コラボレイティブ開発特論

中鉢欣秀

目次

イントロダクション

1. 事前準備

1.1. 資料の入手先

1. 資料等の入手先
    - GitHubの下記リポジトリにまとめておきます
        - https://github.com/ychubachi/collaborative_development
    - 資料は随時updateするので，適宜，最新版をダウンロードしてください

1.2. ソフトウエアのインストール

1. 仮想環境（Vagrant）
    - 各自のPCに仮想環境をインストールしておいてください
    - インストールするソフトウェア
    - Git
        - https://gitforwindows.org/
    - VirtualBox
        - https://www.virtualbox.org/
    - Vagrant
        - https://www.vagrantup.com/

1.3. GitHub/Herokuのアカウントを作成

1. GitHub
    - Join GitHub · GitHub
2. Heroku
    - Heroku - Sign up

2. 授業の全体像

2.1. 学習の目的

- アプリケーションソフトウエアを構築するための基礎力
- 分散型PBLを実施する上で必要となる知識やツールの使い方
- これら活用するための自己組織的なチームワーク

2.2. 学習の目標

- 分散ソフトウェア開発のための道具を学ぶ
  - 開発環境（Ruby），VCSとリモートリポジトリ（GitHub）
  - テスト自動化，継続的インテグレーション，PaaS
- これらのツールの 設計思想 に対する本質理解

2.3. 前提知識と到達目標

1. 前提とする知識
   - 情報系の学部レベルで基礎的な知識を持っていること
2. 最低到達目標
   - 授業で取り上げる各種ツールの基本的な使い方を身につける
3. 上位到達目標
   - 授業で取り上げる各種ツールの高度な使い方に習熟する.

2.4. 授業の形態

1. 対面授業
   - 担当教員による講義・演習
2. 個人演習
   - 個人によるソフトウエア開発
3. グループ演習
   - グループによるソフトウエア開発

3. 授業の方法

3.1. 講義・演習・課題

1. 講義
    - ツールの説明
    - ツールの使い方
2. 演習
    - 個人でツールを使えるようになる
    - グループでツールを使えるようになる

3.2. 成績評価

1. 課題
    - 個人でソフトウエアを作る
    - グループでソフトウエアを作る
2. 評価の方法
    - 課題提出と実技試験
3. 評価の観点
    - PBLで役に立つ知識が習得できたかどうか

4. 「学びの共同体」になろう

4.1. 共に学び、共に教える「場」

- 教室に集うメンバーで 学びの共同体 になろう
- 困った時には助けを求める
- 他人に教えること＝学び

4.2. チーム演習での問題解決（理想の流れ）

1. 困った時はメンバーに聞く
2. わからなかったらチーム全員で考える
3. それでもダメなら他のチームに相談
4. 講師・コーチに尋ねるのは最終手段！
5. …となるのが理想
    - 授業の進め方などの質問は遠慮無く聞いてください

4.3. 共同体になるためにお互いを知ろう

- 皆さんの自己紹介
    - 名前（可能であれば所属も）
    - どんな仕事をしているか（あるいは大学で学んだこと）
    - この授業を履修した動機

コラボレイティブ開発の道具達

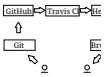
5. モダンなソフトウエア開発とは

5.1. ソフトウエア開発のための方法・言語・道具

5.2. 授業で取り上げる範囲

1. 取り上げること
    - 良い道具には設計思想そのものに方法論が組み込まれている
    - 世界中の技術者の知恵が結晶した成果としてのOSSのツール
2. 取り扱わないこと
    - 方法論そのものについてはアジャイル開発特論で学ぶ
    - プログラミングの初歩については教えない

5.3. Scrumするためのモダンな道具たち



5.4. モダンな開発環境の全体像

1. 仮想化技術（Virtualization）
    - WindowsやMacでLinux上でのWebアプリケーション開発を学ぶことができる
    - HerokuやTravis CI等のクラウドでの実行や検査環境として用いられている
2. ソーシャルコーディング（Social Coding）
    - LinuxのソースコードのVCSとして用いられているGitを学ぶ
    - GitはGitHubと連携することでOSS型のチーム開発ができる

