3L-7

PBLによるソフトウェアのフレームワーク設計力の強化

平野智沙†，村山満†，土屋陽介†，中鉢欣秀†，宮浦智範‡
†産業技術大学院大学，‡日立インフォメーションアカデミー

1. はじめに

開発の現場では，即戦力となる人材を育成するため，フレームワークの利用手順を学習させることが多い．フレームワークにより生産性は飛躍的に向上した反面，設計や実装の多くがブラックボックス化されることにより，長期的な技術力の低下を招いている．

そこで我々は，Web アプリケーションのフレームワークを開発する過程を通して，アプリケーションの開発者に必要な知識やスキルを強化するための教育カリキュラムを考案した．フレームワークの設計を理解することで，抽象化や結合度や凝集度の操作法といったオブジェクト指向設計の理解度向上を狙う．また，一連の開発手順を修得するため，PBL(Project Based Learning)方式[1]を採用した．本論文ではこのカリキュラムを実施した結果に基づきその有用性を考察する．

2. カリキュラム

我々は，アプリケーションの設計改善を通してフレームワークを導出することとした．なお，今回実装したアプリケーションは，会員登録，ログイン・ログアウト，商品一覧・詳細，注文機能を持つ，JavaEE，RDB を用いたインターネット・ショッピングサイトとする．

ここでいう設計改善とは，既存の設計における問題点を洗い出し，その原因を分析し，対策を検討する一連の問題分析プロセスを指す．このプロセスを実現する，一連の手順を図 1 に示す．

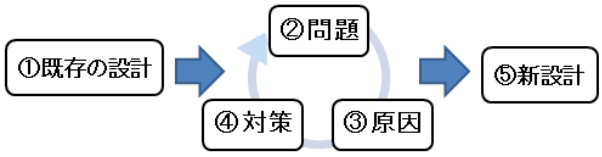

図1　設計改善の手順

まずはMVCモデルに従ってアプリケーションを実装する（図1-①）．この設計をインプットとして問題分析を行い（図1-②③④），対策として新たな設計を導出する（図1-⑤）．実際に導出される問題点は2-1で述べる通り多岐にわたるため，このプロセスを繰り返し行う．

2-1.設計改善の観点

表 1 に示すA，B，C 3つの観点から，設計改善を行う．観点Aでは，複数のクラス間やメソッド間で重複する実装をまとめ，一元化する．コードの重複をなくすことにより，プログラムの変更容易性を向上させる，観点Bでは，各クラスの役割を精査し，“1つのクラスは1つの役割を持つ”を原則としてクラスを分割する．これにより，個々のクラスの変更理由を限定することができる．一方，クラス間の関係が複雑になる恐れがあるため，結合度を減らし変更の局所化を図った．観点Cでは，プログラム中にハードコーディングされる物理名を排除する．物理名とは，ファイル名やパス，エラーメッセージなどを指す．これらはシステムの運用と並行して変更されることが多いため，論理名と物理名をひもづけて一元管理し，プログラム中では論理名を指定することとする．

表1　改善例

観点	適用箇所	問題点
A	JSP	記述内容(ヘッダ, フッタ)の重複
		データ表示の重複
	Servlet	認証チェックの重複
		入力内容の取得処理の重複
		画面遷移処理の重複
	DAO	DB 接続処理が重複
	エラー処理	例外処理が重複
B	Servlet	ビジネスロジック処理と画面遷移処理が混在
	DAO	画面の入出力項目と DB の入出力項目が1つのクラスで管理されている
C	Servlet	入力内容の検査の実装が重複
		画面物理アドレスで指定
		メッセージのハードコーディング

2-2.設計改善の例

ここでは，前述の観点Aによる設計改善の例を図2に示す．

改善前の設計（図2左）では，利用者の入力値の取得や妥当性のチェック，データベースアク

Enhancing framework design capability by PBL
Chisa HIRANO†, Mitsuru MURAYMA†, Yosuke TSUCHIYA†, Yoshihide CHUBACHI† and Tomonori MIYAURA‡
† Advanced Institute of Industrial Technology

