卒業論文

my_help のテスト駆動開発

関西学院大学理工学部

情報科学科 西谷研究室

27018512 大寺華椰

2022 年 3 月

概 要

本研究では，my_help のテストを作成する事により，my_help が想定通りに動くと保証する事を目的とした．また，その過程で，my_help の書き換えも行った．テスト作成は Unit Test で行う事にする．

結果として，my_help の幾つかの機能の中で，list (my_help 内の list 表示機能)，version (my_help の version を示す機能)，set_editor (エディタの選択・変更機能)，edit (作成 help の編集機能)，以上 4 つのテスト作成，及び version メソッド (lib/my_help/my_help.rb)，set_editor メソッド (lib/my_help/my_help.rb)，set_editor メソッド (lib/my_help/my_help_controll.rb)，init_help メソッド (lib/my_help/my_help_controll.rb)，以上 4 つの書き換えを行った．

目 次

図 目 次

第1章　序論

西谷研では，my_help をユーザーメモソフトとして使用している．my_help が提供する幾つかの振る舞いは，他のメモソフトより，効率的で，無駄を省く事ができる．しかし現状では，テストのカバー率は低い．そこで，より多くの状況を想定したテストを加え，最終的には，my_help が正常に動いていると保証する事を本研究の目的とした．また，テスト作成の過程で，my_help に影響してしまった場合には，my_help の書き換えも行う事とした．

my_help とは「CUI(CLA) ヘルプの Usage 出力を真似て，user 独自の help を作成・提供する gem である [1].」my_help で解決したいユーザーメモソフトの問題点としては，

- man は基本的に英語
- manual では重たい
- いつもおなじことを web で検索して
- 同じとこを見ている
- memo しても，どこへ置いたか忘れる．

などがある [1]．上記の問題点を CLI(CUI) 環境として提供する事が，my_help の目的である．仕様としては，

- user が自分にあった man を作成
- 雛形を提供
- おなじ format,looks, 操作, 階層構造
- すぐに手が届く
- それらを追加・修正・削除できる

などがある [1].

第2章　my_helpの使用法

2.1　インストール

ターミナル上で，以下のコマンドを入力し，my_help をインストールする事により使用可能である．

```
> gem install my_help
```

2.2　my_helpのコマンドについて

my_help のコマンド一覧を表示させたい場合は，以下のコマンドを入力すれば良い．

```
> my_help
```

```
Commands:
  my_help delete HELP             # delete HELP
  my_help edit HELP               # edit HELP
  my_help git [push|pull]         # git push or pull
  my_help help [COMMAND]          # Describe available co$
  my_help list [HELP] [ITEM]      # list all helps, speci$
  my_help new HELP                # make new HELP
  my_help set_editor EDITOR_NAME  # set editor to EDITOR_$
  my_help setup                   # set up the test datab$
  my_help version                 # show version
```

次に，コマンド一覧の中から幾つかのコマンドの入力方法とその出力結果を以下に記す．

- version コマンド (my_help の version 表示機能)

```
> my_help version
  1.0b
```

私の環境では，上記の出力結果である．

- set_editor コマンド (エディタの選択や変更の機能)

```
> my_help set_editor emacs
  set editor 'emacs'
```

先ずは，どのエディタを用いるかを選択する必要がある．今回は，emacs を使用しているが，他のエディタを用いたい場合は，上記の emacs 部分を，使用したいエディタ名に変更すれば良い．このコマンドでいつでもエディタを変更する事ができる．

- new コマンド (独自の help 作成機能)

```
> my_help new sample_help
"/Users/ooterahana/.my_help/sample_help.org"
"/Users/ooterahana/.rbenv/versions/2.7.2/lib/ruby/gems/2.7.0/
gems/my_help-0.9.0/lib/templates/help_template.org"
cp /Users/ooterahana/.rbenv/versions/2.7.2/lib/ruby/gems/2.7.0/
gems/my_help-0.9.0/lib/templates/
help_template.org /Users/ooterahana/.my_help/sample_help.org
```

これで，sample_help という独自の help が作成できる．今回は，sample_help という help を例として作成したが，他の新しい help を作成したい場合には，sample_help 部分に作成したい help 名を打ち込めば良い．

