修 士 論 文

アスペクト指向を適用した
組み込みシステムテスト方法の研究

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

楊 明睿

2011 年 3 月

修 士 論 文

アスペクト指向を適用した
組み込みシステムテスト方法の研究

指導教官　落水浩一郎 教授

審査委員主査　落水浩一郎 教授
審査委員　鈴木正人 准教授
審査委員　青木利晃 准教授

北陸先端科学技術大学院大学
情報科学研究科情報科学専攻

0910070 楊 明睿

提出年月: 2011 年 2 月

概 要

ソフトウェア品質への要求は年々高まっている。ソフトウェアの品質を確保するため、テストをするのは非常に重要である。そのため、ソフトウェア開発におけるテストの重要性も年々高まっている。テストソフトウェアのテストにおけて、システムテスト、統合テストと単体テストの三つの段階に分ける。各段階では、それぞれの担当者によってテストケースを作成し、テストを実施する。各段階におけるテストデータには重複があり、時間が無駄になる場合があり、系統的なテスト方法が必要である。本研究では、アスペクト指向を利用して、テストデータの冗長性をなくすための系統的なテスト方法を提案する。

目 次

図 目 次

表 目 次

第1章　序論

1.1　背景

本学「高信頼組込みシステムコース」の PBL 演習において、コンロ制御ボードシステムを設計し、実現した。コンロ制御ボードの開発においては、安全性に関する品質をいかに充足するかが重要である。安全性に関する品質を保証するためには、単体テスト、統合テスト、システムテストの三つテストを実施する必要がある。しかしながら、単体テスト、統合テスト、システムテストの三つの段階は別々で設計し、実施するので、重複するテストデータが設計された。このことはテストの効率にひいてはソフトウェア開発全体の効率にまで影響を与える。このためには、単体テスト、統合テスト、システムテストの各段階において、系統的なテストを実施する必要があるが、その手法は明らかでない。

本研究では Ivar Jacobson が提案した「ユースケースによるアスペクト指向ソフトウェア開発法」におけるユースケーススライスの概念を利用して、テストケースを系統的に作成する手法を提案する。

1.2　本研究の目的

本研究の目的は組込みシステムに関して、「安全性」を保証するための系統的なテスト手法を整備することである。コンロ制御ボードシステムを対象とする。安全性を確保するため、単体テスト、統合テスト、システムテストの各段階で、系統的なテストを実施する必要がある。本研究では、ユースケーススライスをアスペクトとしてとらえる Ivar Jacobson が提案した「ユースケースによるアスペクト指向ソフトウェア開発法」に注目し、ユースケーススライスをもとに単体テスト、統合テスト及びシステムテストのテストデータを系統的に作成し、テストデータ全体の量を削減する手法とテスト対象を明確する手法を提案する。

1.3　論文の構成

本論文の構成は以下のとおりである。

- 第二章　ソフトウェアテスト

ソフトウェアテスト技法 [2] とソフトウェアテストの技法 [3] に従って、ソフトウェアテストのテスト技法と三つの段階のテストを紹介する。

- 第三章　ユースケースによるアスペクト指向ソフトウェア開発 (AOSD)

  ユースケースによるアスペクト指向ソフトウェア開発 [4] に従って、ユースケースによる AOSD を紹介する。

- 第四章　テストにおける問題

  テストにおける重複問題とテスト対象を明確する問題を記述する。

- 第五章　単体テスト、統合テスト、システムテストにおける問題を解決する方法

  重複問題とテストの対象を明確する問題に対する、解決方法を提案する。

- 第六章　評価

  コンロ制御ボードシステムにより、実験する。

- 第七章　結論

  本研究についてまとめ、今後の課題を述べる。

- 付録

  アスペクト指向によるコンロ制御ボードシステムの設計

第2章　ソフトウェアテスト

2.1　ソフトウェアテストの概要

ソフトウェアテストとは、プログラム中のバグをできるかぎり多く発見することを目標として、プログラムを実施し、正しく動作するかどうかを確認する作業のことである。作業はテスト仕様書作成とテスト実施の二つの段階に分けられる。テスト仕様書作成にはテストケース設計を含んでいる。テストケースの作成には同値分割と限界値分析がよく利用される。

一般に、ソフトウェアテストはホワイトボックスとブラックボックスの二種類に分けられる。

ホワイトボックステストとは、システムの内部の構造を理解した上でそれら一つ一つが意図した通りに動作しているかを確認するソフトウェアテスト方法である。つまり、プログラムの全ての部分が、プログラム記述者の意図通りに動作していることを確認するテストである。システムの機能よりも内部構造の整合性を重視したテストである。

ブラックボックステストとは、システムの内部構造とは無関係に、外部から見た機能を検証するソフトウェアテスト方法である。つまり、入力と出力だけに着目し、様々な入力に対して仕様書通りの出力が得られるかどうかを確認する。内部構造の整合性よりもシステムの機能を重視したテストである。

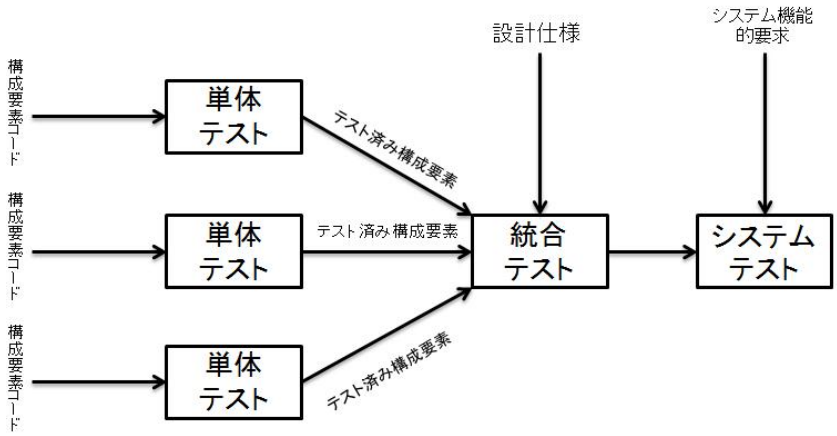


図 2.1: テストの各段階

図 2.1 は、テストの各段階を示す。ソフトウェアテストは単体テスト、統合テストとシステムテストの三つの段階に分けられる。単体とは意味を持った動作をする最も小さなソフトウェアの単位であり、単体テストとはひとつのプログラムの中のサブプログラム、サブルーチン、またはプロセジュアのテスト過程を言う [1]。単体テストはホワイトボックステストを利用して行われる。

統合テスト [2] は、構成要素の集合が単体テストされたあと、次は構成要素中のインタフェースが定義されていて正しく処理されるかをテストすることである。統合テストはホワイトボックステストとブラックボックステスト両方を利用して行われる。

情報が設計通りに構成要素の中に取り込まれていることが確信できれば、要求された機能性を持っていることを保証するためのテストを行う。このようなテストを、システムテストという。システムテストはブラックボックステストを利用して行われる。

2.2　テストの設計技法

テストケースは、テストのために必要な前提条件、操作、それによって起こる結果を記載し、それによりプログラムが正しく動いているかどうかを確認するために使用するものである。単体テストや結合テストなど、すべてのテストはテストケースをもとに計画が立てられる。完全なテストケースの作成は不可能であるが、より少ない労力でそれに近いテストケースを作成する技法がある。同値分割、限界値分析、原因－結果グラフ、命令網羅、判定条件網羅、条件網羅などがある。これらの技法では、それぞれ長所と短所があり、組み合わせて使用するのがいい。テストケースの作成には同値分割と限界値分析をよく利用されている。

2.2.1　同値分割

同値分割とは、仕様上、コンポーネントやシステムの動作が同じと見なせるものについて入力定義域や出力定義域を類別することである。「同値クラス」とは同様の出力結果が得られる入力値の集合である。正常処理を行う同値クラスを「有効同値クラス」と言い、エラー処理を行う同値クラスを「無効同値クラス」と言う。

同値分割により、エラー発見率を確保したままテストケースの個数を減らすことができる。すべての可能な入力のうち、適切なサブセットを選択する。手順としては、同値クラスを識別し、テストケースを設計する。

表 2.1 は同値クラスの識別を示す。表 2.1 には、値と列挙の違いは、値はどれも同じ扱いをするのに対し、列挙はすべての値に対して異なる処理を行う。表 2.2 は同値クラスの識別の例である。

テストケースの作成は下記のとおりに行う。表 2.3 は同値分割によりテストケース設計の例である。

1,　同値クラスに通し番号を付ける。

表 2.1: 同値クラスの識別

入力条件	有効同値クラス	無効同値クラス
値の範囲	テスト対象	範囲以外の値 (複数)
値	指定された値	それ以外の値 (複数)
列挙	個々の値 (複数)	それ以外の値
条件	条件を真とする値	条件を偽とする値

表 2.2: 同値クラスの識別の例

入力条件	入力条件の例	有効同値クラスの例	無効同値クラスの例
範囲	3 から 7	6	1,8
値	1,3,9	1	2,5
列挙	2,5,7	2,3,7	4
条件	奇数	9	6

2, なるべく多くの有効同値クラスをカバーするテストケースを作成する。これをすべての有効同値クラスがカバーされるまで繰り返す。

3, カバーされていない無効同値クラスの内一つだけをカバーするテストケースを作成する。これをすべての無効同値クラスがカバーされるまで繰り返す。

表 2.3: 同値分割によりテストケース設計の例

	テストケース
有効	(6,1,2,9),(6,1,3,9),(6,1,7,9)
無効	(1,1,2,9),(8,1,2,9),(6,2,2,9),(6,5,2,9),(6,1,4,8),(6,1,2,6)

2.2.2 限界値分析

限界値分析は、同値クラスの境界値付近を入力値として選んでテストする方法である。限界値分析は同値クラス分類をより精密にした技法である。一般的に、ソースコードのバグは境界値付近に潜在することが多いといわれているので、同値分割と限界値分析は組み合わせて使用されることが多い。手順としては、同値クラスを識別し、テストケースを設計する。表 2.4 は限界値分析を示す。表 2.5 は限界値分析の例である。

表 2.4: 限界値分析

入力条件	有効同値クラス	無効同値クラス
値の範囲	下端、上端	下端より下、上端より上
値	最小、最大	最小より下、最大より上

表 2.5: 限界値分析の例

入力条件	条件の例	有効同値クラスの例	無効同値クラスの例
値の範囲	100¡x¡200	100,200	99,201
値	3,4,5	3,5	2,6

最初に入力を同値クラスに分類次に出力が同値クラスの境界値になるような入力を考える。テストケースの作成には、なるべく多くの有効同値クラスをカバーするテストケースを作成する。カバーされていない無効同値クラスの内一つだけをカバーするテストケースを作成する。表 2.6 は限界値分析によりテストケース設計の例である。

表 2.6: 限界値分析によりテストケース設計の例

	テストケース
有効	(100,3),(100,5),(200,3),(200,5)
無効	(99,3),(201,3),(100,2),(100,6)

2.3 単体テスト

単体テストとは、プログラムを検査する作業の中でも、プログラムを手続きや関数といった個々の機能ごとに分割し、そのそれぞれについて動作確認を行う手法のことである。単体テストにより、個々の機能を果たすためのプログラム部品（プログラムモジュール）がそれぞれしっかりと動作しているかを検証する。モジュールのインタフェースや処理手順が正しく動作するか、仕様書の通りに動作しているかなどが単体テストによって確認される。単体テストで不備が発見された際には、コーディングの段階に戻ってプログラムの書き直しが行われる。そして再度単体テストが行われ、問題ないことが確認されたら、それらのモジュール同士を組み合わせた場合にもうまく動作するかどうかを検証する「統合テスト」の段階に引き渡される。最初に入力を同値クラスに分類次に出力が同値クラスの境界値になるような入力を考える。テストケースの作成には、なるべく多くの有効

同値クラスをカバーするテストケースを作成する。カバーされていない無効同値クラスの内一つだけをカバーするテストケースを作成する。

オブジェクト指向の場合は、クラスに対してテストを行う。メソッドの仕様に沿ってテストケースを作成し、単体テストを実施する

2.4 統合テスト

統合テストとは、システム開発におけるプログラムの検証作業の中でも、手続きや関数といった個々の機能を結合させて、うまく連携・動作しているかを確認するテストのことである。

