JavaソースコードのCDI(Code Inspection)の開発
～プロセス改善～

M2007MM017 水野貴文
指導教員：野呂昌満

1　はじめに

本研究は，文部科学省の事業である先導的 IT スペシャリスト育成推進プログラムの OJL(On the Job Learning) としておこなう．本プログラムでは，各技術の本質となる技術をベース技術，ベース技術を選択・合成し新たな課題に即応した技術を生み出し応用できる技術をメタ技術と呼ぶ．
現在のソフトウェア開発プロジェクトではオブジェクト指向やアスペクト指向技術などのベース技術を適用して開発をおこなっている．ベース技術を適用してもソフトウェア開発では問題が発生する．しかし，問題を解決した技術を管理する技術が適用されていない．
ソフトウェアアーキテクチャや開発プロセスなどソフトウェア開発プロジェクトの再利用可能な部品を管理してプロジェクトを遂行していく技術にプロダクトラインソフトウェアエンジニアリング (以下，PLSE)[1] がある．本研究室は，PLSE をメタ技術のひとつとして考える．
PLSE とは，共通の特徴を持つソフトウェアファミリに対し，核資産の開発をおこない，核資産をもとに製品であるソフトウェアの開発をおこなう．PLSE はアーキテクチャに基づいた開発である．また，部品を再利用するための工程を定め，製品の標準化をすることにより，製品毎の再利用可能な部品を再利用する系統的な開発をおこなうことができる．

ソフトウェア開発における問題点として，再利用可能なモジュールを共通モジュールとして開発する開発プロセスがないのである．ソフトウェアの要求は製品毎に異なる．しかし，製品によっては違う製品に対しても同じ要求が紛れていることがある．例えば，製品の価格を抑えるために，ある製品の機能を一部削除し，新たな製品とする場合がある．この場合は，同じ機能は共通もジュールとして再利用が可能である．

本研究の目的は，開発するソフトウェアのドメインの特徴から開発プロセスを定義し，共通モジュールを再利用できる開発プロセスに改善することである．本研究では，事例研究として対象とするドメインをコードインスペクション (以下，CDI) ツールとし，CDI ツールの特徴から開発プロセスを構築し，プロセス改善のプロセスを適用し，開発プロセスの改善をおこなう．
本研究は以下の手順でおこなう．

1. CDI ツールの特徴から開発プロセスを定義
2. CDI 機能を実現
3. 開発プロセスの改善
4. 考察

2　メタ技術・ベース技術

2.1　PLSE

PLSE とはソフトウェアの開発を系統的におこなうことを支援する技術のひとつである．その方法として，核資産を開発し，アプリケーションの開発をおこなう．核資産とは，要求分析の結果，再利用可能な部品およびアーキテクチャなどの開発に必要な全ての総称である．PLSE では**図 1** に示す活動を中心に開発をおこなう．

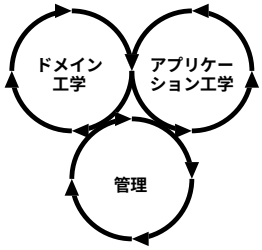

図 1　PLSE における主要な活動

- ドメイン工学
  - ユーザ要求の分析
  - 核資産の開発
- アプリケーション工学
  - ドメイン工学で開発した核資産をもとにアプリケーションを開発
- 管理
  - ドメイン工学やアプリケーション工学で開発，使用した核資産の保守
  - 開発体制の整備

2.2　プロセス改善プロセス

開発プロセスに問題が発生した場合に，プロセスを改善するためのプロセスとして以下の手順がある．

1. プロセスの問題を把握
2. 問題の原因を分析
3. 改善案の提案，実施

2.3　デザインパターン

デザインパターンはオブジェクト指向プログラミングを対象にし，様々なプログラムの設計で利用できるパターンである．オブジェクト指向プログラミングでよく使用される特徴的な構造や機能を抽出したものである．
GoF のデザインパターン [2] は以下のカテゴリに分けられ，全部で 23 種類ある．

- 生成に関するパターン

- 構造に関するパターン
- 振る舞いに関するパターン

3 CDI ツール

3.1 CDI の概要

CDI とはソフトウェアを開発するさいに，品質向上を目的として実施するソースコードの静的解析である．ソースコードを静的に解析することによって，ソフトウェアを実行する前にソースコードの欠陥を発見することができる．従来の CDI は開発者ではない第三者が数人集まり，ソースコードのレビューをおこない，ソースコードの欠陥を見つけることである．しかし，プログラムの複雑化，膨大化により，人がプログラムを検査するには時間とコストがかかってしまう．さらに，レビューでの見落としによってソースコードに欠陥が潜んでしまう可能性がある．CDI ツールは対象となるソースコードを読込み，ソースコードに潜む欠陥を発見する．
本研究での CDI ツールは**図 2** のように Java のソースコードから抽象構文木，制御フローグラフ (以下，CFG)，データフローグラフ (以下，DFG) を作成する．作成した情報をもとに各ノードを走査することでソースコードを静的に解析し，欠陥の発見をおこなう．開発するツールは以下にあげる CDI ツールである．