- edit コマンド (作成 help の編集機能)

```
> my_help edit sample_help
"/Users/ooterahana/.my_help/sample_help.org"
```

edit コマンドを用いる事により，先程作成した sample_help での編集が可能になる．上記のコマンドを実行すると，以下の画面 2.1が開かれる．Emacs org 形式で格納されている．

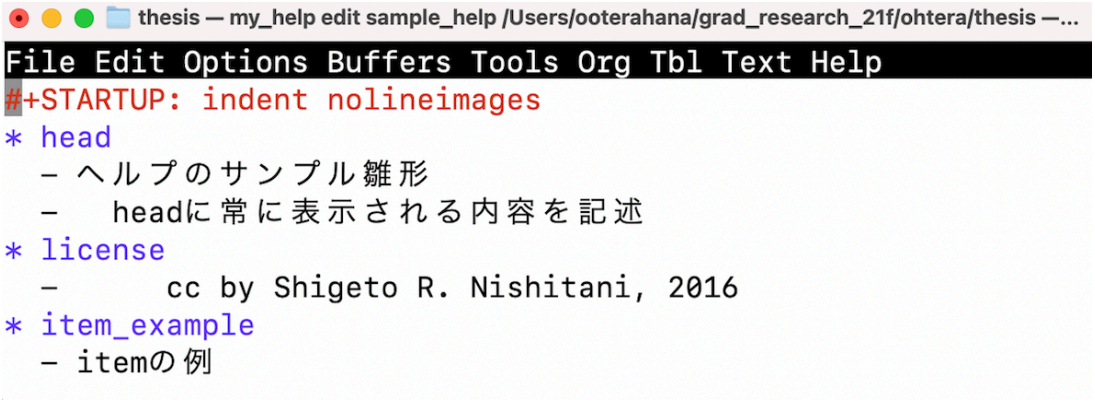

図 2.1: 作成した sample help の org 画面.

- delete コマンド (作成 help の削除機能)

```
> my_help delete sample_help
```

と入力すると,

```
Are you sure to delete
/Users/ooterahana/.my_help/sample_help.org?[Yn]
```

というような問いかけがくるので，削除したい場合は,

```
> Y
  rm /Users/ooterahana/.my_help/sample_help.org
```

と入力すれば，sample_help は削除される．削除したくない場合は,

```
> N
```

と入力すれば良い．他の作成 help を削除したい場合には，sample_help 部分の help 名を削除したい help 名に変更すれば良い.

第3章　開発手法

my_help のコードの見直しに当たってどのような手順で開発を進めるのが，最適であるかを考えた．

3.1　テスト駆動開発

ソフトウェア開発の長い議論の中から生まれたシンプルな考え方に,「Code が動作するか知りたいならば，テストする必要がある」という Russ Olsen の言葉がある [1, p.79]．最新の開発手法として，Kent Beck らが提唱している eXtreme Programming(XP) の中でも，テスト駆動開発 (TDD) は中心的な手法として紹介されている．TDD とは，以下のような手順を何回も繰り返して進める開発手法である [4].

1. テストを作る
2. エラーを出す (red)
3. エラーをなくす (green)
4. code を綺麗にする (refactoring)

3.2　Unit Test

XP では，テストは大きく Acceptance Test(受け入れテスト) と Unit Test(単体テスト) の 2 つに分類される [3, p.21]．Acceptance Test は，そのソフトウェアのユーザーの視点で行われるテストだ．一方 Unit Test は，ソフトウェアの開発者の視点で行われる．本研究では，クラス構成の見直しが必要であり，開発者視点に立った Unit Test での開発を進める事にした．

3.3　テスティングフレームワーク

さらに「勤務時間のすべてを手作業のテストに費やしたくないならば，コンピューターがユーザーに代わってコードを調べるテストフレームワークが必要 [1, p.79]」とされる．Testing Framework(テスティングフレームワーク) とは，ソフトウェアのテスト用プログラムを簡単に作成し，テストの実行を支援するためのツールである [3, p.21]．コマンドでプログラムを動かすだけでは，動作内容の記録が残らない．ターミナル上で rake test と入力し実行すると，これまで作成してきたテストがいつでも動く．これは，テストが正常に動いた証拠や確認となる．

3.4　方針

以上の通り，テスト駆動開発, Unit Test，テスティングフレームワークという，3 つの手法を駆使して開発を行なっていく．TDD の最終目標である「動く綺麗なコードを作ること [4]」に従ってテスト駆動からリファクタリングによって my_help のコードの見直しを行う．