6. enPiT仮想化環境

6.1. 仮想環境にインストール済みの道具

- エディタ（Emacs/Vim）
- Rubyの実行環境
- PostgreSQLのクライアント・サーバーとDB
- 各種設定ファイル（.bash_profile，.gemrc，.gitconfig）
- その他
  1. 仮想化環境の構築用リポジトリ（参考）
- ychubachi/vagrant_enpit

6.2. enPiT仮想化環境にログイン

1. 作業内容
   - 前の操作に引き続き，仮想化環境にSSH接続する
   - 以降，仮想環境のシェル（bash）で作業する
2. コマンド

```
vagrant up
vagrant ssh
```

6.3. enPiT仮想環境の停止

1. 作業内容
   - 仮想環境のシェルからexitした後，次のコマンドを実行
2. コマンド

```
vagrant halt
```

7. 仮想環境の準備から起動

7.1. Port Forwardの設定(1)

1. 説明
   - Guest OSで実行するサーバに，Host OSからWebブラウザでアクセスできるようにしておく
   - 任意のエディタでVagrantfileの「config.vm.network」を変更
   - 任意のエディタでVagrantfileを変更

7.2. Port Forwardの設定(2)

1. 変更前

```
# config.vm.network "forwarded_port", guest: 80, host: 8080, host_ip: "127.0.0.1"
```

2. 変更後

```
config.vm.network "forwarded_port", guest: 3000, host: 3000, host_ip: "127.0.0.1"
config.vm.network "forwarded_port", guest: 4567, host: 4567, host_ip: "127.0.0.1"
```

Git/GitHub演習（別資料）

8. Git/GitHub演習について

1. Git/GitHub演習

GitとGitHubにとことん精通しよう

1. 演習資料

Git/GitHub演習

Ruby on Rails/Heroku

9. Ruby on Railsアプリの生成と実行

9.1. RoRを使ったWebアプリケーション

1. Ruby on Rails（RoR）とは？
    - Webアプリケーションを作成するためのフレームワーク
2. 参考文献
    - Ruby on Rails

9.2. 準備（ごめんなさい，入れ忘れでした）

- railsコマンドとライブラリを追加

```
gem install rails
sudo apt update
sudo apt -y install libpq-dev
```

9.3. GitHubに`rails_enpit`レポジトリを作る

- GitHubのページにてレポジトリを作成する
- 作成時，必ず「Initialize this repository with a README」にチェックを入れること

9.4. `rails_enpit` を作成するコマンド

- `rails new` コマンドを用いて，RoRアプリの雛形を作成する

```
git clone 【rails_enpitのURL】
cd ~/rails_enpit
rm README.md
rails new . --database=postgresql
git add .
git commit -m 'Generate a new rails app'
git push
```

9.5. GemfileにJS用Gemの設定

- GemfileにRails内部で動作するJavaScriptの実行環境を設定する
    - 当該箇所のコメントを外す
        1. 変更前

```
# gem 'mini_racer', platforms: :ruby
```

2. 変更後

```
gem 'mini_racer', platforms: :ruby
```

9.6. Bundle installの再実行

- Gemfileを読み込み，必要なgemをインストールする
  - `rails new` をした際にも， `bundle install` は実行されている
  - `mini_racer` と，それが依存しているgemで まだインストールしていないものをインストール
    1. コマンド

```
bin/bundle install
```

- `+bin/bundle+`と+bin/+`...をコマンドにつけるのは，プロジェクトのbundle配下にあるバージョンのコマンドを明示的に実行させるため

9.7. Gemfile設定変更のコミット

- ここまでの内容をコミットしておこう
  1. コマンド

```
git add .
git commit -m 'Edit Gemfile to enable the rubyracer gem'
git push
```

10. データベースの作成

10.1. アプリのデータベースを作成する

- rails_enpitアプリの動作に必要なDBを作成する
- DatabeseはHerokuで標準のPostgreSQLを使用する
  - RoR標準のsqliteは使わない
- enPiT仮想環境にはPostgreSQLインストール済み

10.2. PostgreSQLにDBを作成

1. 開発で利用するDB

rails_enpit_development	開発作業中に利用
rails_enpit_test	テスト用に利用
(rails_enpit_production)	（本番環境用）

   - 本番環境用DBは**Heroku**でのみ用いる
2. コマンド

```
createdb rails_enpit_development
createdb rails_enpit_test
```