統合テストでは、個々の機能を果たすためのプログラム部品（プログラムモジュール）を組み合わせて、データの受け渡しがうまく行われているか、コードの記述様式は揃っているか、データを授受するタイミングはずれていないか、といった点が確認される。統合テストで不備が発見された際には、再度コーディングが行われる。

統合テストで不備が発見された際には、コーディングの段階に戻ってプログラムの書き直しが行われる。そして再度統合テストが行われ、問題ないことが確認されたら、ソフトウェアシステムが要求を満たしているかを検証する「システムテスト」の段階に引き渡される。

オブジェクト指向の場合は、クラスの集合に対してテストを行う。仕様に沿ってテストケースを作成し、統合テストを実施する。

2.5 システムテスト

システムテストとは、完成したソフトウェアシステムに対して行うテストであり、ソフトウェアシステムが要求を満たしているかの評価を行う。システムテストでは入出力に注目してテストを行うので、ブラックボックステストである。

仕様に沿ってテストケースを作成し、システムテストを実施する。

第3章　ユースケースによるアスペクト指向ソフトウェア開発(AOSD)

3.1　関心事

利害関係者とは、エンドユーザー、プロジェクトスポンサー、開発者など誰でも構わない。

関心事とは、利害関係者が関心をもつあらゆるものである。例えば、機能要求、非機能要求、設計制約などがある。

関心事の分離とは、コンピュータサイエンスでは、問題を分解して、小さくすることである。理想的には、様々な関心事をなんらかのモジュールに明確に分離し、各モジュールを一つずつ個別に検討し開発する。

分類語彙表のカバレジを調査するために、2つの実験を行なった。

3.2　ユースケース

ソフトウェア工学では、ユースケースはシステムの機能について説明するもので、アクターとシステムの相互作用を表す。言葉をかえると、ユースケースはユーザーの観点から連続にアクターによって起動される連続するイベントを説明する。

ユースケースモデルには、アクター、ユースケース、およびそれらの関連が含まれる。ユースケースモデルは要求モデルの一種である。ユースケースモデリングの目標は、ユーザーの関心事を分離することである。図3.1はホテル管理システムのユースケースの例である。

3.3　横断的な関心事

横断的な関心事とは、複数おコンポーネントに影響を及ぼす関心事である。多くの関心事は識別できる。そして、個々のクラス、パッケージ、サービスといったコンポーネントにより、効果的に局所化することもできるが、コンポーネントに局所化できず、ほかのコンポーネントに影響する横断的な関心事もある。

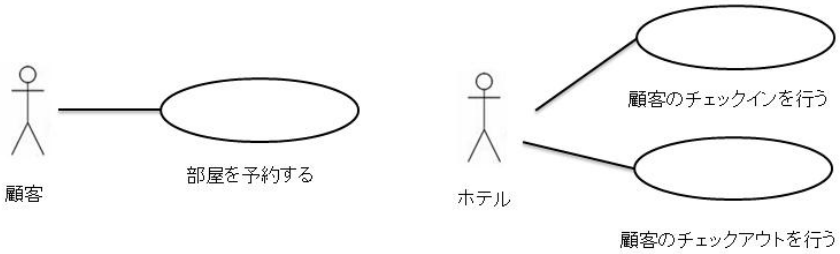


図 3.1: ユースケースの例 (ホテル管理システム)

機能要求を扱う横断的な関心事がある。ユースケースとして特定される機能要求を実現する際に、機能が複数のコンポーネントを横断することがよくある。このように、ユースケース自体も横断的な関心事である。ピアと拡張がある。

ピアは互いに異なる関心事であり、すべてのピアの重要性はほかのピアと同等である。各コンポーネント (クラス) には、さまざまな関心事を満たすための実装 (コード) が含まれている。これをもつれ合いと言う。一つの特定の関心事を実現するコードが複数のコンポーネントに横断する場合がある。これを散らばりと言う。図 3.2 はホテル管理システムのピアの例である。図 3.3 は横断することを表す。

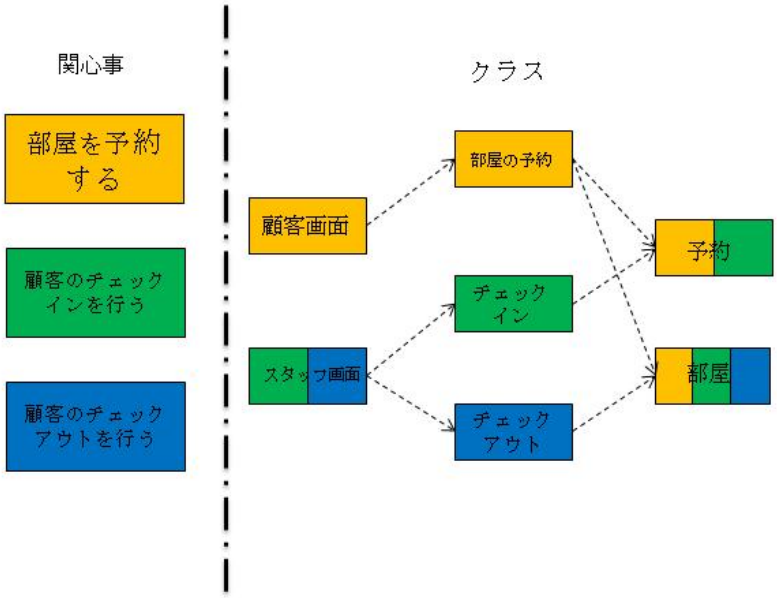


図 3.2: ピアの例 (ホテル管理システム)

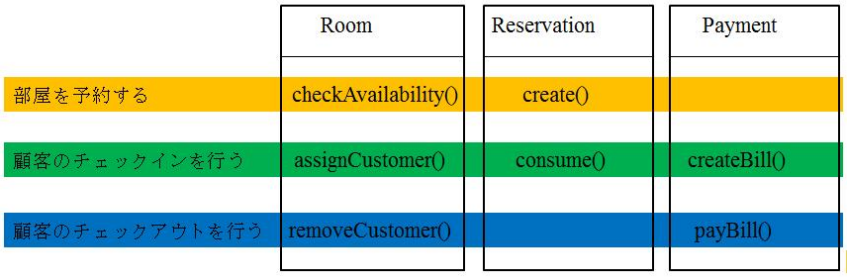


図 3.3: 横断することを表すピアの例 (ホテル管理システム)

拡張はベースの上に定義されるコンポーネントであり、追加のサービスや機能を表す。基底と拡張は別々に表すことが一般的であるが、拡張を実装する際に、問題が発生する。図 3.4 は拡張の例である。

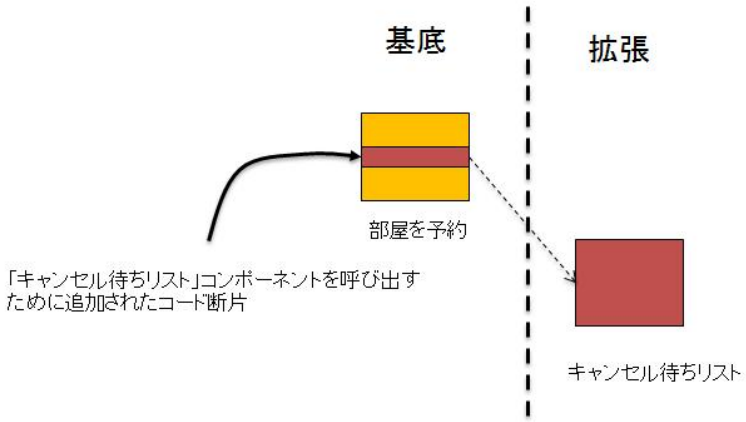


図 3.4: 拡張の例 (ホテル管理システム)

3.4 ユースケースによる AOSD の概要

アスペクト指向ソフトウェア開発 (Aspect-Oriented Software Development、AOSD) とは、要求定義から分析、設計、実装、テストに至るまで、アスペクトによってソフトウェアシステムを開発する手法である。AOSD では、さまざまな種類の関心事を網羅しながらシステム全体をうまくモジュール化する。つまり、機能要求、非機能要求、プラットフォームに依存する事柄などをうまくモジュール化し、それらを互いに分離する。すべての関心事を分離することにより、理解しやすい構造を持ったシステムを構築できる。また

利害関係者の変わりゆくニーズに対応するために、構成と拡張を簡単に行えるシステムを構築できる。

ユースケースによる AOSD では、ユースケースモデリングを使用して横断的な関心事をモジュール化できる。つまり、ユースケースを使用して利害関係者の関心事の要求を表現することから、アスペクトを使用してその要求を実装することへとシームレスに移行できる。

簡単に言うと、ユースケースによる AOSD の実行は以下のようになる。まず、ユースケースを使用し、関心事をモデル化し、捕捉する。次に、ユースケースモジュールによる関心事を分離する。さらに、ユースケーススライスという観点でユースケースを設計し、システムのアーキテクチャを確立する。そして、アスペクト技術を使用してユースケーススライスやユースケースモジュールを構成し、システムの完全なモデルを形成する。図 3.5 はホテル管理システムにより、ユースケースからクラスへの例を示す。

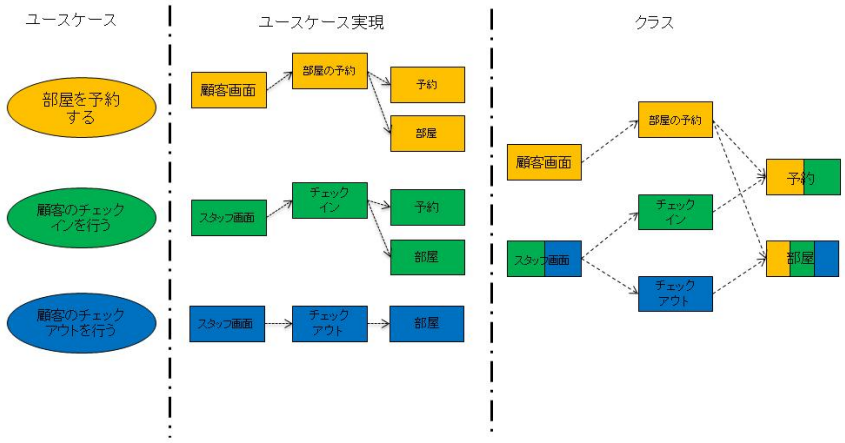


図 3.5: ユースケースからクラスへ (ホテル管理システム)

3.4.1 アスペクトによるピアユースケースの分離

ピアユースケースは相互に関連を持たないが。相異なるユースケースである。しかし、それらの実現は重複し、同一のクラス上に責務を課する。各クラスについて、ユースケースの実現に必要なフィーチャー (属性、操作、関係) を識別する。各クラスにおいて、複数のさまざまなユースケース実現に対応するフィーチャーをまとめられない。もつれ合いが起きる。そのかわりに、各ユースケースの実現に必要なクラスのフィーチャーを一つに照合するため、一つのユースケーススライスに特化する。図 3.6 はホテル管理システムにより、ユースケーススライスによるピアユースケース実現の構成をあらわす。