第4章　my_helpの環境構築

先ず，my_helpを編集可能にする為に，daddygongonのGithubで公開されているmy_helpをforkし，自分のリポジトリに移す作業を以下に記す．このサイト[1]のurlを開くと，以下の図4.1のように表示される．この画面の右側にある緑のcodeをクリックし，SSHアドレスをコピーする．

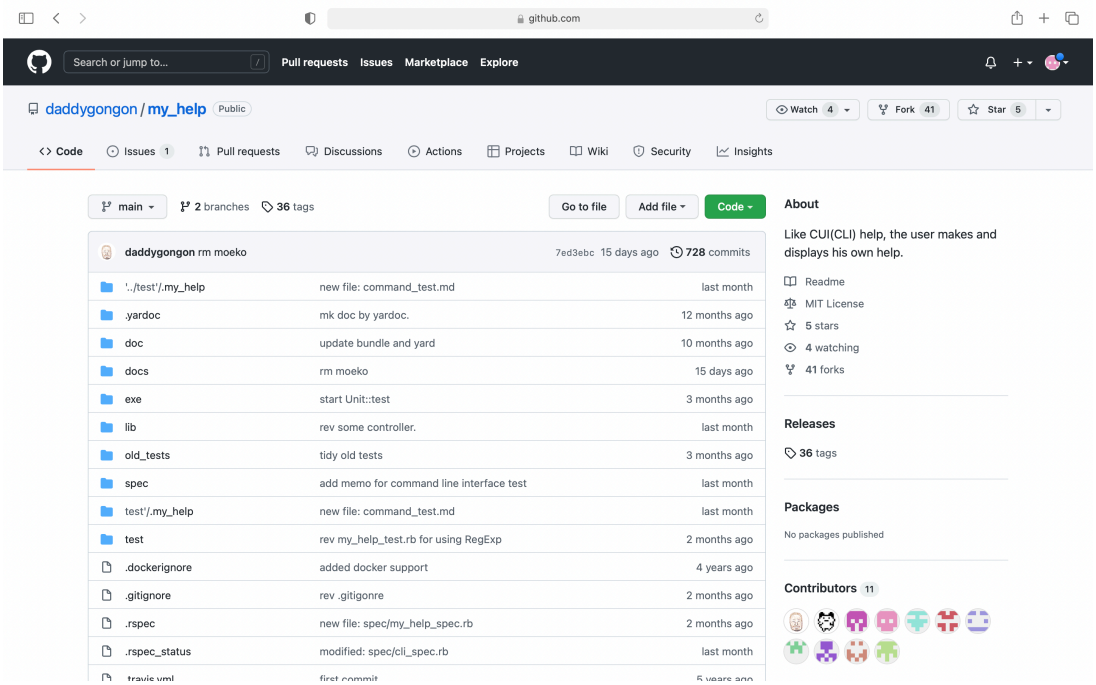

図 4.1: daddygongonのGithubで公開されているMyHelpのサイト画面.

ターミナルに移り，以下の通り入力する．

```
> git clone 上記でコピー済みのSSHアドレス
```

複数人でmy_helpを編集すると，自分以外の編集者に影響を及ぼしてしまう可能性が出てくる．そこで，Githubのpull_request機能を用いる．pushする前に，Github上で変更箇所をpull_requestする事で，それぞれの編集者は円滑な作業が可能となる．先ず，リモートリポジトリの確認を行う．

```
> git remote -v
```

上記のコマンドを入力する事で，現時点でのリモートリポジトリのパスを確認する事ができる．

```
origin  git@github.com:daddygongon/my_help.git (fetch)
origin  git@github.com:daddygongon/my_help.git (push)
```

次に，リモートリポジトリにフォーク元のリポジトリを追加する．この時，フォーク元のリポジトリは upstream という名前で登録する事にする．

```
> git remote add upstream git@github.com:daddygongon/my_help.git
```

もう一度,

```
> git remote -v
```

として，出力結果が以下のようならば，my_help の環境構築完了となる．

```
origin git@github.com:ha88te/my_help.git (fetch)
origin git@github.com:ha88te/my_help.git (push)
upstream git@github.com:daddygongon/my_help.git (fetch)
upstream git@github.com:daddygongon/my_help.git (push)
```

第5章　結果

本研究では，my_help の幾つかの機能の中で，

- my_help version # show version
- my_help set_editor EDITOR_NAME # set editor to EDITOR_NAME
- my_help edit HELP # edit HELP
- my_help list [HELP] [ITEM] # list all helps, specif...

