する高い動作保証，バグの検出が可能である．

5.2 結合テスト

結合テストは，一般にリファクタリングでは実施されない．リファクタリングでは，単体テストによって，既存コードと構造変更後のコードの振舞いが変わらないことを保証する．本 OJL では既存コードをソフトウェアアーキテクチャに対応付けしており，システムの機能に対する保証が必要である．

本 OJL では，システムの機能に対してユースケーステストを適用し，シナリオに従ったテストを行い，機能テストを行った．また，リファクタリング対象領域を広げながら，機能復元を繰り返す反復開発を行っており，復元機能に限定したテストを実現するために，ソフトウェアアーキテクチャの変更箇所を保証し，ユースケーステストに基づくインクリメンタルテストを行った．

5.2.1 ユースケーステストによる機能テスト

ユースケーステストにおいて，ユースケースのシナリオに基づいてイベントの入力に対して期待されるサブシステムの関数の呼び出しが行われることを確認する．

システムの機能で重要な機能であるユースケース「2.2.1 既存システムの関数を呼び出す」について説明する．ユースケース図を図 3 に示す．

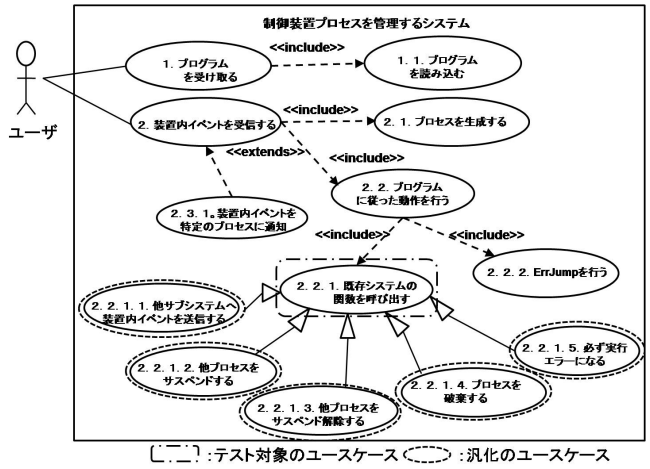

図 3 ユースケース図

「2.2.1 既存システムの関数を呼び出す」は，汎化のユースケースとして，

- 2.2.1.1 他サブシステムへ装置内イベントを送信
- 2.2.1.2 他シーケンスをサスペンド
- 2.2.1.3 他シーケンスをサスペンド解除
- 2.2.1.4 シーケンスを破棄
- 2.2.1.5 必ず実行エラーになる

がある．ユースケースのシナリオは，入力イベントを受け取り，制御装置プロセスに対応するサブシステムの関数を呼び出すと記述されている．装置制御プロセスは，イベントに対応する既存システムの関数を呼び出すシナリオを持つ．装置制御プロセスに対して入力イベントに対する期待出力の既存システムの呼出しを確認し，機能を確認する．

5.2.2 ユースケーステストに基づくインクリメンタルテスト

本 OJL において，リファクタリング対象領域を広げながら，機能復元を繰り返す反復開発を行った．復元される機能に限定したテストを実現するためにユースケーステストに基づくインクリメンタルテストを実施した．復元される機能はユースケース図に追加される．追加されたユースケースのシナリオから，関連するユースケースの特定を行い，関連するユースケースに限ったテストを可能にする．追加されたユースケースは，

- 2.3 プロセス間の同期
- 2.1.1 複数の同一プロセスを生成する

である．二つの機能を実現するために，イベントに識別子が追加された．識別子はどの装置制御プロセスのイベントかを特定する ID である．追加・変更されたモジュールの関数に対して，制御フローテストを適用し，追加・変更されたモジュールのコードに対してバグの検出・動作保証を行う．図 4 にユースケース図とソフトウェアアーキテクチャの関連を示す．

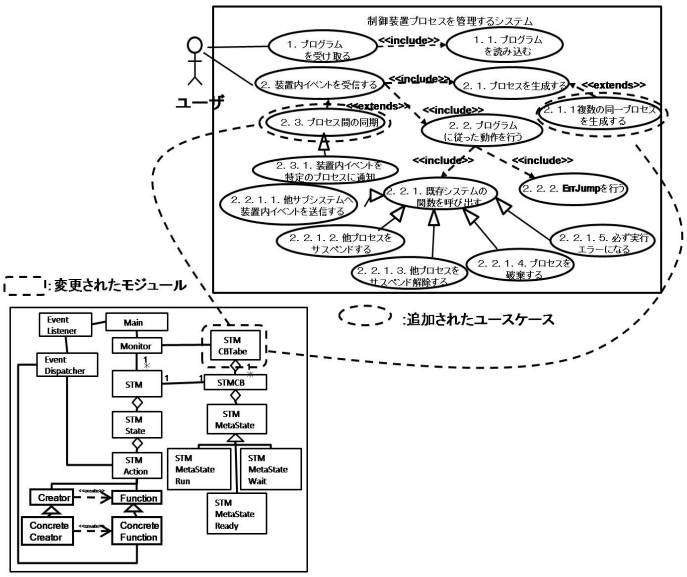

図 4 ユースケースとソフトウェアアーキテクチャの関連

イベントの処理を行うモジュールに対してテストを行い，モジュールの振舞いを保証した．モジュール統合時において，変更されたモジュールに対する動作保証が行われていることから，追加されたユースケースシナリオに基づいたテストが可能である．