- 検査をおこなうソースコード
  Java1.5 の言語仕様にもとづくソースコード
- 開発環境
  Eclipse[3]
- 開発言語
  Java1.5

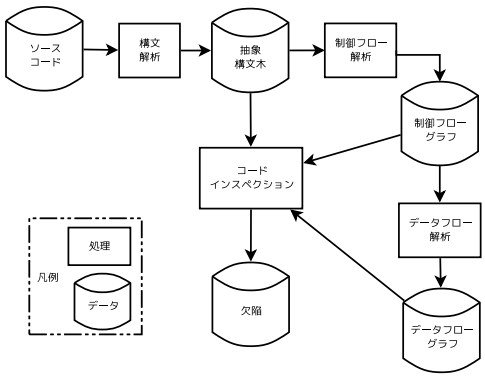

図 2 CDI ツールの概要

3.2 本 CDI ツールの特徴

本 CDI ツールには以下の特徴がある．

- PLSE に基づいた開発
  - ドメイン工学
    本ツールでは，要求や仕様，アーキテクチャ，核資産，テストケース，スケジュール，プロセスがあげられる．また，核資産にはアーキテクチャやフレームワーク，コンポーネントがあげられる．
  - アプリケーション工学
    本ツールでは，CDI 機能があげられる．
  - 管理
    本ツールでは，ドメイン工学やアプリケーション工学で開発，使用した核資産やテストケース，スケジュール，プロセスの保守，開発体制の整備保守があげられる．
- 検査項目毎の要求が独立
  検査項目に対する要求が他の検査項目に依存しない．例えば，NullPointerException の発生を発見する項目と，入れ子構造が一定以上の深さになっているものを発見する項目では，項目間での要求が依存しないのである．
- アーキテクチャが明確
  本ツールで扱うデータ構造は抽象構文木とフローグラフである．抽象構文木は Java の言語仕様 [4] をもとに定義しているので，データ構造の変更頻度は少ない．また，データに対する処理はフロー解析や発見する誤りごとに存在するので，デザインパターンの Visitor パターンと Interpreter パターン，Composite パターンを適用している．また，データフローも構造に対する変更頻度は少ない．しかし，データに対する処理は抽象構文木と同じく発見する誤りごとに存在するので，フローグラフにも Visitor パターンと Interpriter パターン，Composite パターンを適用している．CDI 機能を実現するには，本ツールのアーキテクチャで構築した Visitor のサブクラスに記述することで可能になる．

4 開発プロセス

CDI ツールの特徴から設定した本 CDI ツールにおける CDI 機能の初期の開発プロセスと改善した開発プロセスの説明をおこなう．初期の開発プロセスと改善した開発プロセスは PLSE のドメイン工学の活動である．

4.1 初期の開発プロセス

CDI ツールの特徴から CDI 機能の開発における初期の開発プロセスは**図 3** に示す以下の手順とした．

1. 要求仕様書の分析
2. 走査対象の選択
3. 設計実現
4. テスト

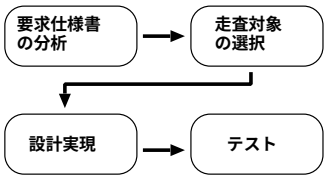

図 3 初期の開発プロセス

4.1.1 要求仕様書の分析

どのような誤りを発見したいのかを把握するために，要求仕様書の分析をおこない，外部仕様書を作成する．外

部仕様書を協力企業に提出し，協力企業からの要求が正しく分析できているか確認をおこなう．また，外部仕様書からテストケースの作成をおこなう．外部仕様書では発見すべき誤りを Java 言語の構文 [4] などを用いて明確に記述し，検査での制限事項などを明記する．テストケースでは実際に誤りを含んだプログラムと誤りを含んでいないソースコードを作成する．

4.1.2 走査対象の選択

4.1.1 節で明確にした発見すべき誤りの検出に必要な走査対象を選択する．走査対象は抽象構文木やフローグラフである．発見すべき誤りがプログラムの構文の各要素の場合は抽象構文木を，制御の流れの場合は CFG を，データの場合は DFG[5] を選択し，Visitor で走査する．