のテストを作成した．本研究では，Unit Test でクラスの入出力をテストする．my_help の CLI の動作は lib/my_help/my_help.rb にある，CLI 構築ライブラリの Thor で記述している．my_help のテスト作成において，

- my_help の各メソッドのコード
- そのコマンドをターミナルで入力した時の出力結果

以上の 2 つを照らし合わせながらテストを作成していく．その過程で，必要に応じて lib/my_help/my_help.rb の改良も行う．

5.1　Version メソッドの改良

先ず最初に Version メソッドのテスト作成から始める．

5.1.1　テスト作成の為の準備内容

lib/my_help/my_help.rb での Thor で書かれた Version メソッドのコードは以下の通りである．

```
1  desc 'version', 'show version'
2      def version
3        invoke :setup
4        puts VERSION
5      end
```

my_help の Version コマンドを入力した時の出力結果は,

```
> my_help version
  1.0b
```

となる．これを参照しながらテストを作成した.

5.1.2 最終テストコード

最終的なテストコードは以下のようになった.

```
1  sub_test_case "Version" do
2    test "show version" do
3      expected = "1.0b"
4      assert_equal expected, VERSION
5    end
6  end
```

これは，Unit Test の典型的な記述である．テストコードの内容を以下で詳しく説明する．1 行目に使われている sub_test_case は，Test Unit が用意しているテストをネストする為の機能で，よく似たテストをまとめる事によって一覧性を高める機能である．よって 1 行目は，テストを行うメソッド (今回は Version メソッド) の sub_test_case というテストを実行する事を宣言している．2 行目は，Version メソッドのテストを show version という名前をつけて実行する．3 行目は，ターミナル上で my_help version と入力した際に，出力されてほしい期待値を文字列で用意する．今回の期待値は，1.0b である．4 行目に使われている assert_equal は,「equal かどうかを確かめる (assert) 関数」である．よって，4 行目が示す意味は，期待したオブジェクト (expected) と実際のオブジェクト (VERSION) が，同じ出力

結果だった場合 (expected == VERSION が真の場合) に，アサーションが成功する．今回の変数 expected は，3 行目の期待値 1.0b である．VERSION は，lib/my_help/my_help.rb の Version メソッドで出力される結果の事である．つまりは，ターミナル上で，my_help version とした時に出力される結果の事である．

5.1.3 my_help の改良と pull_request

これらの過程から，lib/my_help/my_help.rb のコードの見直しも行ったので，コードの改良点と pull_request 内容を以下に記す．lib/my_help/my_help.rb にて，3 行目の invoke :setup は Version メソッドの結果を出力する為に，全く必要のないコードである．そこで，西谷に pull_request をかけ，lib/my_help/my_help.rb のメソッド Version を以下のように改良した．

```
1  desc "version", "show version"
2      def version
3        #invoke :setup
4        puts VERSION
5      end
```

この節での作業をまとめると，Version メソッドに関する改良として，

- test/my_help_test.rb に Version メソッドのテスト作成
- lib/my_help.rb の Version メソッドの書き換え

を行った．

5.2 Set_Editor メソッドの改良

次に Set_Editor メソッドのテスト作成に移る．

5.2.1 テスト作成の為の準備内容

lib/my_help/my_help.rb での Thor で書かれた Set_Editor メソッドのコードは以下の通りである.

```
1  desc "set_editor EDITOR_NAME", "set editor to EDITOR_NAME"
2      def set_editor(editor_name)
3        invoke :setup
4        $control.set_editor(editor_name)
5      end
```

my_help の Set_Editor コマンドを入力した時の出力結果は,

```
> my_help set_editor emacs
  set editor 'emacs'
```

となる. これを参照しながらテストを作成した.