10.3. PostgereSQLクライアントのコマンド

1. クライアントの起動
   - `psql` コマンドでクライアントが起動
2. psqlクライアンで利用できるコマンド

Backslashコマンド	説明
\l	DBの一覧
\c	DBに接続
\d	リレーションの一覧
\q	終了

10.4. Rails serverの起動

- 次のコマンドでアプリケーションを起動できる
  1. コマンド

```
bin/rails server -b 0.0.0.0
```

10.5. Webアプリの動作確認

- Host OSのWebブラウザで， `http://localhost:3000` にアクセスして確認
- 端末にもログが表示される
- 確認したら，端末でCtrl-Cを押してサーバを停止する

11. Controller/Viewの作成

11.1. Hello Worldを表示するController

- HTTPのリクエストを処理し，Viewに引き渡す
  - MVC構造でいうControllerである
- `rails generate controller` コマンドで作成する

```
bin/rails generate controller welcome
```

11.2. 生成されたControllerの確認とコミット

- git statusコマンドでどのようなコードができたか確認

```
git status
```

- Controllerのコードを作成した作業をコミット

```
git add .
git commit -m 'Generate the welcome controller'
```

11.3. Hello Worldを表示するView

- HTML等で結果をレンダリングして表示する
  - erbで作成するのが一般的で，内部でRubyコードを動作させることができる
- `app/views/welcome/index.html.erb` を（手動で）作成する
  - コードは次スライド

11.4. Hello Worldを表示するViewのコード

1. **index.html.erb**

```
<h2>Hello World</h2>
<p>
  The time is now: <%= Time.now %>
</p>
```

11.5. 作成したViewの確認とコミット

- git statusコマンドで変更内容を確認

```
git status
```

- Viewのコードを作成した作業をコミット

```
git add .
git commit -m 'Add the welcome view'
```

11.6. routeの設定

- Routeとは？
- HTTPのリクエスト（URL）とコントローラを紐付ける設定
    - ここでは `root` へのリクエスト（ `GET /` ）を

`welcome` コントローラの `index` メソッドに紐付ける

- `config/routes.rb`を次の通り書き換える

```
Rails.application.routes.draw do
  root 'welcome#index'
end
```

- `bin/rake routes` コマンドで確認できる

11.7. routes.rbの設定変更の確認

- `routes.rb` は既にトラッキングされているので， git diffコマンドで変更内容を確認できる

```
git diff
```

- routes.rbを変更した作業をコミット

```
git add .
git commit -m 'Edit routes.rb for the root controller'
```

11.8. ControllerとViewの動作確認

- 再度， `rails server` でアプリを起動し，動作を確認しよう

- Webブラウザで `http://localhost:3000/` を開く
  1. コマンド

     ```
     bin/rails server -b 0.0.0.0
     ```

11.9. GitHubへのPush

- ここまでの作業で，controllerとviewを1つ備えるRoRアプリができた
- 作業が一区切りしたので，GitHubへのpushもしておく
  - 一連の作業を `git log` コマンドで確認してみると良い
    1. コマンド

       ```
       git push
       ```

12. Herokuにディプロイする

12.1. RoRをHerokuで動かす

- 作成しとRoRアプリをHerokuで動作させせよう
- Getting Started on Heroku with Rails 5.x

12.2. Heroku用設定をGemfileに追加

- GemfileにRubyのバージョンを指定しておく

```
ruby '2.5.5'
```

- `Gemfile` を変更したら必ず `bundle install` すること

12.3. デプロイ前にGitにコミット

- Herokuにコードを送るには，gitを用いる
  - 従って，最新版をcommitしておく必要がある
- commitし，まずはGitHubにpushしておく
  1. コマンド

```
git commit -a -m 'Set up for Heroku'
git push
```

- 2行目: pushする先はorigin（=GitHub）である

12.4. Herokuアプリの作成とディプロイ

- `heroku` コマンドを利用してアプリを作成する
  1. コマンド

```
heroku create
git push heroku master
```

- 1行目: `heroku create` で表示されたURLを開く
- 2行目: `git push` はherokuのmasterを指定． ディプロイすると，Herokuからのログが流れてくる

12.5. Herokuへのディプロイの自動化

- HerokuにGitHubを直接接続することで，ディプロイを自動化することができる。

1. Herokuでアプリケーションのダッシュボードを開く
2. DeployタブのDeployment methodからGitHubを選ぶ
3. GitHubのURLを設定し，Automatic deploysをenableする
4. 最初は，Manual deployをする