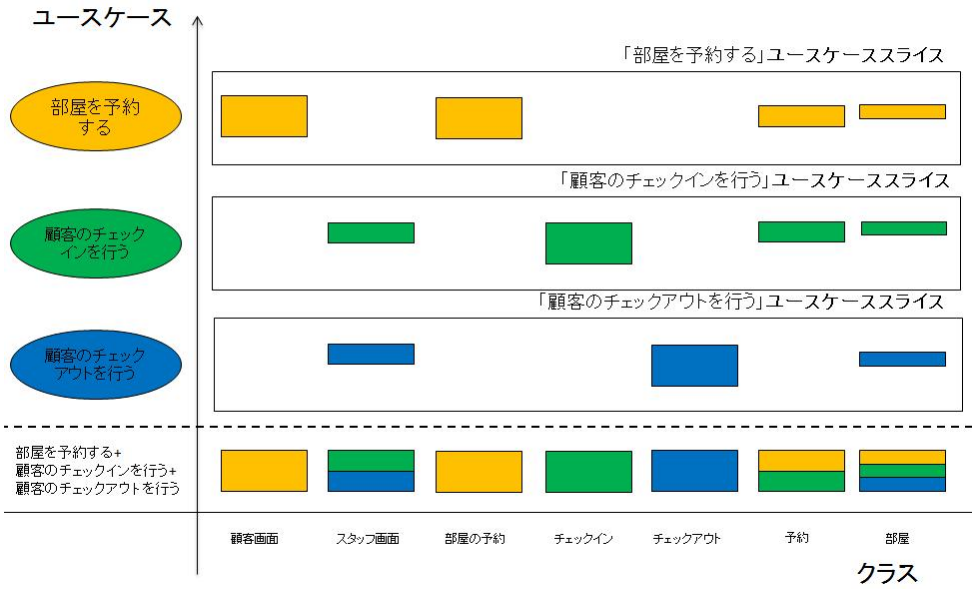


図 3.6: ユースケーススライスによるピアユースケース実現の構成（ホテル管理システム）

図 3.6 には、横軸は、システム内のクラスを識別する要素構成を示す。縦軸はユースケース構成を示す。横の列はそれぞれ、その例のユースケースを実現するために必要なクラスの拡張を含むユースケーススライスを示す。

3.4.2 アスペクトによる拡張ユースケースの分離

拡張ユースケースは、基底ユースケースを拡張する。拡張ユースケースは基底ユースケースで定義された拡張ポイントへと挿入されなければならない一連のアクションを持つ。この一連のアクションは、拡張ユースケースフローと呼ばれる。拡張ユースケースフローは基底ユースケースから分離した関心事であるから、拡張ユースケースの中で示す。図 3.7 はホテル管理システムにより、拡張ユースケースの例である。

図 3.8 はホテル管理システムにより、拡張ユースケース実現と基底ユースケース実現の構成を表す。横軸は、システム内のクラスを識別する要素構成を示す。縦軸はユースケース構成を示す。横の列は、ユースケースと対応するユースケース実現が必要とするクラス拡張を示す。「キャンセル待ちリストを扱う」ユースケーススライスは二つの操作拡張を持つ。これらの操作拡張は、「顧客画面」クラス拡張と「部屋の予約」クラス拡張の中でそれぞれ定義されている。

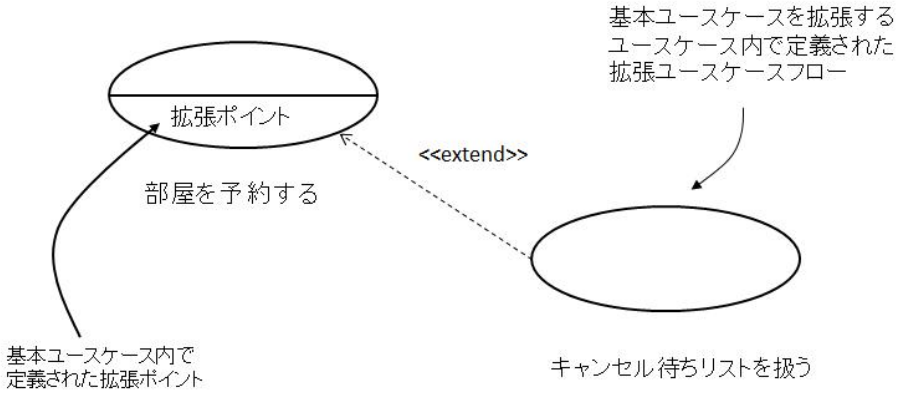


図 3.7: 拡張ユースケースの例 (ホテル管理システム)

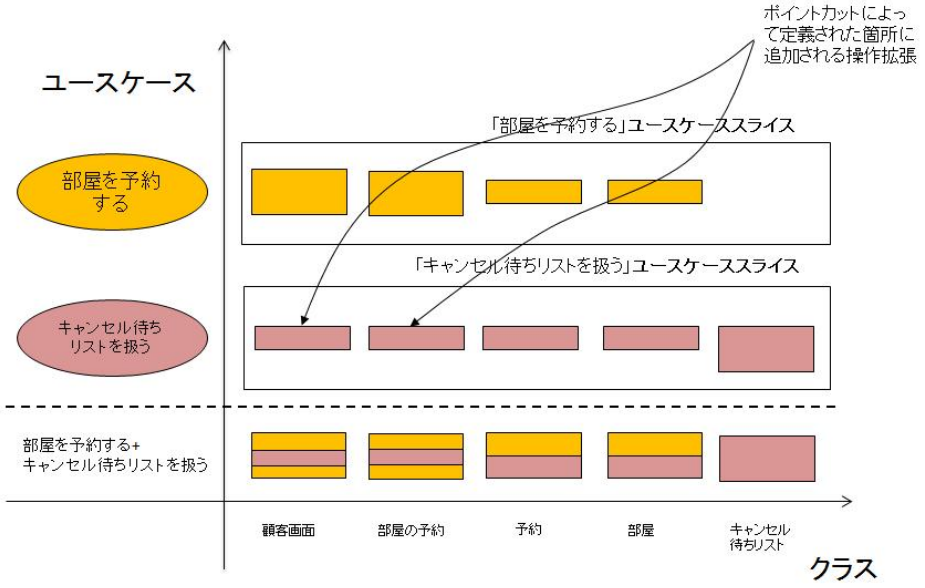


図 3.8: 拡張ユースケース実現と基底ユースケース実現の構成 (ホテル管理システム)

3.5 ユースケースモジュール

「ユースケースによるアスペクト指向ソフトウェア開発」では、システムを構築するとき、ユースケースモデル、分析モデル、設計モデル、実装モデルごとにシステムを作ることではなく、図 3.9 のとおり、ユースケースごとにシステムを作る。

様々なモデルを通じて徐々にユースケーススライスを洗練していくことによって、シス

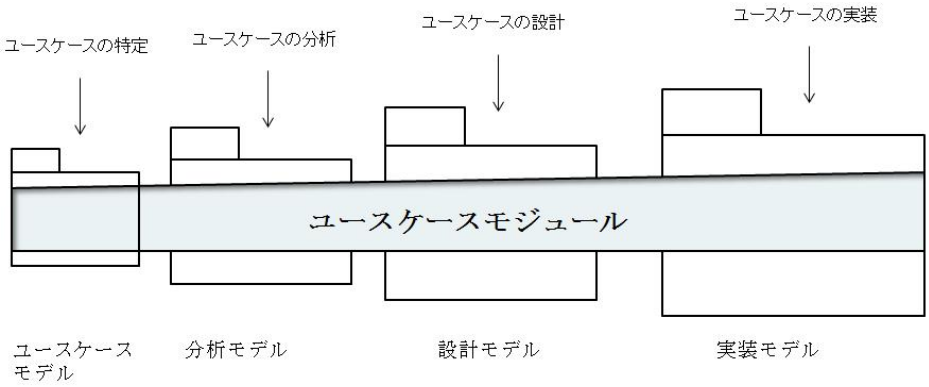


図 3.9: 開発におけるユースケースモジュール

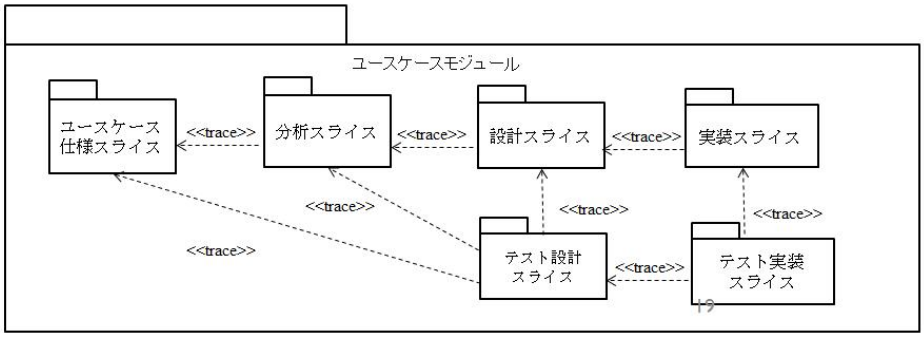


図 3.10: ユースケースモジュール内のスライス

テムを開発する。ユースケースモジュールには、ユースケースモデルのユースケース仕様スライス、分析モデルの分析スライス、設計モデルの設計スライス、実装モデルの実装スライスが含まれている。テストを実施するための設計モデルのスライスと実装スライスもある。図 3.10 はユースケースモジュール内のスライスを示す。

第4章　テストにおける問題

4.1　階層的テスト

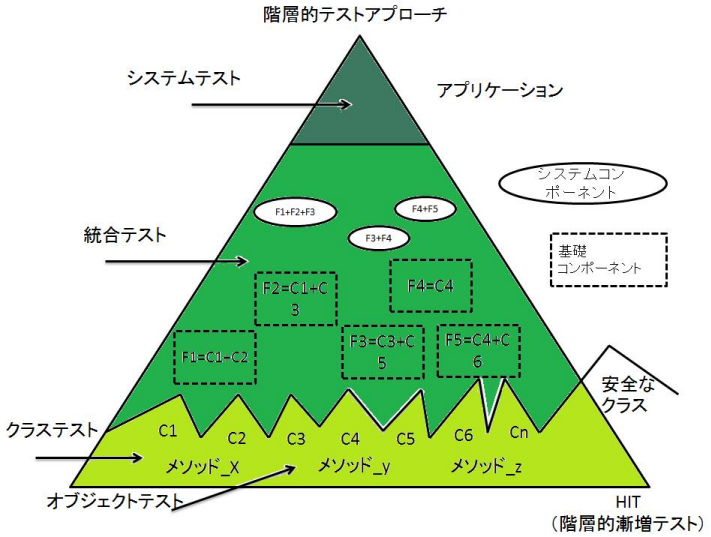


図 4.1: ユースケースモジュール内のスライス

図 4.1 は、完全なテストシステムの構造を表す。テストシステムはクラステスト、統合テストとシステムテストに構成される。

4.1.1　単体テスト

図 4.2 は単体テストを示す。オブジェクト指向の場合、単体テストは各クラスに対するテストを実施する。言い換えれば、オブジェクト指向の場合は、すべてクラスのすべてメソッドに対して、テストを実施する。同値分割と限界値分析を利用して、テストケースを設計する。

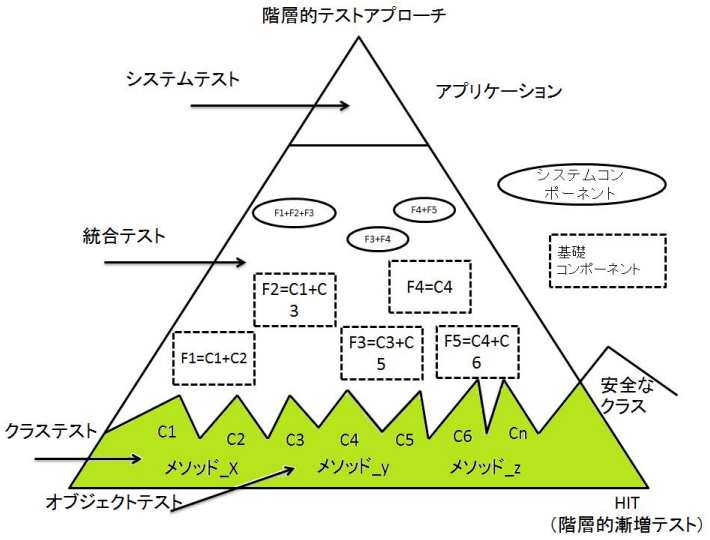


図 4.2: 階層的テストアプローチの単体テスト

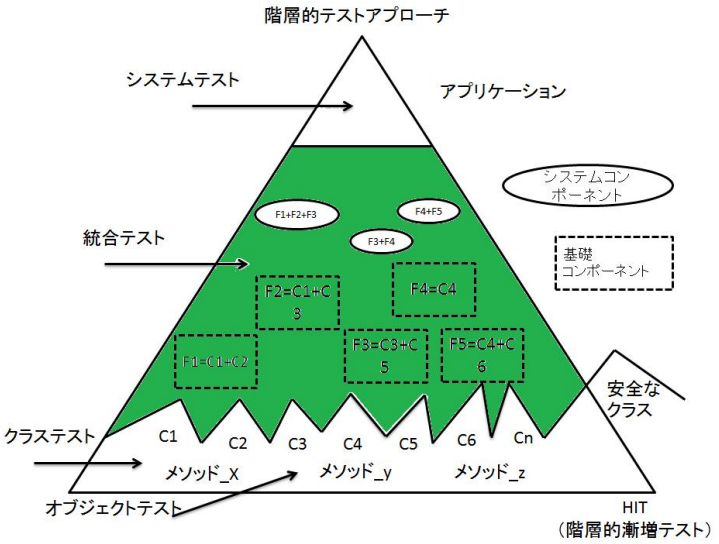


図 4.3: 階層的テストアプローチの統合テスト

4.1.2 統合テスト

図 4.3 は統合テストを示す。オブジェクト指向の場合、統合テストはいくつのクラスを結ばれたシステムコンポーネントである。依存木に利用して、複数のクラスを基礎コンポーネントを構成して、インタフェースをテストする。依存木のすべての可能なパスをテストする。同値分割と限界値分析を利用して、テストケースを設計する。

