東海大学紀要情報通信学部
vol.6,No1,2013,pp.1-7

論文

フレームワークを用いたWebアプリケーション開発における変更容易性の評価

高橋明日香[*1]，小林 洋[*2]

Evaluation of Web Application Modifiability in Software Development Using Framework

by

Asuka TAKAHASHI[*1] and Hiromi KOBAYASHI[*2]

(received on March 14, 2013 & accepted on July 19, 2013)

Abstract

Nowadays, applications are not totally constructed from new codes, but using the modification of existing applications in software development. Therefore, the modifiability of application becomes a crucial metrics. Frameworks are widely used in Web application developments. We evaluate the modifiability of application program when using frameworks. This evaluation is considered as a change impact analysis. We evaluate Hibernate and Ruby on Rails that are standardized using JSP. We evaluate the modifiability of these using a qualitative analysis and quantitative analyses based on LOC, the number of folders, and that of based on human senses.

Keywords: *framework, change impact analysis, modifiability, Rails, Hibernate*
キーワード:フレームワーク, 変更波及解析, 変更容易性, Rails, Hibernate

1. 概要

最近の業務系のWebアプリケーション開発では，開発効率や信頼性を高めるのを目的として，フレームワークを使用した開発が増えて来ている[1)]．しかしながら，その評価について示された文献はほとんど見当たらない．個別にフレームワークについての書籍，例えば文献1-4)等を見れば，メリットや使い方について書かれているが，横断的な評価はあまり行われていないように見受けられる．そこで本研究では，この問題を扱うことにした．フレームワークを用いた場合のソフトウェア開発の生産性評価には，様々な問題点がある．プログラム開発の生産性は，元々個人によって著しく異なる上に，開発対象（ドメイン）によってプログラムの難易度が異なり，また，プログラムでの記述には自由度があるため，定量的指標として良く使われているLOC（Lines of Code）での単純な量的評価や開発時間（工数）での評価には，問題があることはしばしば指摘されている．更には，開発規模（スケーラビリティ）による難易度の違いという問題もある．以上のような問題に加え，フレームワークに対する慣れ，つまり学習時間をどのように扱うのかという問題もある．しかしながら，フレームワークを用いた開発が増えている現状を鑑みると，評価が困難であっても，限定された条件の下にせよ，今後は評価を試みる必要があると考えられる．

ところで，アプリケーション開発では，一旦開発した後，良く使われるものは通常，修正・追加の開発が入る．また，最近の開発では，全てを新規に作成するというよりも，既存のものを改修するケースが増えて来ている．そこで，本研究ではこの点に着目して，修正・追加の際の変更箇所についての評価を行うことにした．但し，今回の変更の程度としては，例えば文献14)のようなプログラムの構造に関わるようなものではなく，プログラムの構造の変更を伴わない，比較的マイナーなアプリケーションの変更に留めることにした．フレームワークは，プログラムの構造の変更を伴わないような変更であれば，比較的容易に変更が出来るように作成されていると考えられる．本研究では，Webアプリケーション開発での基本と考えられるJSP（Java Server Pages)を用いた場合と，フレームワークHibernate[2)] 及びRuby on Rails[3-4)]を各々用いた場合で，まず，図書管理システムモデル[5)]の開発を行い，その後，修正・追加を行い，その際の変更の容易性についての評価，つまり，一種の変更波及解析(Change Impact Analysis)[6-12)]を行った．このように修正・追加分の評価を行うようにしたのは，あえてフレームワークの学習時間を排除したかったからであり，開発現場などにおいては，その方が妥当と思えたからである．評価としては，先ず，修正・追加の際の特徴的な事を比較する定性的評価を行った．次に，伝統的な手法で未だに良く使われているLOCによる比較を定量的評価として行った．この際，最近のフレームワークによる開発においては，プログラミング言語による記述と共に，XMLでの設定ファイルを用いる事が普通であるので，XMLでの記述量についてもこの評価に加えること

*1 工学研究科情報理工学専攻
Graduate School of Engineering
*2 情報通信学部情報メディア学科 教授
School of Information and Telecommunication Engineering，Department of Information Media Technology，Professor