- 以後，GitHubのmasterが更新されるたび，自動でディプロイされる

13. 演習課題

13.1. 演習課題

1. RoRアプリの作成
    - ここまでの説明に従い，Herokuで動作するRoRアプリ（ `rails_enpit` ）を完成させなさい

14. DBを使うアプリの開発と継続的統合

14.1. Scaffold

1. Scaffoldとは
    - https://www.google.co.jp/search?q=scaffold&client=ubuntu&hs=PiK&channel=fs&hl=ja&source=lnms&tbm=isch&sa=X&ei=smUdVKaZKY7s8AXew4LwDw&ved=0CAgQ_AUoAQ&biw=1195&bih=925[scaffold
        - Google 検索]
2. RoRでは，MVCの雛形のこと
    - CRUD処理が全て自動で実装される

14.2. Scaffoldの生成方法

1. コマンド

```
git checkout -b books
bundle exec rails generate scaffold book title:string author:string
```

    - 多くのコードが自動生成されるので，branchを切っておくと良い
        - 動作が確認できたらbranchをマージ
        - うまく行かなかったらbranchごと削除すれば良い

14.3. routeの確認

- Scaffoldの生成で追加されたルーティングの設定を確認
    1. コマンド

```
bundle exec rake routes
```

- `git diff` でも確認してみよう

14.4. DBのMigrate

1. migrateとは
    - Databaseのスキーマ定義の更新
    - Scaffoldを追加したり，属性を追加したりした際に行う
2. コマンド

```
bundle exec rake db:migrate
```

14.5. 参考：Migrateの取り消しの方法

- DBのmigrationを取り消したいときは次のコマンドで取り消せる

```
bundle exec rake db:rollback
```

- 再度，migrateすれば再実行される

```
bundle exec rake db:migrate
```

14.6. 参考：Scaffold作成の取り消しの方法

1. コマンド

```
git add .
git commit -m 'Cancel'
git checkout master
git branch -D books
```

   - 1〜2行目：自動生成されたScaffoldのコードをbranchに一旦コミット
   - 3行目：masterブランチに移動
   - 4行目：branchを削除（ `-D` オプション使用 ）

14.7. 動作確認

1. 動作確認の方法
   - Webブラウザで http://localhost:3000/books を開く
   - CRUD処理が完成していることを確かめる
2. コマンド

```
bundle exec rails server
```

14.8. 完成したコードをマージ

1. ブランチをマージ
   - 動作確認できたので， `books` branchをマージする
   - 不要になったブランチは， `git branch -d` で削除する
2. コマンド

```
git add .
git commit -m 'Generate books scaffold'
git checkout master
git merge books
git branch -d books
```

14.9. Herokuにディプロイ

1. ディプロイ
   - ここまでのアプリをディプロイする
   - herokuにあるdbもmigrateする
   - Webブラウザで動作確認する
2. 設定ファイル(Procfile)

```
release: bundle exec rake db:migrate
web: bundle exec rails server -p $PORT
```

15. RoRアプリのテスト

15.1. テストについて

1. ガイド
    - A Guide to Testing Rails Applications — Ruby on Rails Guides

15.2. テストの実行

1. テストコード
    - Scaffoldはテストコードも作成してくれる
    - テスト用のDB（ `rails_enpit_test` ）が更新される
2. コマンド

```
bin/exec rake test
```

16. Travis CIとの連携

16.1. Travis CIのアカウント作成

1. アカウントの作り方
    - 次のページにアクセスし，画面右上の「Sign in with GitHub」のボタンを押す
        - Travis CI - Free Hosted Continuous Integration Platform for the Open Source Community
    - GitHubの認証ページが出るので，画面下部にある緑のボタンを押す
    - Travis CIから確認のメールが来るので，確認する

16.2. Travisの設定

1. 設定ファイルの変更
    - まず、Rubyのバージョンを指定する
    - 変更の際はYAMLのインデントに注意する
2. .travis.yml を書き換える

```
language: ruby
rvm:
- 2.5.5
```

16.3. Travis用DB設定ファイルと作成

- RubyのVersionなど
- テストDB用の設定ファイルを追加する
    1. `.travis.yml`

```
language: ruby
rvm:
- 2.5.5
services: postgresql
bundler_args: "--without development --deployment -j4"
cache: bundler
before_script:
  - cp config/database.travis.yml config/database.yml
  - bundle exec rake db:create
  - bundle exec rake db:migrate
script: bundle exec rake test
```