4.1.3 システムテスト

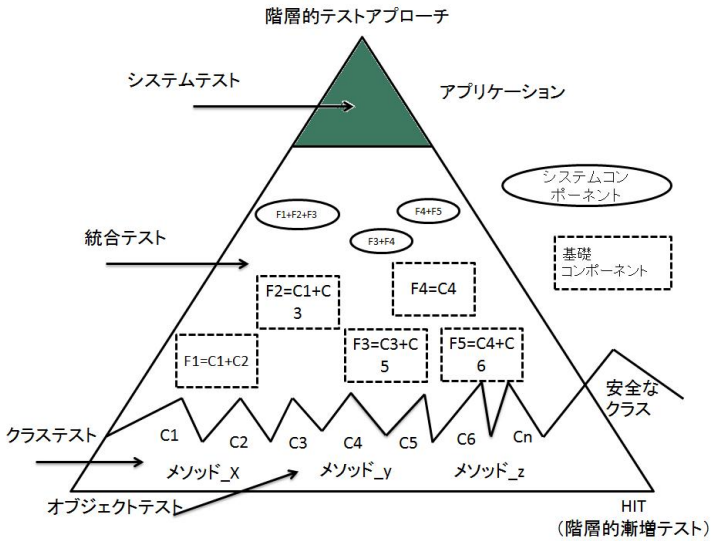


図 4.4: 階層的テストアプローチのシステムテスト

図 4.4 はシステムテストを示す。オブジェクト指向の場合、システムテストはシステムの機能に満たすかどうかをテストする。同値分割と限界値分析を利用して、テストケースを設計する。

4.2 テストにおける問題の記述

テストにおいて、テストケースの重複とテスト対象の選択という二つの問題がある。下記に詳しく記述する。

4.2.1 重複問題

単体テストは、クラスのメソッドについて、テストを実施する。メソッドのアルゴリズムやデータに注目する。統合テストは、システムコンポーネントについて、テストを実施する。インタフェースに注目する。システムテストは、システムについて、テストを実施する。システムの入出力に注目する。

しかしながら、上記の三つのテストでは、同じ対象 (同じプログラム構造：クラス) についてテストを実施する。そのため、同じクラスは複数のテスト段階でテストされた可能性がある。注目する点は違うが、テストケースを重複に設計する可能性がある。

下記の例により、説明する。

図 4.5 のとおり、単体テストは 1 から 7 までの七つのクラスを一つずつテストする。

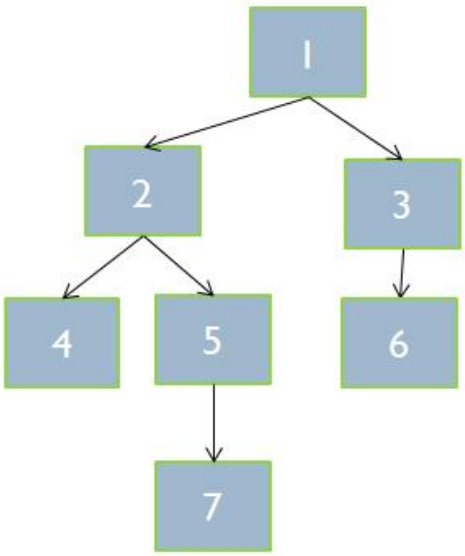


図 4.5: 単体テストの例

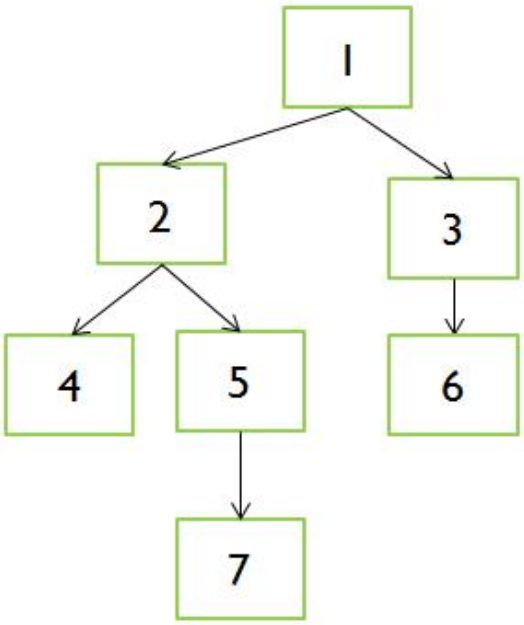


図 4.6: 統合テストの例

図 4.6 は統合テストを示す。統合テストではビッグバンテスト、トップダウンテスト、ボトムアップテストとサンドイッチテストいった四つのテスト方法がある。統合テストは

複数のクラスを組み合わせて、テストを実施する。

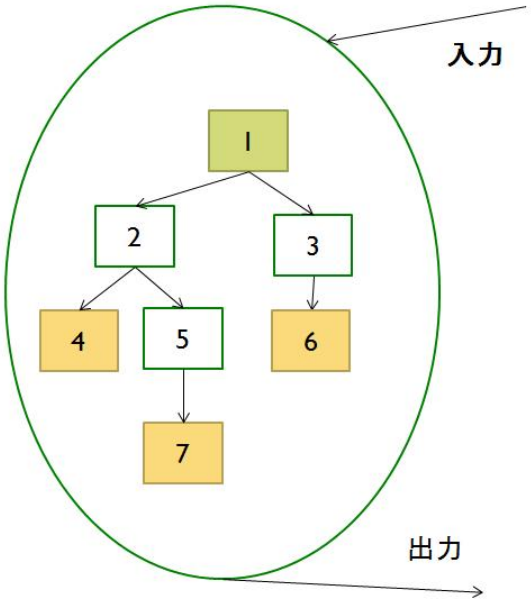


図 4.7: システムテストの例

図 4.7 はシステムテストを示す。システムテストはシステムをブラックボックスとして、テストを実施する。入出力に注目する。入出力を対応するクラスは 4,6,7 である。

具体的には下記の通り、簡単な例により説明する。

関数 1：

getMaxInTwo(int a,int b)

if a ＞＝ b return a;

else return b;

関数 2：

getMaxInThree(int a,int b,int c)

a=a+1;

int max=getMaxInTwo(a,b);

max=getMaxInTwo(max,c);

関数 1 の単体テストには (3,2)、(1,3)、(2,2) という三つのテストケースが必要である。関数 2 の単体テストには、(1,2,3) という一つのテストケースが必要である。関数 1 と関数 2 について統合テストを実施する。関数 1 は a と b 二つのパラメータを関数 2 に送る。a と b の間は (a ＞ b)、(b ＞ a)、(a=b) 三種類の場合の関係がある。(1,2,3) というテストケースは (b ＞ a) と (a=b) しかテストしていない。それは完全なテストと言えない。そのため、(2,2,2) というテストケースが必要になる。つまり、関数 1 と関数 2 の統合テストと関数 2 の単体テストのテストテストには、重複部分がある。

まとめると、図 4.8 のように、重複問題を表している。

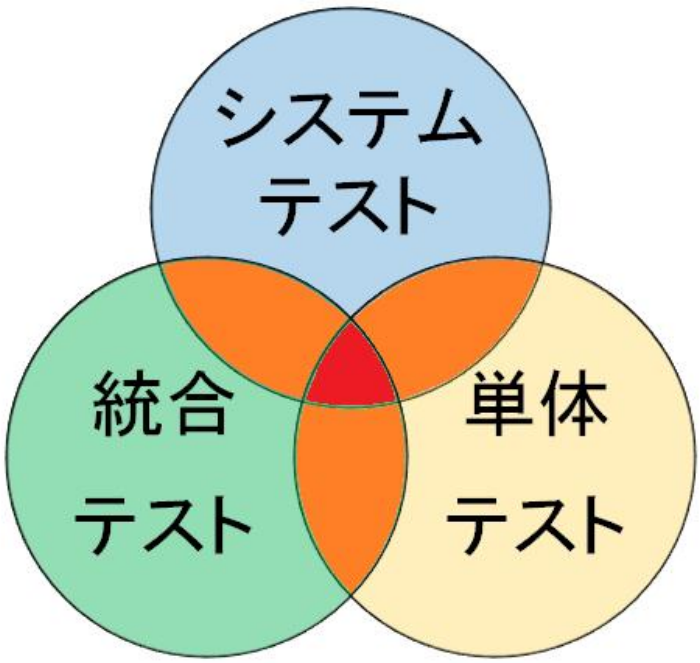


図 4.8: テストにおける重複問題

単体テストにおいては、すべてのクラスについてテストを実施する。統合テストにおいては、単体テストを実施した上、一つのクラスをもとに、一つずつを増加して、インタフェースについて、テストを実施する。つまり、すべてのクラスについてもう一度テストを実施する。システムにおいては、システムあるいはすべてのクラスを一つのモジュールとして、テストを実施する。したがって、単体テスト、統合テスト、システムテストにおいて、同じ部分について二回三回テストを行う可能性がある。

4.2.2 テストの対象を選択する問題

単体テストは、各クラスについて、テストを実施する。統合テストは、システムコンポーネントについて、テストを実施する。システムテストは、システムについて、テストを実施する。