5.2.2 最終テストコード

最終的なテストコードは以下のようになった.

```
1  sub_test_case "Set Editor" do
2      test "set_editor" do
3        expected = "set editor 'emacs'"
4        conf_path = File.join(Dir.pwd, "test")
5        assert_equal expected, Control.new(conf_path).set_editor("emacs")
6      end
7    end
```

これは, Unit Test の典型的な記述である. テストコードの内容を以下で詳しく説明する. 1 行目, 2 行目は Version メソッドのテストコードの 1 行目, 2 行目と同様の意味を示す為, 省略する. 3 行目は, ターミナル上で my_help set_editor eamcs と入力した際に, 出力されてほしい期待値を文字列で用意し, 変数 expected に代入している. 今回

の期待値は，set editor 'emacs' である．4 行目の File.join は，第一引数と第二引数を連結させる関数である．デフォルトでは，~/my_help/my_help.conf を参照する為，テスト環境に依存して，異なった挙動を取る．これを統一する為，conf_path に，環境依存しない path を指定する．今回の conf_path には現在の実行ディレクトリに"test"を付けた path を指定している．5 行目の assert_equal は Version メソッドで使われていた意味と同様である．第一引数の変数 expected は，3 行目で代入された set editor 'emacs' である．一方，第二引数である Control.new(conf_path).set_editor("emacs") のコード説明は以下の通りである．先ず，Controll.new で新しいオブジェクト Controll が生成される．その後，lib/my_help/my_help_controll.rb の initialize メソッドが実行される．このメソッドは，new でオブジェクトが生成された時に必ず 1 度は実行されるメソッドである．なので，1 度 initialize が呼び出されるが，デフォルト動作では何もしない．まとめると，3 行目で代入した変数 conf_path を引数とした新しいオブジェクト Controll が生成された後，lib/my_help/my_help.rb の Set_Editor メソッドが，文字列 emacs を引数として呼ばれている．

5.2.3 my_help の改良と pull_request

これらの過程から，テスト実行におけるエラー内容，lib/my_help/my_help.rb のコードの改良点と pull_request 内容を以下に記す．先程作成した Set_Editor メソッドのテストを動かした所，以下のように，nil が返ってきた．

```
<"set editor 'emacs'"> expected but was
<nil>
```

これは，期待された出力内容 set editor 'emacs' が存在しない事を示す．原因は，以下のコード，lib/my_help/my_help_controll.rb の Set_Editor メソッドにある．

```
1  def set_editor(editor)
2      @conf[:editor] = editor
3       conf = { editor: editor }
4       File.open(@conf[:conf_file], "w") { |f| YAML.dump(conf, f) }
5       puts "set editor '#{@conf[:editor]}'"
```

```
6  end
```

５行目の puts を return に変えると，テストが通る．この改良により，my_help 本体に影響が出た．lib/my_help/my_help.rb を動かした入出力結果を以下に記す.

```
< bundle exec exe/my_help set_editor emacs
  my_help called with 'emacs'
```

これは正常に動いていない事を示し，実際返ってくるはずの出力結果set editor 'emacs' が返ってきていない．先程，lib/my_help/my_help_controll.rb の Set_Editor メソッドを改良した事が原因である．lib/my_help/my_help.rb にも lib/my_help/my_help_controll.rb にも出力メソッド (puts) が無い為，上記のような出力結果になる．そこで，lib/my_help/my_help.rb の Set_Editor メソッドに puts を加え，出力できるように改良した．以上より，my_help 本体も正常に動くようになった．最終的な Set_Editor に関する my_help の改良されたコードを以下に記す.

- lib/my_help/my_help.rb の Set_Editor メソッドのコード

```
1  def set_editor(editor)
2      @conf[:editor] = editor
3      conf = { editor: editor }
4      File.open(@conf[:conf_file], "w") { |f| YAML.dump(conf, f) }
5      return "set editor '#{@conf[:editor]}'"
6  end
```

- lib/my_help/my_help_controll.rb の Set_Editor メソッドのコード

```
1  def set_editor(editor_name)
2      invoke :setup
3      puts $control.set_editor(editor_name)
4  end
```

これらの過程から，my_help の書き換えを行ったので，西谷に上記の改良点の pull_request をかけた.

この節での作業をまとめると，Set_Editor メソッドに関する改良として，

- test/my_help_test.rb に Set_Editor メソッドのテスト作成
- lib/my_help/my_help.rb の Set_Editor メソッドの書き換え
- lib/my_help/my_help_controll.rb の Set_Editor メソッドの書き換え

を行った.

5.3 Edit メソッドの改良

次に Edit メソッドのテスト作成に移る.