4.1.3 設計実現

3.2 節で説明した本ツールの特徴から，CDI 機能の設計実現は容易であると考えた．また，本ツールのアーキテクチャと核資産はドメイン工学で開発されているので，アーキテクチャと核資産をもとに CDI 機能を実現すれば良い．なので設計プロセスを短縮できないかを考えた．そこで開発プロセスに eXtreme Programming(以下，XP) を導入した．XP はアジャイルソフトウェア開発手法の一つである．XP は初期段階の設計よりもコーディングを重視しており，各工程を順序立てて順番に積み上げてはいかない．常にフィードバックをおこなって修正・再設計していくことを重視している手法である．また連携企業の開発手法も XP を導入しており，実践を想定した開発をおこなう．以上の事から，設計と実現を一つにまとめたプロセスを適用した．
4.1.2 節で選択した走査対象をもとに，欠陥を発見するアルゴリズムを構築する．実現は Visitor のサブクラスでおこなう．

4.1.4 テスト

CDI 機能に対してテストをおこなうのは開発者ではなく，テスト担当者がおこなう．テストは 4.1.1 節の外部仕様書をもとに作成したテストケースを使用するブラックボックステストである．

4.2 プロセス改善

初期の開発プロセスでは共通モジュールを部品として再利用するプロセスがない．共通部品を抽出し，再利用できるように改善をおこなう．

4.2.1 プロセスの問題を把握

原因を把握するために，どのプロセスに問題があるかを分析した．初期の開発プロセスは要求が依存していないと判断し，開発プロセスを定義した．その結果，プロセスの短縮をはかった設計実現プロセスに問題があることが判明した．

4.2.2 問題の原因を分析

問題が起きた原因として，以下の点をあげる．

設計実現を一つにまとめたプロセス

初期の開発プロセスでは，設計と実現を一つにまとめて実現するので，プログラムの設計に問題が潜んでいても，気付くのは難しい．
各検査項目の要求が独立しているという判断に誤りがあり，一部の検査項目に依存関係がある事が判明した．依存関係がある項目の共通箇所を部品化し，項目の実現をおこなう前に部品として抽出をおこなえば，共通部品として利用できるのではないかと考えた．
以上のことから，本プロセスを細分化し共通部品の利用しやすいプロセスへ改善をおこなう．

4.3 改善したプロセス

初期の開発プロセスでは設計実現を一つにまとめたプロセスにおいて，手戻りが発生した．手戻りを解決するために，設計実現のプロセスを細分化し，設計で潜んだ問題を解決できるように変更した．初期の開発プロセスで発生したプロセスの手戻りを解決した開発プロセスとして，**図 4** に示す以下の手順をあげる．

1. 要求仕様書の分析
2. 走査対象の選択
3. 設計実現
   (a) 内部仕様書の作成
   (b) 内部仕様書レビュー
   (c) 共通部品の抽出
   (d) 実現
   (e) ソースコードレビュー
4. テスト

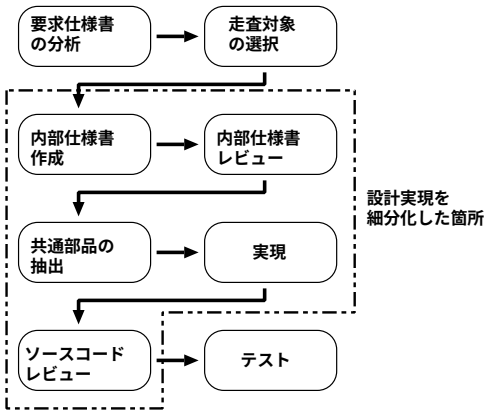

図 4 改善した開発プロセス

5 考察

アーキテクチャではなくプロセスの改善をおこなったのは 5.3 節で述べるが，開発者の能力を過大評価していたからである．開発者に能力がある事を前提に XP を開発手法として導入をしたので，プロセスの改善をおこなった．一般的な開発手法である，ドキュメント重視の開発にプロセスを改善した．

5.1 内部仕様書の作成とレビュー

4.1.1 節で作成した外部仕様書をもとに，発見したい誤りを走査するアルゴリズムを内部仕様書として作成する．作成した内部仕様書は第三者がレビューをおこない，アルゴリズムが冗長ではないか，他にもっと単純なアルゴリズムはないか検査する．抽象構文木，フローグラフを選択する場合においても，選択する対象に誤りがないか検査する．内部仕様書にアルゴリズムを記述することで開発者がアルゴリズムを整理することができる．実現をしたい機能に必要なモジュールの有無が内部仕様書を作成する段階で分かるようになった．