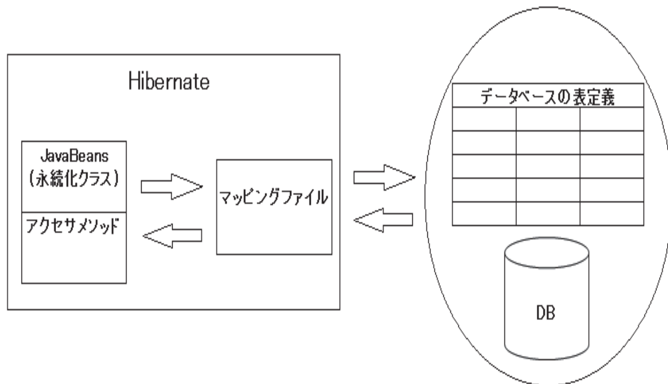

Fig.1　Hibernateの構成

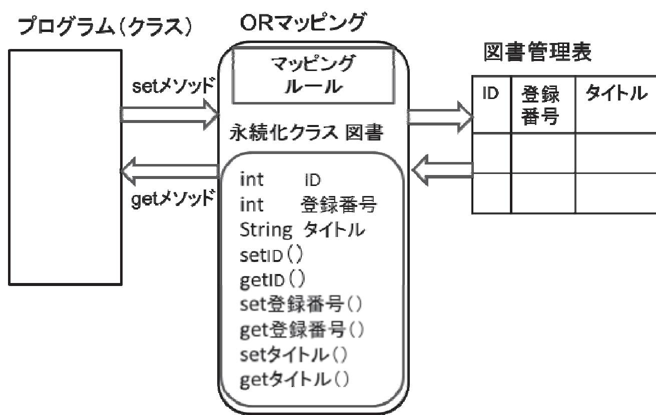

Fig.2　ORマッピング概念図

にした．また，フレームワークを用いた場合には，デフォルトでフォルダがいくつか作られることから，フォルダ数についても評価することにした．更に，修正・追加には，該当する箇所の特定がまず必要であることから，人間系を含めた評価も行った方が妥当であると考えられるため[10]，修正箇所の発見の容易性，修正操作の容易性，修正記述の容易性という人間系から見た評価も加えることにした．

2. フレームワークについて

2.1 Hibernateについて

Hibernate は，Java 言語がベースとなるフレームワークである．Hibernate の構成を Fig.1 に示す．Hibernateでは，MVCモデルアーキテクチャをサポートしており，データベース処理を含む業務処理部分を担当する Model，データ表示を行う View，及びこれらを制御する Controller の部分に分割される．Hibernateでは，データベースにアクセスするための設定ファイルの作成に加え，データベースの表の列に対応するアクセサメソッドを持つ永続化クラスと，Fig.2 に示すようなオブジェクトの各属性値とデータベースのテーブルの各属性値との対応付けを行うためのマッピングファイルの定義を行うことで，プログラムとテーブルの間のデータ表現の違いが吸収される．またHibernateToolsの使用により，リレーショナル・データベースのテーブルと属性値の作成から，マッピングファイルと永続化クラスを自動的に生成することができるため，コードの記述を削減することができる．

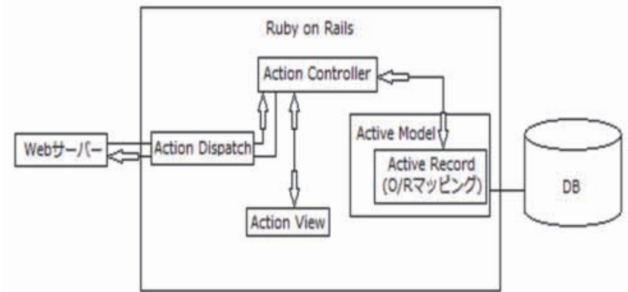

Fig.3　Ruby on Rails の構成

2.2 Ruby on Rails

Ruby on Rails（以下，Rails）はスクリプト言語Ruby を用いるフレームワークで，Web システムのアジャイルな開発にしばしば用いられている．Railsの構成をFig.3に示す．RailsもHibernate同様にMVCモデルアーキテクチャをサポートしており，Model，View，及びControllerの自動生成機能により容易にプロトタイプの構築を行うことが可能である．