5.3.1 テスト作成の為の準備内容

lib/my_help/my_help.rb での Thor で書かれた Edit メソッドのコードは以下の通りである.

```
1  desc "edit HELP", "edit HELP"
2  def edit(help_name)
3    invoke :setup
4    $control.edit_help(help_name)
5  end
```

my_help の Edit コマンドを入力した時の入出力結果を以下に記す.

```
> my_help edit [help_name]
```

help_name の部分が my_help_test であれば，my_help_test の emacs が開く仕組みである. この場合のテストコードについて，西谷と相談した所，emacs が開くようなテストを作成する事は，現時点の私の知識では困難であるという結論が出た．その為，Edit メソッドのテストでは,

- テストを見送る理由を出力するテスト作成
- lib/my_help/my_help_controll.rb の Edit_Help メソッド内にある

  local_help_entries.member?が一体何を返すかのテスト作成

を行う事にした.

5.3.2 最終テストコード

以下の 2 つのテストコードは，Unit Test の典型的な記述である．先ず 1 つ目の，テストを見送る理由を出力するテストコードは最終的に以下のようになった．

```
 1  sub_test_case "Edit Help" do
 2      test "edit_help" do
 3        puts "
 4        システムコールが呼ばれているのでテストできません.
 5        edit_help では，thor で my_help_test.org などが呼ばれたときに,
 6        呼ばれるものである.
 7        そうすると target_help にいき,
 8        それが local_help_entries.member かを見て,
 9        member の時にはシステムコールがされる.
10        ところがこれを test することはできない為，今は先送りしています.
11        "
12        end
13    end
```

コード内容は，puts で上記の理由を出力しているだけである．

次に lib/my_help/my_help_controll.rb の Edit_Help メソッド内にある local_help_entries.member?が一体何を返すかのテスト作成における途中経過と最終コードを以下に記す．先ず，Local_Help_Entries のテストを作成した．

```
1  sub_test_case "Local Help Entries" do
2      test "edit_help" do
3        conf_path = File.join(Dir.pwd, "test")
4        puts Control.new(conf_path).edit_help("my_help_test")
5      end
6    end
```

テストコードの内容を以下で詳しく説明する．1 行目，2 行目，3 行目は Set_Editor メソッドのテストコードの 1 行目，2 行目，4 行目と同様の意味である為，省略する．4 行

目の Controll.new(conf_path).edit_help(”my_help_test”) のコード説明を以下に記す．Controll.new(conf_path) までは Set_Editor メソッドの 5 行目の第二引数部分と同様の意味である．今回の場合は，文字列 my_help_test を引数として，lib/my_help/my_help.rb の edit_help が呼ばれている．上記のコードを rake test してみた所，

```
Local Help Entries:
    test: local_help_entries: "/Users/ooterahana/
          my_help/test/.my_help/my_help_test.org"
  file /Users/ooterahana/my_help/test/.my_help/
  my_help_test.org does not exits in
  /Users/ooterahana/my_help/test/.my_help.
  make my_help_test first by ’new’ command.
```

と返ってきた．これは，my_help_test を作りなさいとの提案である．test/.my_help の中に，my_help_test がなかった為である．my_help_test を新たに作る必要があるので，先ず，my_help の New コマンドに関する lib/my_help/my_help_controll.rb の Init_Help メソッドが動くかどうかのテストを作成する．そこで，Init_Help メソッドのテストを追加した Local_Help_Entries のテストコードを以下に示す．

```
 1  sub_test_case "Local Help Entries" do
 2        test "init_help" do
 3          conf_path = File.join(Dir.pwd, "test")
 4          puts Control.new(conf_path).init_help("my_help_test")
 5        end
 6        test "edit_help" do
 7          conf_path = File.join(Dir.pwd, "test")
 8          puts Control.new(conf_path).edit_help("my_help_test")
 9        end
10      end
```

Init_Help のテストコード内容は，Edit_Help のテストコード内容と同様の意味である．4 行目で，lib/my_help/my_help_controll.rb の Init_Help メソッドが呼ばれている点だけは異

なる．これで rake test すると，my_help_test という help が作成され，emacs が開いた．正常に Init_Help メソッドが動いた為，my_help_test という help が新しく作成できたという事だ．