しかしながら、統合テストにおけるシステムコンポーネントとシステムテストにおけるテスト単位を選択する方法が必要である。

そのため、下記の方法を提案する。

システムテスト設計では、各ユースケースについて、テストケースを設計し、テストを実施する。IBM の「ユースケースからテストケースへの追跡」を利用する。

統合テスト設計では、各ユースケース内のクラス群について、テストケースを設計し、

テストを実施する。

単体テスト設計では、各ユースケーススライス内のクラス群の各クラスについて、テストケースを設計し、テストを実施する。

上記を実現するため、分析からテストまでユースケースモジュールによるシステムを開発できる「ユースケースによるアスペクト指向ソフトウェア開発」が利用できると考える。

第5章　単体テスト、統合テスト、システムテストにおける問題を解決する方法

5.1　重複問題について

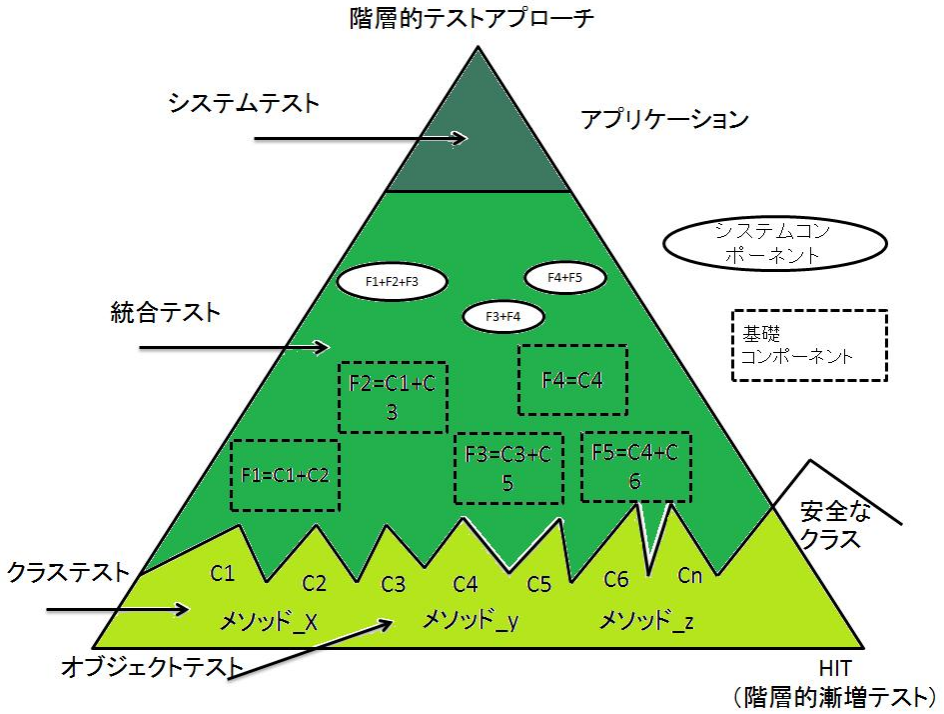


図 5.1: 階層的テストアプローチ

第四章で記述されたテストにおける重複問題について、下記のテスト方法を提案する。

まず、テストを実施する手順としては、単体テスト、統合テスト、システムテストの順番でテストを実施する。そして、各システムコンポーネントについて、依存木を利用する。次には、単体テストでは、依存木中の一番上位クラスと一番下位クラスについて、テストを実施する。統合テストでは、各システムコンポーネントの中で、単体テストでテス

トした一番上位のクラスをドライバとして、下位のクラスについてトップダウンテストを実施する。一番下位クラスをスタブとして、上位クラスについてボトムアップテストを実施する。システムテストでは、各システムコンポーネントについて、入出力のテストを実施する。それぞれ段階のテストにおいて、具体的なテスト方法は表 5.1 により示す。表 5.2 は継承関係がある場合にのやり方を示す。

表 5.1: テスト方法

テストレベル	テスト対象	テスト内容	テスト方法
単体テスト	依存木の一番上位 と一番下位のクラス	すべてのメソッド	
統合テスト	依存木の単体 テストされたクラス以外のクラス	すべてのパス	サンドイッチテスト
システム	クラス群	すべてのフロー	入出力

表 5.2: 継承関係がある場合のテスト方法

条件	テスト対象
オーバーライドされた	サブクラスでオーバーライドされたメソッド
オーバーライドされていない	スーパークラスのメソッド

提案した重複問題を解決する方法により、図 5.1 に表現された階層的テストアプローチは図 5.2 のとおりに改善する。

5.2 重複問題について

「ユースケースによるアスペクト指向ソフトウェア開発」では、ユースケースを実現するため、ユースケーススライスという新しいモジュールを作成した。システムの開発において、分析からテストまで、ユースケースモジュールによりシステムを開発する。「ユースケースによるアスペクト指向ソフトウェア開発」を利用し、図 5.1 に表現されたシステムは下図のとおりに構成する。

図 5.3 では、「ユースケースによるアスペクト指向ソフトウェア開発」において、クラス、ユースケーススライスとユースケースの間の関係を表している。横軸は、システム内のクラスを識別する要素構成を示す。縦軸はユースケース構成を示す。横の列はそれぞれ、その例のユースケースを実現するために必要なクラスの拡張を含むユースケーススラ

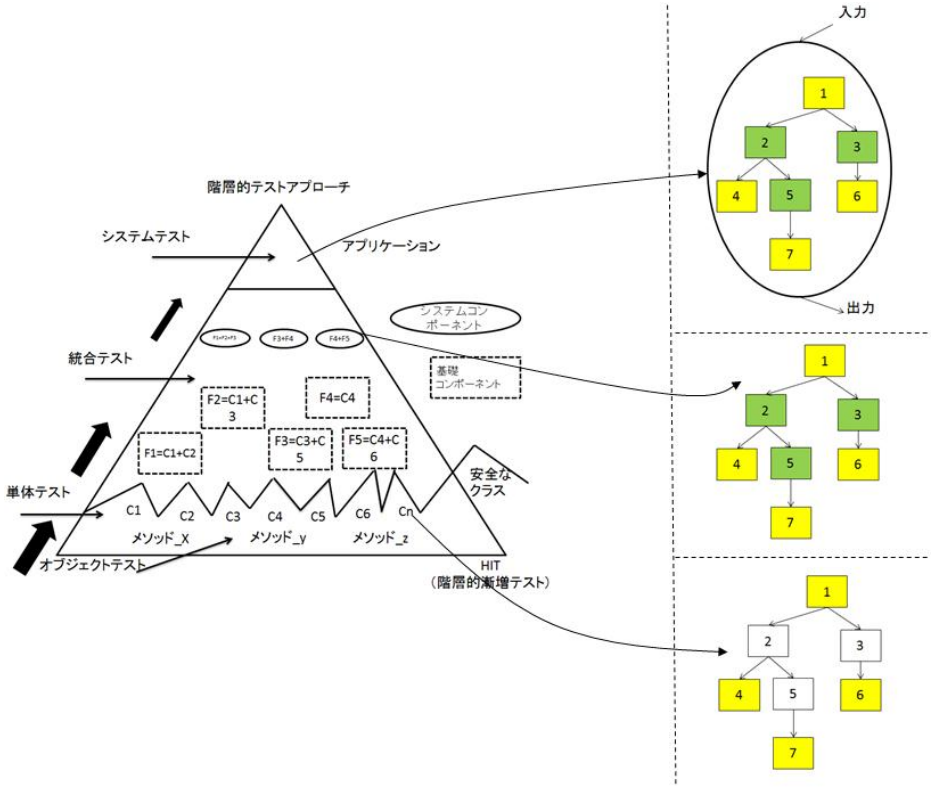


図 5.2: 重複問題を解決する方法により改善した階層的テストアプローチ

イスを示す。本研究では、単体テストの段階には、各ユースケーススライスに関わるクラスあるいはメソッドについてテストを実施する；統合テストの段階には、各ユースケーススライスについて、テストを実施する；システムテストの段階には、各ユースケースについて、テストを実施するテスト方法を提案する。

「ユースケースによるアスペクト指向ソフトウェア開発」を利用した上、図 5.1 に表現された階層的テストアプローチは下図のとおりに改善する。

図 5.4 には、赤い線はシステムを分割する。システムテスト段階では、黄色の線により、システムをいくつのユースケースにわける。統合テストは各ユースケースに関わるクラス群あるいはシステムコンポーネント (ここでは、ユースケーススライスであらわす) を示している。単体テスト段階では、各ユースケーススライスに関わるクラスあるいはメソッドを示している。ユースケースを単位として、単体テスト、統合テスト、システムテストが実施できると考える。

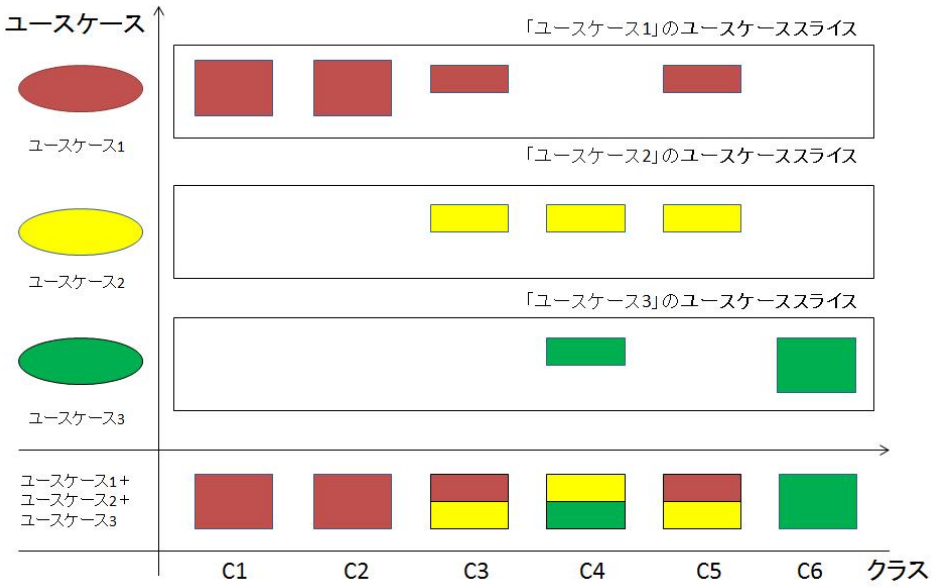


図 5.3: ユースケース実現の構成

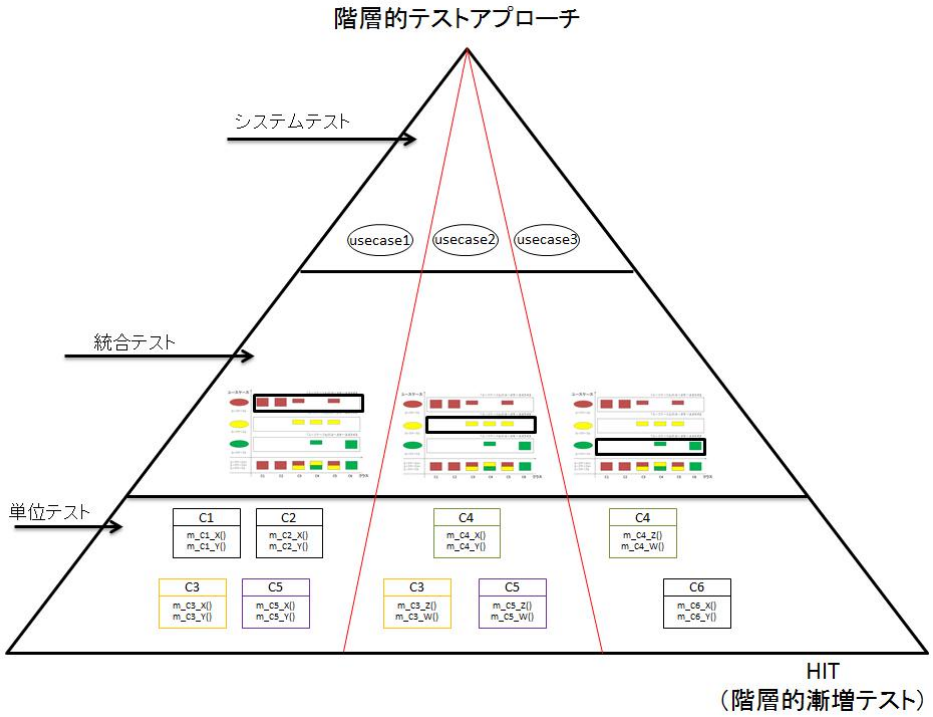


図 5.4: 改善された階層的テストアプローチ

5.3　テスト方法のまとめ

テストにおける重複問題の解決方法とテストの対象を選択する問題の解決方法をまとめて、下記の系統的なテスト方法を提案する。

まず、テストを実施する手順としては、単体テスト、統合テスト、システムテストの順番でテストを実施する。そして、各ユースケーススライスについて、依存木を利用する。次には、単体テストでは、依存木中の一番上位クラスと一番下位クラスについて、テストを実施する。統合テストでは、各ユースケーススライスの中で、単体テストでテストした一番上位クラスをドライバとして、下位クラスについてトップダウンテストを実施する。一番下位クラスをスタブとして、上位クラスについてボトムアップテストを実施する。システムテストでは、各ユースケースについて、入出力のテストを実施する。それぞれ段階のテストにおいて、具体的なテスト方法は表 5.3 により示す。表 5.4 は継承関係がある場合にのやり方を示す。

表 5.3: まとめたテスト方法

テストレベル	テスト対象	テスト内容	テスト方法
単体テスト	依存木 の一番上位と 一番下位のクラス	ユーススケーススライス で定義された すべてのメソッド	
統合テスト	依存木の 単体テストされた クラス以外のクラス	ユーススケーススライス で定義されたクラス間 のすべてのパス	サンドイッチテスト
システム	ユースケース	ユースケースの すべてのフロー	ユースケースから テストケースへの追跡