    2. `config/database.travis.yml`

```
test:
  adapter: postgresql
  database: travis_ci_test
  username: postgres
```

16.4. GitHubとTravis CI連携

1. 説明
    - ここまでの設定で，GitHubにpushされたコードはTravis CIでテストされるようになった.
    - GitHubにプッシュしてWebブラウザでTravis CIを開いて確認する
2. コマンド

```
git add .
git commit -m 'Configure Travis CI'
git push
```

16.5. CI通過後のHerokuへの自動deploy

1. HerokuへのDeploy
    - テストが通れば，自動でHerokuに配備されるように、Herokuに設定を追加する

17. 演習課題

17.1. 演習課題7-1

1. `rails_enpit` の拡張
    - Viewを変更
        - welcomeコントローラのviewから， booksコントローラのviewへのリンクを追加する etc
    - Scaffoldの追加
        - 任意のScaffoldを追加してみなさい
        - DBのmigrationを行い，動作確認しなさい
    - Herokuへの配備
        - Travis経由でHerokuへdeployできるようにする

補足資料

18. 90min Scrum

表 *1. 90min Scrum* タイムテーブル

1	計画をたてる(Sprint Planning)	10分	計画を立て終えたら開発に進んでよい
2	開発する(Development)	50分	途中 (Daily) Scrumをするとよい
3	成果物レビュー(Sprint Review)	15分	動くもののデモ
4	振り返り(Sprint Retrospective)	15分	KPT　(Keep-Probrem-Try)

19. Vagrant関連

19.1. Vagrantの補足

1. 仮想環境とのファイル共有
    - Guest OS内に `/vagrant` という共有フォルダがある
    - このフォルダはHost OSからアクセスできる
    - 場所はVagrantfileがあるフォルダ
    - 共有したいファイル（画像など）をここに置く

20. Git関連

20.1. Gitの補足

1. 元いたbranchに素早く戻る方法

```
git checkout other_branch # masterで
# 編集作業とcommit
git checkout - # masterに戻る
```

2. Git brame
    - だれがどの作業をしたかわかる（誰がバグを仕込んだのかも）
        - Using git blame to trace changes in a file · GitHub Help

20.2. バイナリのコンフリクト(1)

- git mergeでバイナリファイルがコンフリクトした場合、 ファイルはgit merge実行前のままとなります[1]。
- 以下を実行してコンフリクトが発生したとします。

```
git checkout master
git merge branch_aaa
```

20.3. バイナリのコンフリクト(2)

- そのままにしたいとき(=masterを採用）は

```
git checkout --ours <binaryfile> #明示的な実行は不要
git add <binaryfile>
git commit
```

- branch_aaaのファイルを採用したいときは

```
git checkout --theirs <binaryfile>
git add <binaryfile>
git commit
```

20.4. Hubコマンドについて

- enPiT環境にはHubコマンドが仕込んである
    - github/hub

- 通常のGitの機能に加えて，GitHub用のコマンドが利用できる
  - コマンド名は「git」のまま（エイリアス設定済み）
- 確認方法

```
git version
alias git
```

[1] git mergeでバイナリファイルがコンフリクトした場合 · Issue #6

21. GitHub関連

21.1. GitHubの補足(1)

1. Issue
    - 課題管理（ITS: Issue Tracking System）
    - コミットのメッセージでcloseできる
        - Closing issues via commit messages · GitHub Help
2. Wiki
    - GitHubのリポジトリにWikiを作る
        - About GitHub Wikis · GitHub Help

21.2. GitHubの補足(2)

1. GitHub Pages
    - 特殊なブランチを作成すると，Webページが構築できる
        - GitHub Pages

22. Heroku関連

22.1. Herokuの補足

1. HerokuのアプリのURL確認

```
heroku apps:info
```

2. `rails generate` などが動かない

```
spring stop
```

23. Travis CI関連

23.1. Travis CIの補足

1. Status Image
    - README.mdを編集し，Travisのテスト状況を表示するStatus Imageを追加する
    - Travis CI: Status Images
2. Deploy後、自動で heroku の db:migrate
    - 次のURLの「Running-commands」の箇所を参照
        - Heroku Deployment - Travis CI