セスを中心とするビジネスロジックの実行，処理結果に応じた画面の表示，そしてエラーが発生した場合の例外処理といった一連の処理を，機能毎に設けた Servlet にて実装していた．しかし異なる機能間でもアルゴリズムは似通っており，実装時にはプログラムをコピー&ペーストし開発していた．

一方，改善後の設計（図 2 右）では，Servlet 間で共通の処理および処理フローを，1 つの Servlet にてまとめて実装し，機能固有の処理は別のクラスに実装することとした．この結果，アプリケーションの為にユーザーが記述したコード(User Own Coding : UOC)を局所化することができた．

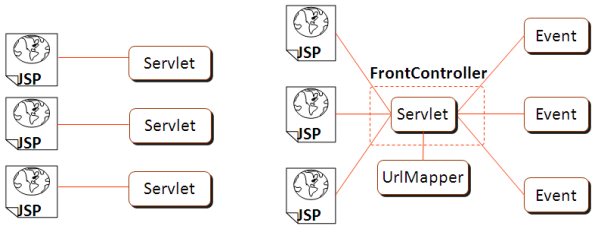

図2　設計改善の例

2-3. 設計改善の評価

改善結果の妥当性を評価するため，性能の評価とメトリクス解析を行った．

システムの性能を測定した結果，スループットは約 1.4 倍の向上，CPU 使用率は 4 分の 1 に抑えることができた．これは，主な改善点として実施した，フロントコントローラパターンの導入によるサーブレットの一元化や，コネクションプーリングの導入によって，リソースの有効活用と繰り返し処理の排除が可能になったことによるものである．

表2　設計改善での違い

観点	旧設計	新設計
コード量	1k	1.5k
凝集度	0.181	0.308
複雑度	2.705	1.528
結合度	0.420	0.419

※Metrics Plugin for Eclipse を使用して計測

観点Bに基づき役割に応じてクラスを分割したことにより，各クラスのアルゴリズムはシンプルになり，複雑度を下げることができた．事実，UOC部のプログラムは一定のルールに従って実装すれば良くなり，当初フレームワーク無しで 4 週間かかった実装が設計改善後は 2 週間で実装でき，実装難易度が下がったことを体感することができた．しかし今回の取り組みでは，UOC部のプログラム量を減らすには至らなかった．より実際の開発で使えるものに近付けさせるためには，例えば画面仕様やデータベース・スキーマからプログラムを自動生成する仕掛けを設ける等の措置を講じさせる必要がある．

3.考察

本カリキュラムを受講することで，受講者はフレームワークの設計に関するいくつかの知見を得た．例えば，フレームワークを使用しないコードの実装から，コードの重複の排除や再利用性・保守性に重点を置いた設計改善を行い，効率を追及した部分が，正にフレームワークになっていたことに気がついた．

また，カリキュラムの最後で よく使われるフレームワークの 1 つである Struts と作成したフレームワークを比較したところ，似通った構造になっていた．今後受講者が類似のフレームワークを使用する際，背後で行われる処理を意識することで より無駄のないアプリケーション設計ができるのではないかと期待される．

一方で，今回のメンバーはアプリケーション開発の経験が乏しく，前提となる知識やスキルが不足していた．このため，各プロセスの実施に必要な知識やスキルの不足を，適宜教育を挟むことで解決した．具体的には，プログラミングスキル，UML による設計モデリングスキル，性能テストなどのテスティングスキルを補う教育を実施した．

しかし，理解度まで確認する方法がなく，理解できたとの思い込みで次のステップに進むことがあり，予定遅延を招くことがあった．したがって，繰り返しスキルチェックを行い，実施する教育カリキュラムを体系化する必要がある．

4.まとめ

今回の PBL を通して，効率，保守性を考慮した改善の集大成がフレームワークとなっていることを受講者が実感でき，このカリキュラムが設計力強化のプログラムとして有効であることがわかった．今後このカリキュラムを円滑に実施するには，各プロセスで必要なスキルを明確にし，作業に入る前に行うスキル補完の教材が必要であると考える．

参考文献

[1] 中鉢 欣秀: "ソフトウェア・アーキテクト育成のための PBL 型教育" ,日本 e-Learning 学会研究会報告,(2006 年 9 月)