5.2 共通部品の抽出

内部仕様書を作成する事により，項目毎のアルゴリズムを明記するので，どのモジュールが共通の部品となるのか判断がおこないやすくなる．また，抽出した部品は，走査するアルゴリズムを構築する時に利用可能なモジュールとして判断がおこないやすくなり，共通部品として使用が用意になる．また抽出した共通部品はコンポーネントとなるので，CDI ツールの核資産となる．また，抽出した部品が核資産となることによって，再利用が可能になる．共通部品の抽出は本 CDI ツールの開発における特徴であると考えることができる．

5.3 過大評価

開発プロセスを定義する時に，開発者の開発能力を過大評価していた．開発者が考えた誤り検出アルゴリズムが冗長であった．中には子要素を取得するメソッドが抽象構文木にあるのにもかかわらず，Visitor で子要素を取得していたケースがあった．抽象構文木やフローグラフを理解すれば，より単純なアルゴリズムで CDI 機能を実現できることが分かった．

5.4 プロセスの評価

プロセス改善の指標の一つとして CMMI(Capability Maturity Model Integration)[6] がある．CMMI は開発をおこなうためのプロセス改善成熟度モデルである．アメリカ国防省が米国のカーネギーメロン大学に設置したソフトウェア工学研究所 (Software Engineering Institute) が考案した．CMMI における評価の一つに段階表現がある．CMMI は複数の CMM が統合されたものであるが，段階表現は複数ある CMM のソフトウェア CMM を継承している．

段階表現は 5 段階のレベルで評価をおこなう．5 段階のレベルによってプロセスの改善度合いをあらわす．また，レベルが高ければ高いほど開発プロセスは成熟していることを表す．

改善前のプロセス　改善をおこなう前のプロセスはレベル 2 であると考えられる．改善前の開発プロセスは，プロジェクトの計画を立て，プロセスを定義し，定義したプロセスに乗っ取り開発をおこなうものであった．また段階表現のレベル 2 は，管理された段階であり，プロセスは方針に従って計画・実施される．要件の管理，プロジェクトの計画，監視・制御測定などの基本的なプロジェクト管理が確実におこなわれる．というのもである．よって，改善前の開発プロセスは基本的なプロジェクト管理がおこなわれていると判断することができ，段階表現のレベル 2 であると考えることができる．

改善後のプロセス　改善をおこなった後のプロセスは，レベル 3 であると考えられる．改善後のプロセスは，改善前のプロセスの問題点を把握し，分析をおこない，改善したものである．また段階表現のレベル 3 は定義された段階であり，プロセスの特性が十分に明確化され，理解されている．標準・手順・ツール及び手法の中で記述されている．標準のプロセスがあり，継続して改善されている．この標準プロセスは，各プロジェクト向けにテーラリングして利用する．というものである．よって，改善前のプロセスに対し，テーラリングをおこなうことにより，本プロジェクトに有効な開発プロセスへ改善をおこなうことができ，段階表現のレベル 3 に改善できたと考えることができる．

6 おわりに

3.2 節で説明した本 CDI ツールの特徴から開発工数が短縮できると考え，短縮した開発プロセスを適用した．しかし，実際には短縮したプロセスに問題が発生し，開発工数の短縮どころか，依存する項目の共通モジュールの部品化，利用の妨げになっていた．プロセスを改善するプロセスを適用して，プロセスの改善をおこなった．その結果，改善したプロセスでは共通モジュールの判断が容易におこなうことができ，部品の抽出や利用が容易になった．本ツールの開発プロセスを改善する時に，メタ技術を適用することでプロセスの改善ができ，プロセスを改善するというメタ技術を習得した．

今後の課題は，本ツールの CDI 機能の実現においては適用が可能であったが，本ツールのユーザインターフェースや他のドメインでも適用が可能であるか検証する必要がある．

参考文献

[1] L. M. Northrop,“ SEI’s Software Product Line Tenets, ”*IEEE Software*, vol. 19, no. 4, 2002, pp. 32-40.

[2] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns Elements Of Reuseable Object-Oriented Software*, Addison Wesley, 1995.

[3] Eclipse, *Eclipse*, http://www.eclipse.org/.

[4] J. Gosling, B. Joy, G. L. Steele, and G. Bracha, *The Java Language Specification, Third Edition*, Addison Wesley, 2005.

[5] 中田育男, コンパイラの構成と最適化, 株式会社朝倉書店, 1999.

[6] Software Engineering Institute, *CMMI Website*, http://www.sei.cmu.edu/cmmi/.