Railsでは，ActiveRecordというORマッピングのライブラリにより，データベースを使用する Web アプリケーション開発において，記述するソースコード量を削減することができる．

3.　図書管理システムの開発と修正・追加

最初に，JSP，Hibernate，及び Rails を各々用いて，図書管理システムモデルの Web アプリケーション開発を行った．開発は全て，eclipse3.7 上で行った．なお，JSP での開発では，Hibernate，Rails との比較が可能なように，MVCモデルを採用した．最初に作成したアプリケーションでは，図書の一覧表示，図書の登録，図書の削除の機能からなり，図書の属性は，登録番号，分類番号，ISBN，タイトル，著者，出版社，出版日，分類の8つからなる．

次に，このアプリケーションに属性の追加，更新機能の追加，図書合計数の追加，ログイン機能の追加を以下のように行い，三者での変更箇所の比較を行った．なお，各画面については，三者とも同等のものになるように今回は，あえて GUI ツールを使わずに作成した．

（作成した画面の内，6画面については付録に掲載）

Table 1 修正・追加における特徴

(a)属性の追加	JSP	SQLとJava文の追加
	Hibernate	リバース・エンジニアリングファイル更新とJava文の生成
	Rails	属性追加のコマンド入力
(b)更新機能	JSP	自動生成なし
	Hibernate	自動生成なし Controller内にUpdateの処理を作ると行数が増加する.
	Rails	自動生成あり Controller内にUpdateの処理ができる.
(c)図書合計	JSP	標準的なSQL.
	Hibernate	標準的なSQL.
	Rails	SQLと記述が異なる.
(d)ログイン機能	JSP	Controllerのページ定義が不要.
	Hibernate	Controllerのページ定義が不要.
	Rails	Controllerのページ定義が必要.

（a）属性追加：図書の属性として価格の追加を行った．図書一覧，図書登録，及び図書削除の画面に価格の属性を追加した．なお，図書削除の画面では,一旦，全属性を表示しデータの確認後削除を行うようにした．

（b）更新機能追加：最初に作成したアプリケーションでは，あえて作成していなかった更新機能を追加した．更新機能は，メニュー画面から図書更新を選択すると，一旦，全ての図書データを表示し，その後更新図書の選択を行えるようにした．そのために，View では,メニュー画面に図書更新のハイパーリンクを追加し，更新図書選択画面と更新画面を作成した．Model では,図書選択画面で選択した図書の全属性を表示後，該当箇所の属性値を入力し更新ボタンを押すと更新が行えるようにした．Controller では，View と Model の登録番号の受け渡しと取得データの受け渡しを行うようにした．

（c）図書合計数カウント機能追加：SQL 文の Count 命令を使用し，登録した図書の合計数を求める機能を追加し，合計数を図書一覧画面に表示するようにした．

（d）ログイン機能追加：ログイン機能の追加を行った．ログイン画面では，ID とパスワードを入力する事により，ログインが行われメニュー画面への遷移が行われ，ID・パスワードが正しくない場合は，ログインエラーページへ遷移するようにした．今回，ID とパスワードに関しては，データベースに関わらない機能についての追加の影響を見るために，データベースには登録せず，Model 内に登録を行った．

4. 評価

4.1 定性的評価

前節での修正・追加について，特徴的な事を比較すると以下のようになった．これらをまとめたものを，Table 1 に示す．

（a）属性追加：Hibernate では，データベースのテーブルへの属性追加のために，SQL 文の追加と共に，テーブルとオブジェクトの属性値の対応付けのために，リバースエンジニアリングファイルの更新を行った．なお，永続化クラスとマッピングファイルの更新行うための Java コードと XML については自動的に作成された．Rails では，コマンドを入力するだけでテーブルの更新が自動的に行われるため，ソースコードの追加は不要であった．

（b）更新機能追加：Rails では，コマンド入力を行う事で，CRUD 機能を自動的に作成できた．但しこの場合，Model ではなく Controller の部分に Update の処理が作成された．