表 5.4: 継承関係がある場合のテスト方法

条件	テスト対象
オーバーライドされた	サブクラスでオーバーライドされたメソッド
オーバーライドされていない	スーパークラスのメソッド

まとめると、系統的なテスト方法により、図 5.1 に表現された階層的テストアプローチは図 5.5 に改善する。

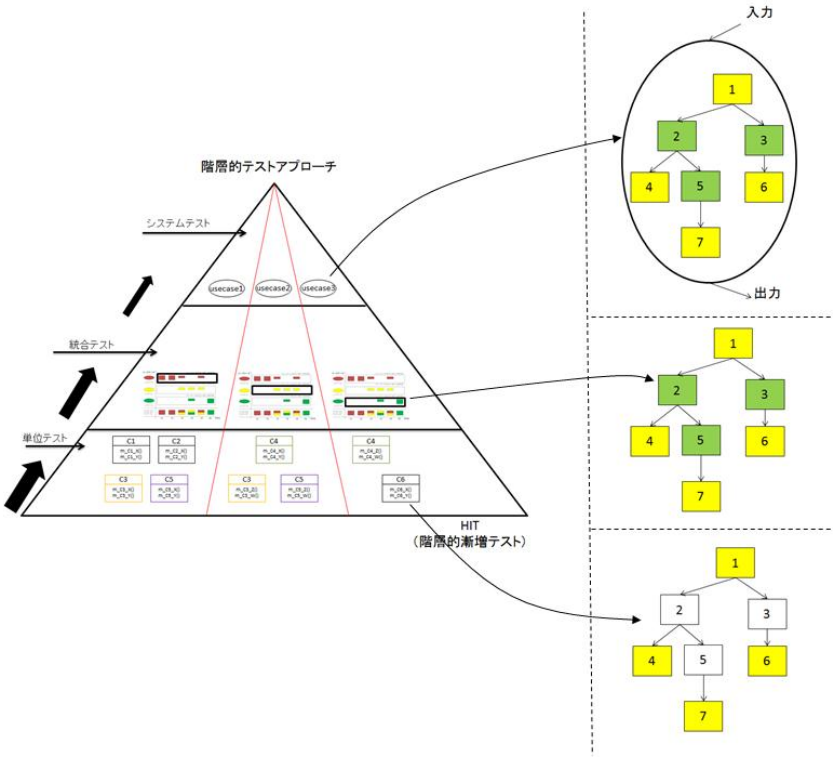


図 5.5: 系統的なテスト方法により改善された階層的テストアプローチ

第6章　評価

本章では、はコンロ制御ボードシステムを対象として、本研究により提案した系統的なテスト方法を利用して下記のとおりに実験を実施する。

6.1　統合テストのテストケース設計事例

本研究に提案された系統的なテスト方法により、統合テストでは、単体テストを実施した後、単体テストにおけるテストを実施されたクラスをもとに、各依存木中それら以外クラスについてサンドイッチテストを実施する。下記はコンロ制御ボードの事例である。表6.1は各ユースケーススライスにおける統合テストのテストケースを示す。

6.2　システムテストのテストケース設計事例

本研究に提案された系統的なテスト方法により、システムテストでは、統合テストを実施した後、入出力についてテストを実施する。下記はコンロ制御ボードの事例である。表6.2は各ユースケースにおけるシステムテストのテストケースを示す。

表 6.1: 統合テストのテストケース設計

ユースケース	クラスとメソッド	テストケース番号	変数 1	変数 2
点火	IgnitionHandler.ignition()		cookerNO	set_sw
		1	1	
		2	1	3
		3	2	
		4	2	4
燃焼	BurningHandler.stableTemp()		cookerNO	
		5	0	
		6	1	
		7	2	
自動温度調整	Switch.setModeSW()		set_sw	
		8	7	
	Switch.TempSW()		set_temp	
		9	160	
		10	180	
		11	200	
	TempHandler.setTemp()		temp	set_temp
		12	179	180
		13	180	180
		14	181	180
コンロを OFF にする	OFFHandler.offCooker()		cookerNO	
		15	0	
		16	1	
		17	2	

ユースケース	クラスとメソッド	テストケース番号	変数 1	変数 2
サンプリング	SamplingHandler.sampling()		timerNO	
		18	1	
		19	2	
		20	3	
エラー処理	ErrDeleHandler.errDele()		errCode	
		21	101	
		22	102	
		23	201	
		24	202	
		25	301	
		26	302	
		27	401	
		28	402	
		29	501	
		30	502	
		31	601	
		32	701	

表 6.2: システムテストのテストケース設計

ユースケース	クテストケース番号	変数 1	変数 2	変数 3
点火		cookerNO	GetLight	
	1	1		
	2	1	3	
	3	2		
	4	2	4	
燃焼		cookerNO	StableTemp	temp
	5	1	110	131
	6	1	110	130
	7	1	110	129
	8	2	110	131
	9	2	110	130
	10	2	110	129
自動温度調整		cookerNO	temp	
	11	1	251	
	12	1	250	
	13	1	249	
	14	2	251	
	15	2	250	
	16	2	249	
コンロを OFF にする		cookerNO		
	17	0		
	18	1		
	19	2		
温度センサー断線エラー		cookerNO	temp	
	20	1	9	
	21	1	10	
	22	1	11	
	23	2	9	
	24	2	10	
	25	2	11	

ユースケース	クテストケース番号	変数 1	変数 2	変数 3
異常過熱エラー		cookerNO	temp	
	26	1	291	
	27	1	290	
	28	1	189	
	29	2	291	
	30	2	290	
	31	2	289	
途中失火エラー		cookerNO	GetLgiht	
	32	1	291	
	33	1	290	
	34	1	189	
	35	1	189	
連続使用エラー		on_A	on_B	GetLgiht
	36	0	0	1
	37	0	0	1
	38	1	1	1
電源電圧低下エラー		voltage		
	39	2.6		
	40	2.5		
	41	2.4		
	42	2.1		
	43	2		
	44	1.9		

第7章　結論

7.1　本研究のまとめ

本研究では、ユースケーススライスをアスペクトとしてとらえるIvar Jacobsonが提案した「ユースケースによるアスペクト指向ソフトウェア開発法」に注目し、ユースケーススライスをもとに単体テスト、統合テスト及びシステムテストのテストデータを系統的に作成し、テストデータ全体の量を削減する手法とテスト対象を明確する手法を提案する。系統的なテスト手法は、ソフトウェアテストを実施するための時間を節約できる。つまり、安全性に関する品質を保証することに利用可能である。

7.2　今後の課題

本研究はコンロ制御ボードシステムを対象として、研究を行う。本研究では、単体テスト、統合テスト、システムテストにおける重複問題とテスト対象を明確する問題の二つの面から、研究を行う。そのため、重複問題を解決する方法に対して、プログラム言語 (C言語、JAVAなど) に関わらず、たくさんのシステムにより、実験する必要がある。テスト対象を明確する方法に対して、異なるプログラム言語 (C言語、JAVAなど) により開発されたシステムと比べることで、テスト対象を明確する方法を利用できるアスペクト指向による開発されたシステムのテストはどの程度有効のかを実験する必要がある。つまり、今後の課題としては、本研究により提案されたテスト方法を実験することに注目する。

謝辞

本研究を行うにあたり、研究目的の確定から完成まで貴重なご指導とご助言頂きました北陸先端科学技術大学院大学　情報科学研究科　落水浩一郎教授に心より深く感謝申し上げます。

本研究の審査員として、大変有益なご意見とご助言頂きました北陸先端科学技術大学院大学　情報科学研究科　鈴木正人准教授と青木利晃准教授に心より深く感謝申し上げます。

最後に学業と生活を支えて頂きました家族、友人に心より深く感謝申し上げます。

皆様、本当にありがとうございました。

参考文献

[1] 大塚俊章、荻野富二夫、 ソフトウェアテスト技術、 UNISYS TECHNOLOGY REVIEW 第 93 号、AUG、2007

[2] ボーリス・バイザー [著]、 Software Testing Techniques、 小野間彰、山浦恒央 [訳]、ソフトウェアテスト技法　第二版、2008

[3] G.J.Myers[著]、The Art of Software Testing、長尾 真,、松尾 正信 [訳]、ソフトウェアテストの技法　第二版、2007

[4] Ivar Jacobson,、Pan-Wei Ng[著]、Aspect-Oriented Software Development With Use Cases、鷲崎弘宜、 太田健一郎、鹿糠秀行、立堀道昭 [訳]、ユースケースによるアスペクト指向ソフトウェア開発、2006

[5] Shel Siegel[著]、Object Oriented Software Testing: A Hierarchical Approach、古宮誠一、広田豊彦 [訳]、オブジェクト指向ソフトウェアテスト技法―リスク管理への技術的 アプローチ、2000

付 録A　コンロ制御ボードシステム

この付録では、設計したコンロ制御ボードシステムを述べる。

A.1　洗い出したシステムフィーチャー

A.1.1　点火と燃焼

点火 SW が押されたらバルブとイグナイタを ON
５秒以内に着火が確認できなければ点火エラーとしてバルブ OFF
着火したらイグナイタ OFF、燃焼中に移行
燃焼中は温度の急上昇 (焦げ付き) を監視
バーナー A のみモード切替 SW が押されたら自動温度調整モードに移行

A.1.2　自動温度調整

SW により目標値を設定する
目標温度になるように火力調整バルブを制御
温度が高くなりすぎたら異常加熱エラーとする

A.1.3　安全性

以下は常時監視、検出すること
異常加熱 (290 度以上を 5 秒間)
センサー断線 (10 度以下を 5 秒間)
途中失火 (着火センサーが OFF5 秒)
電源電圧低下 (2 レベル)
2 時間以上の連続使用
上記のエラー発生時には全てのバルブを OFF にしてガスを止めること