ユースケース「2.3 プロセス間の同期」のシナリオは，イベントをプロセス間の同期によって，適切な装置制御プロセスにイベントを送信すると記述されている．イベントを適切な装置制御プロセスに対してイベントを渡すことで，装置制御プロセスは「2.2.1 既存システムの関数を呼び出す」を行う．「2.2.1 既存システムの関数を呼び出す」は既に保証されており，イベントが適切に割り振られる

ことで，適切なサブシステムの関数が呼び出されることを確認し，機能を保証する．
ユースケース「2.1.1 複数の同一プロセスを生成する」のシナリオは，イベントに対応する装置制御プロセスが既に存在するとき，新たな装置制御プロセスを生成すると記述されている．従って，イベントに対応する装置制御プロセスが生成され，既存システムの関数が呼ばれることを確認する．
既に保証されている機能を考慮し，追加されたモジュールに対するインクリメンタルテストによって，テストの必要な点に限定したテストが可能になる．

5.3 システムテスト

システムテストは，一般にリファクタリングでは行わない．部分的なコードの変更では非機能要求にはシステムへの影響は少ないからである．本 OJL では構造の大規模な変更を行っており，機能を保証するユースケーステストでは保証できないことから，システムテストにおいて非機能要求に対するテストを行う．装置制御プロセスを管理するシステムでは，長期間におけるシステムの稼働を保証する耐久性が求められる．耐久性を保証するために，メモリリーク，高負荷 (CPU, メモリの使用率) での動作保証を確認する．メモリリークは，使用メモリが開放されない問題で，メモリ監視ツールを用いて，システムを動作させ，メモリの使用状況を確認する．高負荷 (CPU, メモリの使用率) での動作保証は CPU，メモリの資源に対して負荷が高いとき，システムが動作することをテストし，動作保証する．

6 考察

6.1 提案するテストプロセスの妥当性の確認

オブジェクト指向リファクタリングでは，システムの振舞いを保証する．本 OJL において，既存コードをソフトウェアアーキテクチャに対応付けるリファクタリングである．一般に行われるリファクタリングのテストであるコードに対するテストでは，システムの振舞いを保証することができない．システムの仕様を基に，機能に対する保証を行うことでシステムの振舞いを保証した．
システムの機能に対するテストを行うさい，入力イベントの組合せが膨大である．テストケースを削減し，機能保証を行う必要がある．
ユースケーステストによる機能テストを用いて，テストケースを削減しながら，機能保証を行うテストプロセスを提案した．単体テストにおいてホワイトボックステストを選択し，ソフトウェアアーキテクチャを基に，モジュールに対する動作保証を行った．結合テストでは実現した機能が動作することを確認し，機能保証を行った．システムテストにおいて内部構造の大規模な変更から，システムの非機能要求を保証した．
また，本 OJL はリファクタリング対象領域を広げながら，機能復元を繰り返す反復開発である．復元される機能に限定したテストを実現するためにユースケーステストに基づくインクリメンタルテストを行い，ユースケースとソフトウェアアーキテクチャの対応関係を明確にすることで，追加・変更モジュールの動作保証を行い，復元された機能に対する保証が可能になり，システムの機能保証を行うテストプロセスを提案できた．

6.2 振舞いの保証の考察

本 OJL のテストにおいて，システムの振舞いが変化していないことを保証するテストであることを考察する．
一般のリファクタリングにおける既存コードに対するテストでは，本 OJL におけるリファクタリングに対して，システムの振舞いを保証することが難しい．本 OJL においてシステムの振舞いを示す仕様を基に，仕様に示す機能に着目したテストを行ってきた．
本 OJL のテストは，内部構造の変更に対して，単体テストにおいて，内部構造に対するバグの検出・動作保証を行った．システムの機能に対する保証として，結合テストで，システムの機能に対する保証を行うユースケーステストを行い，システムに対する機能保証を行った．
本 OJL のリファクタリングにおけるテストは，システムの機能に対する保証であり，システムの振舞いに対する保証とは異なる．本 OJL では，既存コードからシステムの振舞いを考慮した仕様復元を行っている．復元した仕様を基にテストすべき機能を決定する．システムの振舞いについて考慮した機能テストを実現し，システムの動作を確認していることからシステムの振舞いの保証として十分と考える．

7 おわりに

本 OJL では，ソフトウェアアーキテクチャに基づくソフトウェア開発プロセスにおけるテストプロセスの考察を行ってきた．オブジェクト指向リファクタリングを行う上で，システムの振舞いを保証するために，一般のリファクタリングでは保証できないことから，システムの機能に着目し，テストプロセスを実現し，システムの振舞いに対する保証を行った．
今後の課題として，反復開発によって発見されるバグを基に，機能テストに対して，バグの検出箇所を考慮した重点的なテストの実施が必要である．

参考文献

[1] G.Booch, “Object-oriented development”,*Proc.IEEE Transaction on Software Enginnering*,Vol12,p.211-221,1986.

[2] Martin Fowler, *Refactering Improving the Design of Existing Code*, Addison-Wesley, 1999.

[3] Beizer Boris, *Software Testing Techniques, 2nd Edition*, Van Nortrand Reinhold, 1990.

[4] Black Rex, *Pragmatic Software Testing: Becoming an Effective and Efficient Test Professional*, 2007.

[5] Rick D.Craig, Stefan P.Jaskiel, *Systematic Software Testing*, 2004.

[6] 野呂昌満，張漢明，坂野将秀，太田将吾，安江基規，”オブジェクト指向による SECS 通信制御ソフトウェアの開発”,2007