（c）図書合計数カウント機能追加：Rails でのみ，データベース操作の記述が標準的な SQL 文とは異なったものであった．例えば，total という属性名でテーブルの行数をカウントするとき，次のような記述になる．

（テーブル名）select（’属性値，count() as total’）

（d）ログイン機能追加：Rails では，Controller でのページの定義を行う必要があった．また，Rails ではハイパーリンクを用いたページの遷移に際して，URL に応じて Controller 内の表示・登録・削除・更新に関するメソッドを指定するための Routing の記述が必要となった．

以上から解ることは，フレームワークを用いた場合には，コードの作成・変更がコマンドにより自動的に行われる．しかしながら，MVC の分離が必ずしもうまく行われるとは限らない．また Controller での記述のように，仕様からは不要と思えても，記述が必須となる項目もある．データベースの操作については，SQL 標準の記述と異なる場合がある．

4.2 定量的評価

4.2.1 LOC

プログラムは，記述に自由度があるため，同じ仕様からであっても，プログラム構造の違いにより LOC は異なって来る．また，LOC では，コードが複製されたものであっても，これについては全く考慮されない．しかしながら，LOC は，他にシンプルでより妥当な評価尺度が見当たらないことから，未だに最も良く使われている評価尺度である．今回の場合の変更箇所を LOC で比較した結果を，以下に，それらをまとめたものを Fig.4 に示す．なお，最近のフレームワークでは，プログラムコード以外に，XML ファイルでの記述を設定として用いていることが多いため，プログラム(PG)と,XML の追加･修正の行数を Model，View，Controller 別に示している．

	PG	XML
Model	9	0
View	3	7
Controller	1	0

①JSP

	PG	XML
Model	3	0
View	3	7
Controller	1	0

②Hibernate

	PG	XML
Model	[3]	0
View	3	7
Controller	0	0

③Rails

(a)属性の追加

	PG	XML
Model	57	0
View	33	67
Controller	21	0

①JSP

	PG	XML
Model	16	0
View	33	67
Controller	22	0

②Hibernate

	PG	XML
Model	0	0
View	26	64
Controller	11	0

③Rails

(b)更新機能の追加

	PG	XML
Model	33	0
View	2	4
Controller	2	0

①JSP

	PG	XML
Model	6	0
View	2	4
Controller	2	0

②Hibernate

	PG	XML
Model	3	0
View	3	4
Controller	3	0

③Rails

(c)図書合計値の追加

	PG	XML
Model	11	0
View	8	28
Controller	10	0

①JSP

	PG	XML
Model	11	0
View	8	28
Controller	10	0

②Hibernate

	PG	XML
Model	11	0
View	8	28
Controller	15	0

③Rails

(d)ログイン画面の追加

Fig.4 機能追加時のコード追加行数（LOC）

（a）属性追加：Hibernate の Model では，リバースエンジニアリングファイルの更新と Hibernate コードの作成を行うことで，永続化クラスとマッピングファイルの更新行う Java 文が自動的に作成される．Rails の Model では,コマンド操作により作成を行う．コマンドとプログラムのコードを同等に扱って良いものかという問題があるので，コマンド３行分を[]付で記しておいた．

（b）更新機能追加：Hibernate の Model では，登録番号の属性値の受け渡しと更新のためコードの追加を行ったが,Rails ではコードの追加は必要なかった．Rails の View では，JSP や Hibernate と異なり，リストボックス使用時，繰り返し文が不要であった．

（c）図書合計数カウント機能追加：Hibernate の Model では，session の取得でデータベースにアクセスし,SQL の実行で図書合計数の計算を行った.Rails の Model では，3 行の SQL 文の追加のみで済んだ．Rails の Controller では，図書合計数の計算結果の受け渡しに 3 行の追加を行った．

（d)ログイン機能の追加:Rails の Controller では,ログインが成功した際のメニュー画面へのアクセスと不成功の際のエラー画面への遷移を制御するためのコード，ログイン画面とエラー画面の定義を行うためのコード，及びページ遷移に必要なルーティングに処理の記述が必要となった．

以上をまとめると，次のようなことが言える．

①Model に関わる部分の機能の修正･追加の場合には，フレームワークを用いた方が，コードの自動生成機能により修正・追加の行数が少なくて済む．特に，更新機能の追加や図書合計数の追加の際の Rails において顕著である．