A.1.4　信頼性

点火 SW　 OFF で一般エラーから回復する
電池交換エラーからは回復しない

A.2　ユースケースモジュール設計

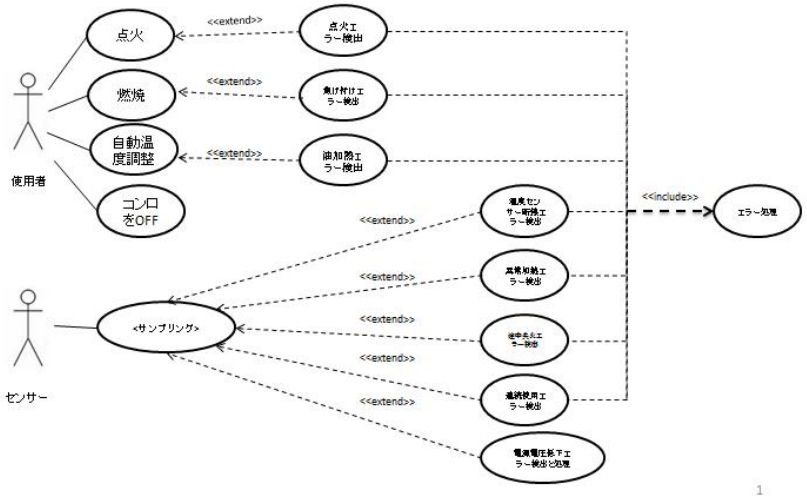


図 A.1: コンロ制御ボードシステムのユースケース図

A.3　ユースケーススライス設計

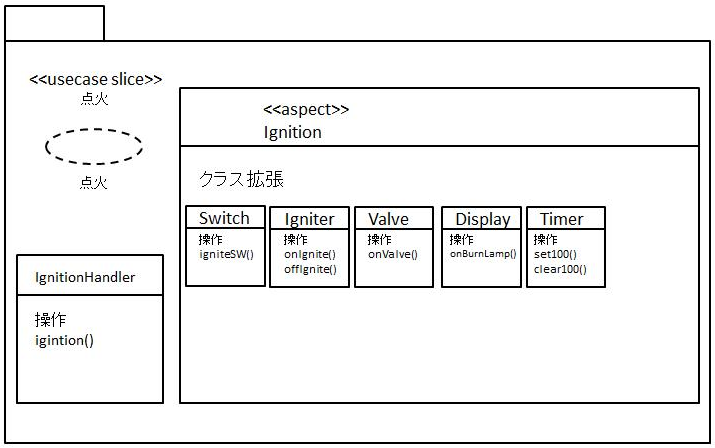


図 A.2: 点火ユースケースのユースケーススライス

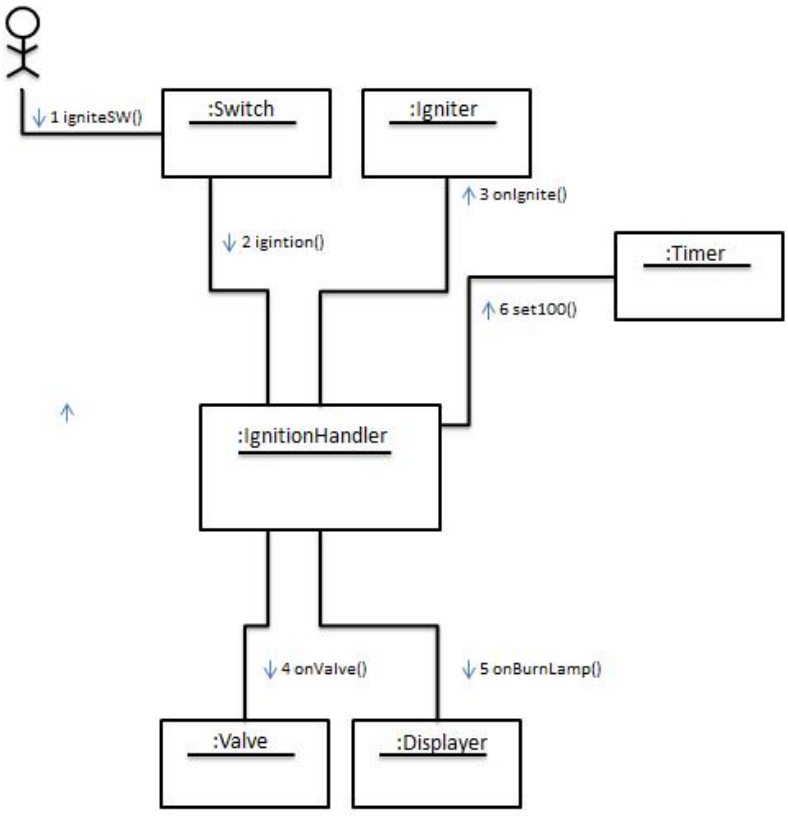


図 A.3: 点火ユースケーススライスのコミュニケーション図

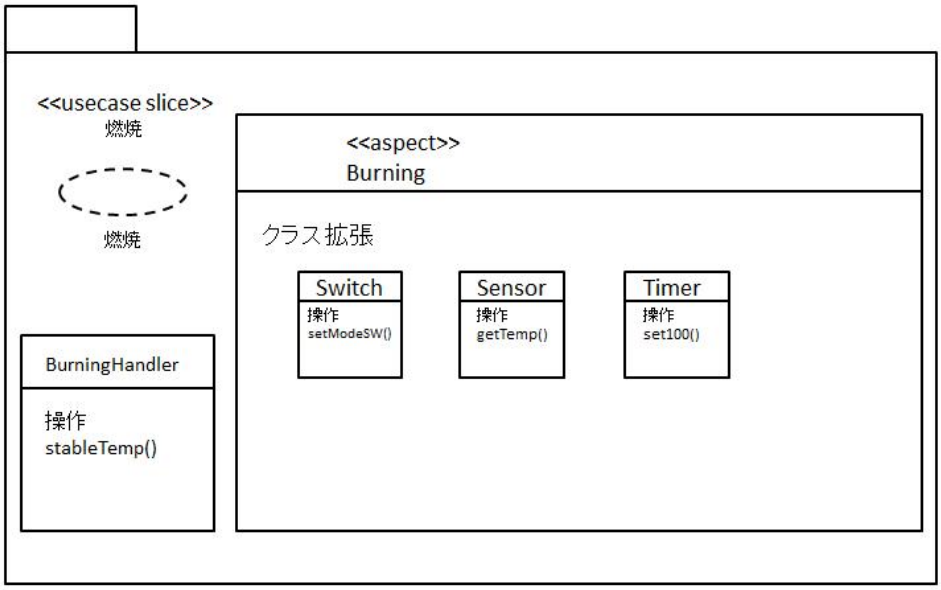


図 A.4: 燃焼ユースケースのユースケーススライス

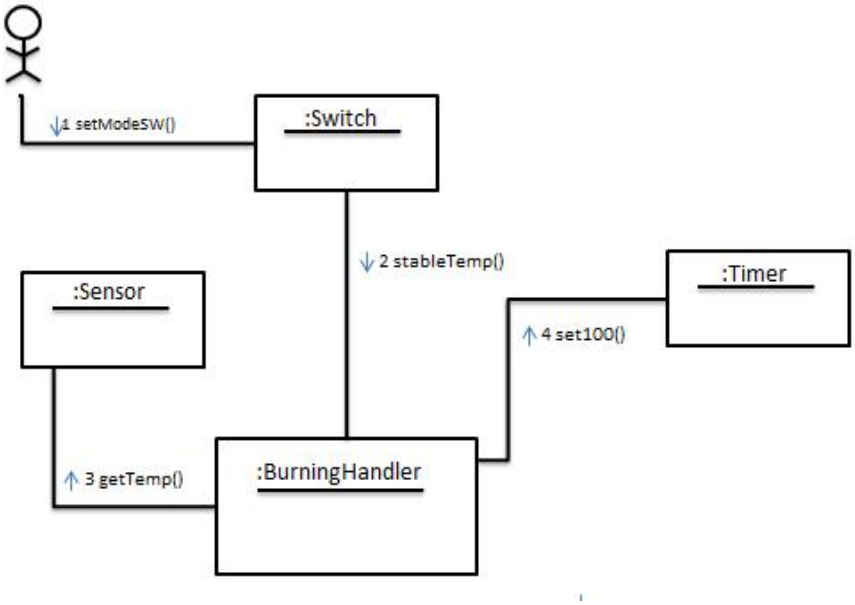


図 A.5: 燃焼ユースケーススライスのコミュニケーション図

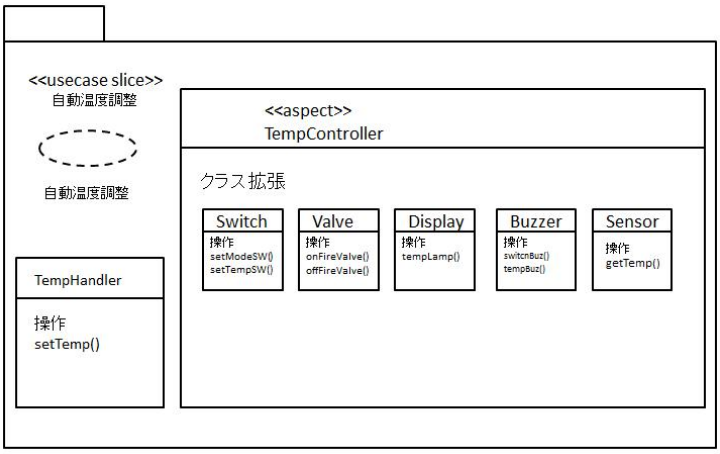


図 A.6: 自動温度調整ユースケースのユースケーススライス

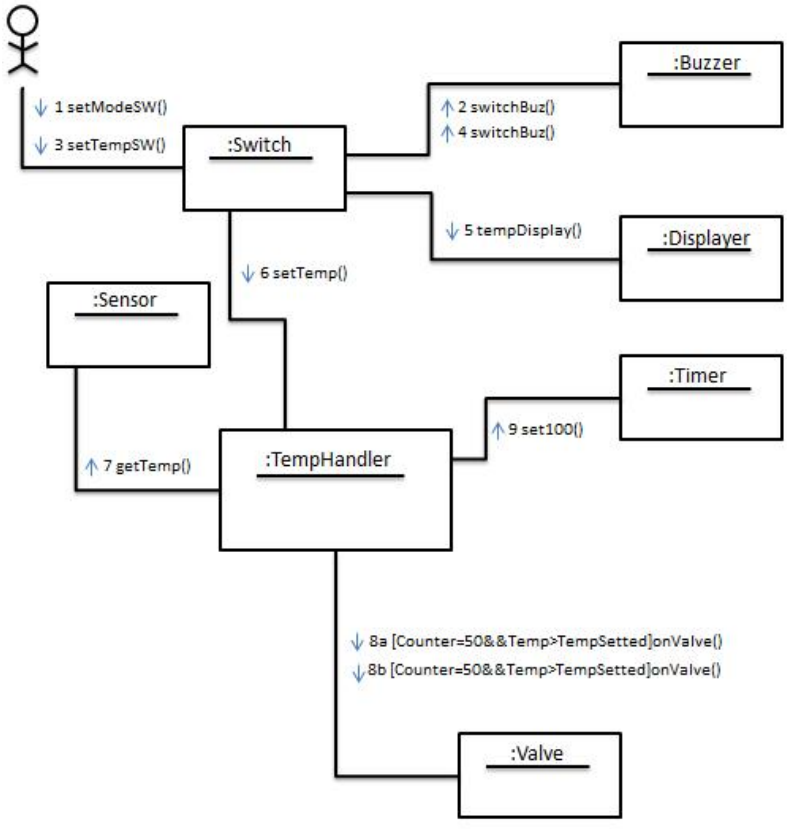


図 A.7: 自動温度調整ユースケーススライスのコミュニケーション図

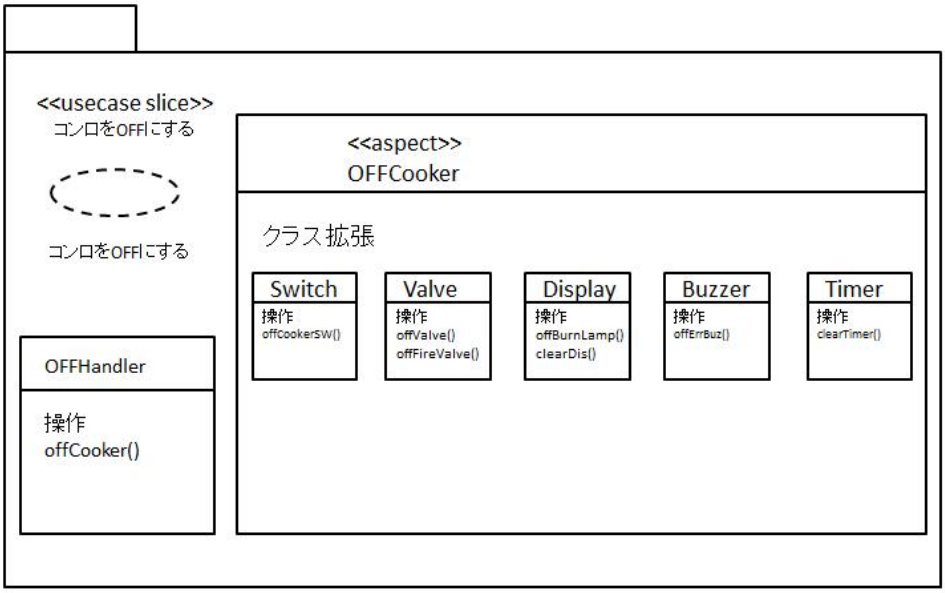