②View に関わる部分の修正・追加については，今回は三者ともコードの記述により行ったので，ほとんど同じ結果となった.Rails には画面をグラフィカルに作成する機能もあるが，細かな記述までしようとする場合には，コード記述によらなければならない．

③ログイン機能のように，Rails では Controller でルーティングにコードを修正・追加しなければならない場合がある．

4.2.2 フォルダ数

アプリケーション開発では，役割別にフォルダを作成するのが普通であることから，この数についても定量的に比較する必要があると思われる．三者の開発でのフォルダ数について見ると，フォルダ数はアプリケーションの規模や作成者の意図によって変化するものであるが，今回の eclipse3.7 での開発の場合には，以下のようになった．

①JSP：デフォルトの 8 個と Model・View・Controller の各ソースコード用フォルダ 3 個の計 11 個（他に派生的にできた実行ファイルの格納フォルダ 2 個）．

②Hibernate：デフォルトの 8 個と Model・View・Controller の各ソースコード用フォルダ 3 個，更にマッピングとデータベースのアクセス情報記述ファイル用のフォルダ 1 個で計 12 個（他に派生的にできた実行ファイルの格納フォルダ 2 個）．

③Rails：デフォルトで 36 個のフォルダ．

一般的には，フォルダ数が多いと修正・追加箇所の特定に時間がかかると考えられる．従って，この点ではデフォルトで作られるフォルダ数の多い Rails は，修正・追加箇所の特定が難しいと言える．しかしながら，これらのフォルダは役割別に分かれているので，Rails に習熟すれば修正・追加箇所の特定の困難性は緩和されるのではないかとも考えられる．

4.2.3 人間系を考慮した評価尺度

アプリケーションプログラムの修正の容易性を評価する場合，プログラムは人間が作るものである以上，どうしても人間系についても考慮せざるを得なくなる．今回は，これに関する評価尺度として，

(a)発見の容易性，
(b)修正操作の容易性，
(c)修正記述の容易性，

の三つを上げ，各指標の難易度について 1（易しい）～５（難しい）の５段階の相対評価を行った．以下にその評価について記し，結果をまとめたものを Table2 に示す．

Table2　修正者から見た難易度

	JSP	Hibernate	Rails
発見	3	3	5
修正操作	4	2	1
修正記述	4	2	4

（1易しい－5難しい：5段階相対評価）

(a)発見の容易性：Rails では，フォルダ数が多いため5とした．
(b)修正操作の容易性：JSP では，全てをコードの修正・追加のみで行うため 4[1] とし，Hibernate では，属性追加の際にリバースエンジニアリングファイルの更新により，永続化クラスとマッピングファイルのための Java 文が自動的に作成されるため2とした．Rails では，CRUD 機能はコマンド入力を行う事で自動的にコード作成され，属性の追加を行う場合も Model部分は,コマンド入力のみで済むため1とした．
(c)修正記述の容易性：JSP では，SQL 文と Java 文の記述で修正・追加を行うため 4[1] とし，Hibernate では，CRUD に関する部分は，SQL 文の記述を行う必要がなく Session.(CRUD機能)メソッドを記述することで処理が行えるため 2，Rails では，SQL 文が標準とは異なるため4とした．

各指標の評点を xi，各重みを mi で表すと，総合的な難易度 d は，

$$d = \Sigma xi \cdot mi \quad （但し，\Sigma mi=1） \tag{1}$$

と表されることになる．修正操作の容易性の重み mi については，コード記述とコマンド操作による自動生成の違いがあるので，他よりも重みを大にしても良いと思われるが，今回は，妥当な値の検証まで出来なかったので，とりあえず，全て同じ値で 1/3 ずつとした．すると，JSP は 3.7，Hibernate は 2.3，Rails は 3.3 となった． つまり，全てコードを記述する JSP が一番難しく，Rails はフォルダ数が多く，SQL の部分が標準と異なることから次に難しく，Hibernate が一番易しいとなったが，上述の修正操作の重みの変更でこの評価は変わるので，重みの値の検証が必要である．