図 A.8: コンロを OFF にするユースケースのユースケーススライス

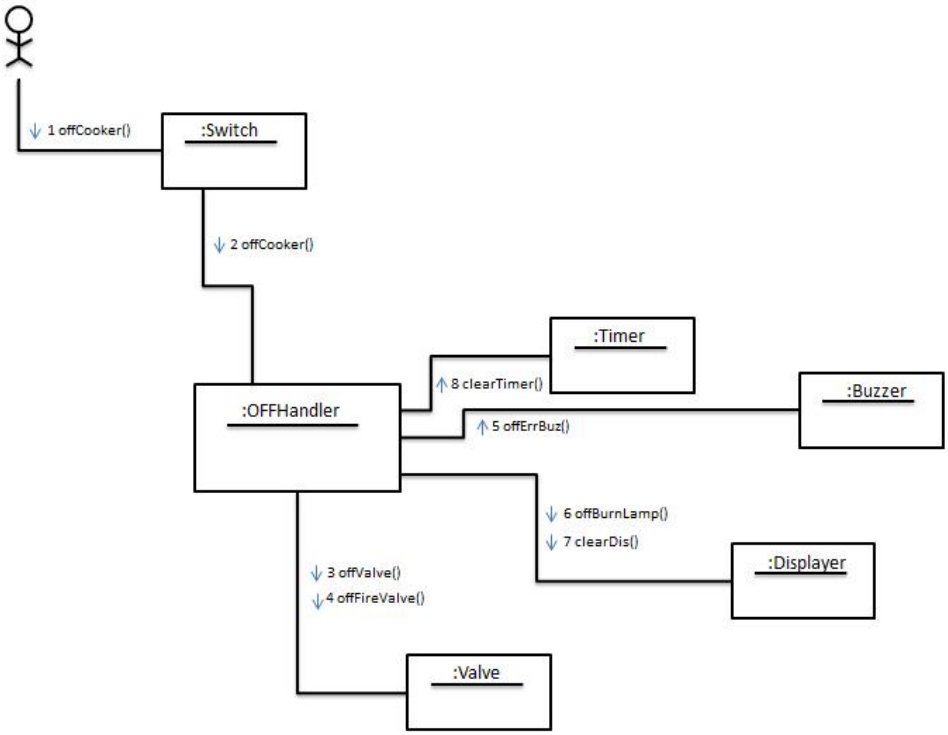


図 A.9: コンロを OFF にするユースケーススライスのコミュニケーション図

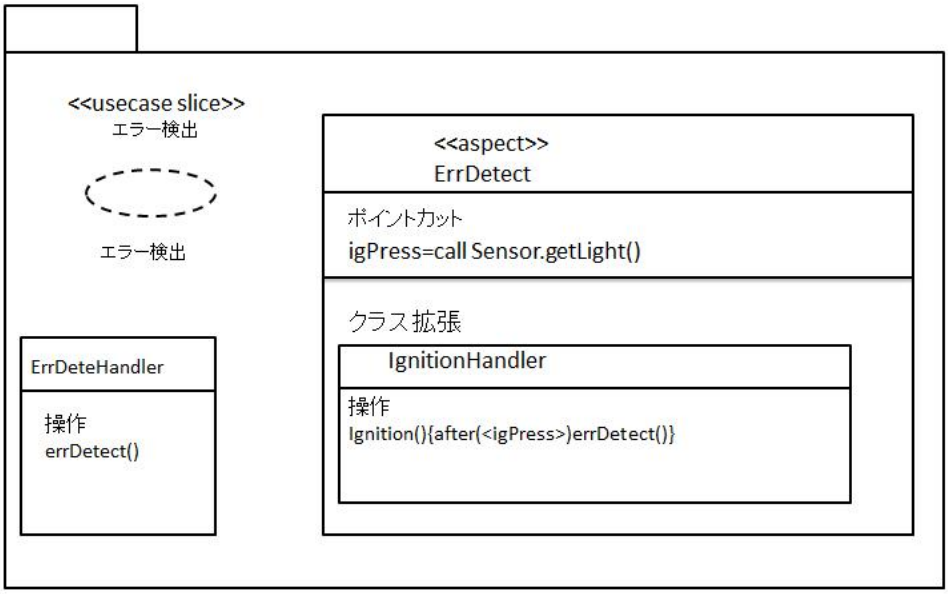
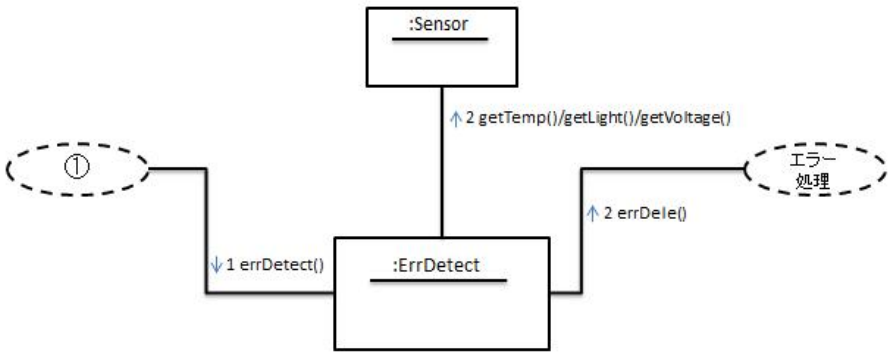


図 A.10: エラー検出ユースケースのユースケーススライス



注訳:①は点火と燃焼、自動温度調整、サンプリングを表示する

図 A.11: エラー検出ユースケーススライスのコミュニケーション図

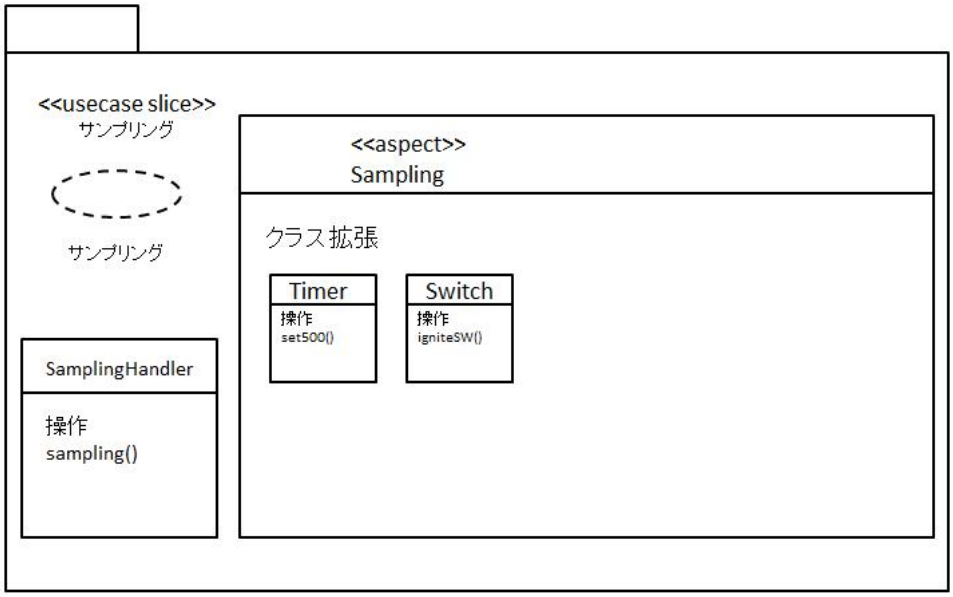


図 A.12: サンプリングユースケースのユースケーススライス

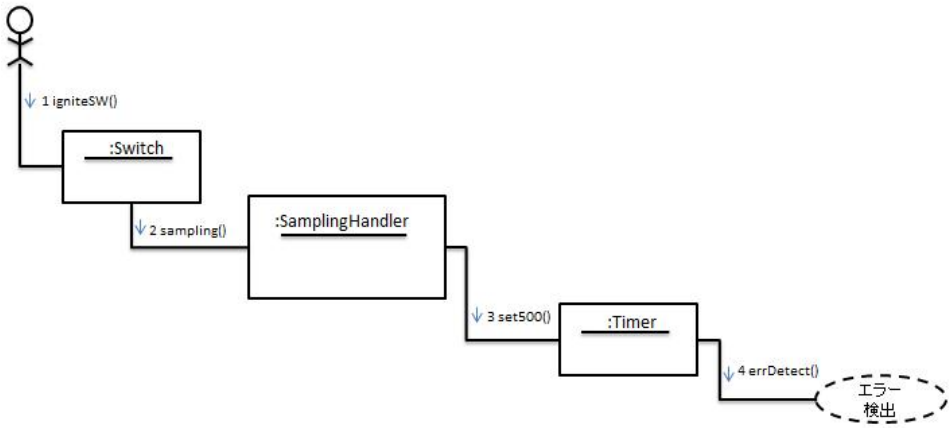


図 A.13: サンプリングユースケーススライスのコミュニケーション図

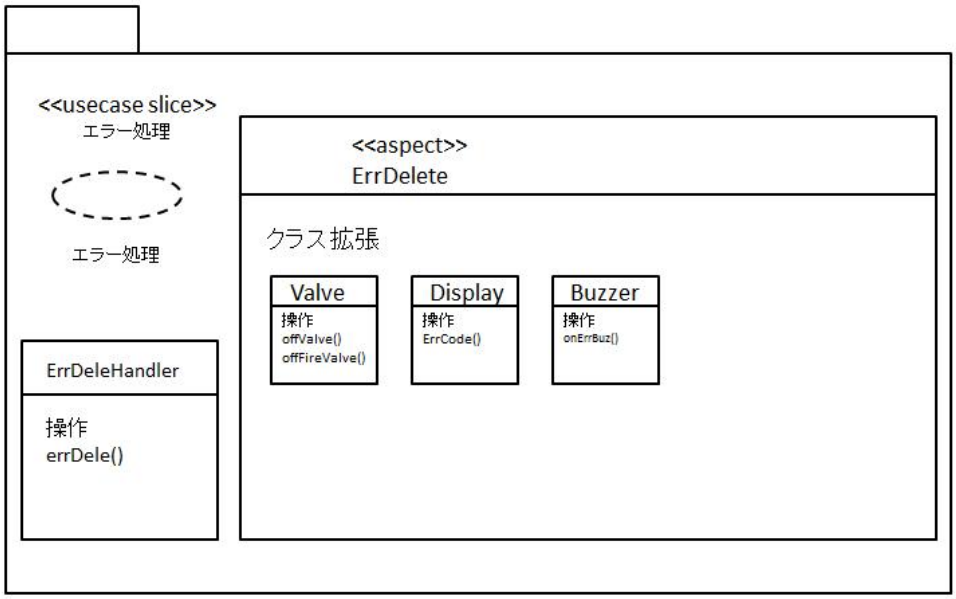


図 A.14: エラー処理ユースケースのユースケーススライス

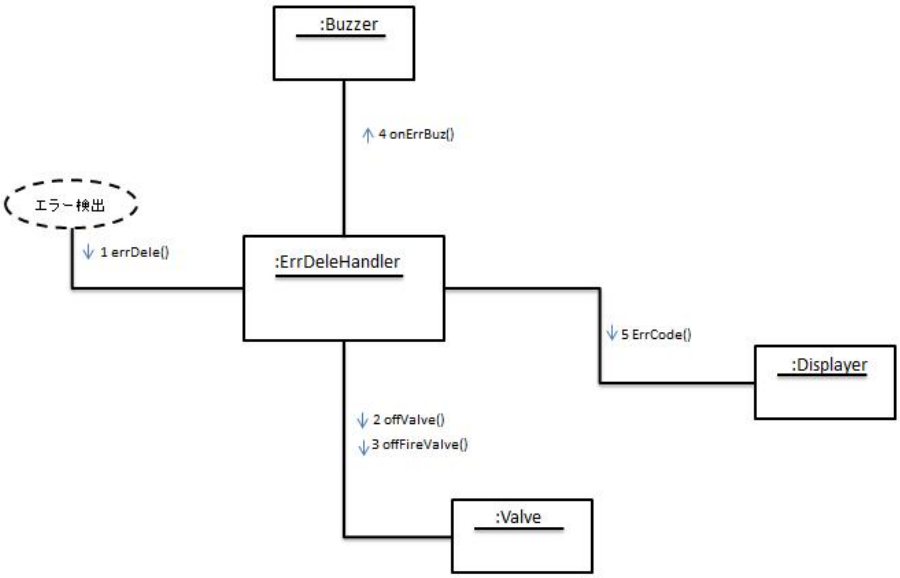


図 A.15: エラー処理ユースケーススライスのコミュニケーション図