5.　おわりに

本研究では，フレームワークを用いた場合のアプリケーションの修正・追加の容易性の評価を行った．Webアプリケーションを対象に Hibernate と Rails を取り上げ，JSP と比較検討を行うことで評価を行った．但し，本論文の例での修正・追加は，プログラムの構造に変更を及ぼすような大きなものではなく，恐らくはフレームワーク作成者が想定している範囲内のマイナーな変更である．しかしながら，Web アプリケーションでは，そのような修正・追加が比較的多いのではないかと考えられる．

定性的評価として特徴の比較，定量的評価としては LOC による比較，フォルダ数による比較，更に人間の感じる難易度による評価を試みた．それらを要約すると，フレームワークを用いた場合には，コードの作成・変更がコマンドにより自動的に行われるため容易と言えるが，コードが生成される場所が発見しにくい場合があり，また僅かな変更であっても Controller 部分に必須の記述項目があるような場合もあり，その点が難しいということになる．

フレームワークを用いた開発の評価で，良く習熟度が問題となる．あるフレームワークを初めて扱う場合には，学習が必要となるが，これを難易度の評価に加えて良いものかどうかという問題である．ところで，企業等でのフレームワークを用いた開発を考えた場合には，初回のみは学習が必要であるが，その後の開発では必要でなくなる．すると，フレームワークをある程度習熟した人間を前提に，プログラミングの難易度を評価するという方法もあって良いのではないかと考えられる．本研究では，この立場から，あえて，最初の開発における，フレームワークの学習時間は評価しないことにした．そして，その後の修正・追加の段階で，つまりフレームワークに習熟した段階で，評価を行うことにした．

また，アカデミックの世界における開発では，スケーラビリティの妥当性という点も問題視されることがある．この点に関しては，本研究のように，修正・追加という差分について評価することにすれば，この問題については多少は解消されるのではないかと考えられる．本来は，大規模なアプリケーションに対して評価すべきであろうが，現状，産業界においては，そのような人的余裕など無く実施困難であろうし，一方，アカデミックの世界においては，学生が言語だけの開発の場合にでも，プログラム構造やフレームワークに慣れるだけで，相当な時間を費やしてしまうことになる．しかしながら，フレームワークによる開発が多くなってきている現状を鑑みると，何となく良さそうだから使ってみようという人間の勘のようなものに頼るだけではなく，評価を行う必要があると考えられる．

フレームワークを用いた開発で他に問題になる点としては，バージョンが変わると API（Application Programming Interface）が変更になる場合があり，その影響を考慮しなければならない事が上げられる．

また，フレームワークを用いることによる処理速度の低下という問題もある．但し，これについては，コンピュータの処理性能の向上が著しいため，数年経てば問題ならなくなるケースもある．

本研究における，フレームワークの初期学習の扱いと差分を評価する方法の妥当性については，今後，議論が必要であると思われる．また，今後の課題と

[1] 基準とする JSP を 3 [普通]とすると,他が表現しにくくなるため，4 [やや難しい]とした．

して，本研究で提案した人間の感じる難易度の評価について，評価項目や妥当な重みの検証について更なる研究が必要であると考えられる．更には，フレームワークは，作成された目的によってサポートしている機能が異なるため，それらを単一の量で評価して良いかという問題があるが，一方，長所と短所の列挙による定性的な評価だけで良いのかという問題もあり，これも今後の課題として上げられる．

参考文献

1) 満田成紀,福安直樹: ウェブアプリケーションフレームワーク利用に潜む課題，コンピュータソフトウェア，Vol.27,No.3, pp.2-12 (2010).

2) 岡本隆，吉田英嗣，金子崇之，権藤夏男:Light Weight Java JSF/Hibernate/Spring によるフレームワークで Web アプリケーションの開発効率向上，毎日コミュニケーションズ (2005).

3) D. Thomas, D.H. Hansson, 前田修吾監訳: Rails によるアジャイル Web アプリケーション開発，オーム社 (2006).

4) まつもとひろゆき: プログラミング言語 Ruby の世界普及戦略，情報処理学会デジタルプラクティス，Vol.2, No.2, pp.74-79 (2011).

5) 中所武司，藤原克哉: Java による Web アプリケーション入門，サイエンス社 (2005).

6) B. G. Ryder, F. Tip: Change Impact Analysis for Object-Oriented Programs, PASTE’01, pp.46-53 (2001).

7) L.C. Briand, Y. Labiche, L. O’Sullivan, and M.M. Sowka: Automated impact analysis of UML models, The Journal of Systems and Software, No. 79, pp.339-352 (2006).

8) R.S. Arnold and S.A.Bohner: An Introduction to Software Change Impact Analysis, IEEE Computer Press (1996).

9) R.S. Arnold and S.A. Bohner: Impact Analysis - Towards A Framework for Comparison, ICSM2001, pp.292-301 (2001).

10) D. Kung, J. Gar, P. Hsia, F. Wen, Y. Togoshima, and C. Chen: Change Impact Identification in Object -Oriented Software Maintenance, ICSM 1994, pp.202-211 (1994).

11) S.A. Bohner: A Graph Traceability Approach for Software Change Impact Analysis, Ph.D. Dissertation, George Maison University(1995).

12) 早瀬康裕，松下誠，楠本真二，井上克郎，尾林健一，吉野利明: 影響波及解析を利用した保守作業の労力見積りに用いるメトリックスの提案，電子情報学会論文誌，Vol. J90-D, No. 10, pp.2736-2745 (2007).

13) 山田茂,高橋宗雄:ソフトウェアマネジメント入門，共立出版社 (1993).

14) 平山祐資,小野康一,深澤良彰:Ajax アプリケーションの保守容易性計測のためのソフトウェアメトリクス，情報処理学会研究報告 Vol.2012-SE-178, No.24, pp.143-150 (2012).

付録：開発アプリケーションでの画面

Fig. A1　図書一覧表示画面

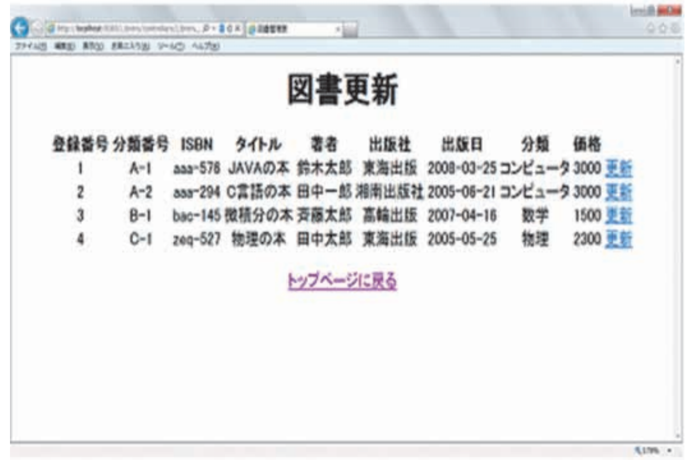

Fig. A4　図書更新選択画面

Fig. A2　図書登録画面

Fig. A5　図書更新画面

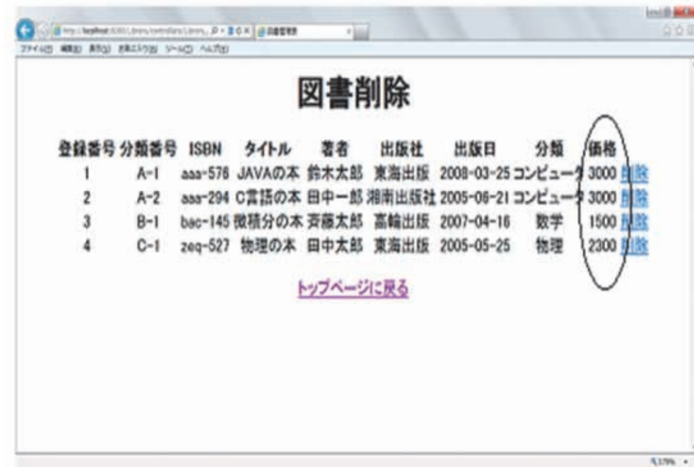

Fig. A3　図書削除画面

Fig. A6　ログイン画面