次に Init_Help の他の場合のテストとして，my_help_test ではなく，nil(何もなし) の場合で呼ぶテストを追加した．以下に，Local_Help_Entries の最終的なテストコードを示す．

```
 1  sub_test_case "Local Help Entries" do
 2        test "init_help" do
 3          conf_path = File.join(Dir.pwd, "test")
 4          puts Control.new(conf_path).init_help("my_help_test")
 5        end
 6        test "edit_help" do
 7          conf_path = File.join(Dir.pwd, "test")
 8          puts Control.new(conf_path).edit_help("my_help_test")
 9        end
10        test "init_help without argument(引数)" do
11          conf_path = File.join(Dir.pwd, "test")
12          puts Control.new(conf_path).init_help()
13        end
14      end
```

追加コードの Init_Help without argument(引数) のテストコード内容は Init_Help のテストコード内容と同様の意味である．Init_Help の引数には何も入れない．

5.3.3 my_help の改良と pull_request

これらの過程から，テスト実行におけるエラー内容，lib/my_help/my_help.rb のコードの改良点と pull_request 内容を以下に記す．上記で作成した Local_Help_Entries のテストを実行した結果，Edit_Help と Init_Help しかない状態で，追加した Init_Help without argument がないという内容のエラーが返ってきた．これは，lib/my_help/my_help_controll.rb

の Init_help メソッドのコードが間違えていた為だ．lib/my_help/my_help_controll.rb の Init_Help メソッドのコードを以下に記す．

```
 1  def init_help(file)
 2        if file.nil?
 3          puts "specify NAME".red
 4          exit
 5        end
 6        p target_help = File.join(@conf[:local_help_dir], file + ".org")
 7        if File::exists?(target_help)
 8          puts "File exists. delete it first to initialize it."
 9          exit
10        end
11        p template = File.join(@conf[:template_dir], "help_template.org")
12        FileUtils::Verbose.cp(template, target_help)
13      end
```

9 行目に書かれている通り，exit で終わってしまっている事が問題であった．

> exit とはプログラムを終了させるために使います [5].
>
> また return も同じく処理を終了しますが，return の場合はメソッドの中で戻り値が呼び出されたところでプログラムが終了となりますが，exit の場合は呼び出された時点でプログラム終了となる点で違いがあります [5].

Init_Help メソッドではプログラム終了の為に exit を用いていた．呼び出されたその時点でプログラム終了となる事から，Init_Help without argument がないというエラーが出たと考えられる．そこで，Init_Help メソッドの 2 つの exit をコメントアウトし，その代わりに，return を追加した．しかし，また別のエラーが返ってきた．エラーは，file.nil だとその時点でプログラムが止まるという内容である．def init_help(file) の引数部分を file.nil に変える事でテストが通るようになった．改良した lib/my_help/my_help_controll.rb の Init_Help メソッドの最終コードを以下に記す．

```
 1  def init_help(file=nil)
 2      if file.nil?
 3          puts "specify NAME".red
 4          #exit
 5          return
 6      end
 7
 8      p target_help = File.join(@conf[:local_help_dir], file + ".org")
 9      if File::exists?(target_help)
10          puts "File exists. delete it first to initialize it."
11          #exit
12          return
13      end
14      p template = File.join(@conf[:template_dir], "help_template.org")
15      FileUtils::Verbose.cp(template, target_help)
16     end
```

test: init_help without argument(引数) の部分のエラーが specify NAME となって出力されている為，成功している．また my_help の書き換えを行ったので，西谷に上記の改良点の pull_request をかけた．

この節での作業をまとめると，Edit メソッドに関する改良として，

- test/my_help_test.rb に Local_Help_Entries メソッドのテスト作成 (lib/my_help_ controll.rb の edit_help メソッドの local_help_entries.member?が，一体何を返すかのテスト作成)
  - edit_help
  - init_help
  - init_help without argumaent(引数)
- lib/my_help/my_help_controll.rb の Init_Help メソッドの書き換え

を行った．

5.4 List メソッドの改良

5.4.1 テスト作成の為の準備内容

lib/my_help/my_help.rb での Thor で書かれた List メソッドのコードは以下の通りである．

```
 1  desc 'list [HELP] [ITEM]', 'list all helps, specific HELP, or ITEM'
 2  def list(*args)
 3     invoke :setup
 4     file = args[0]
 5     item = args[1]
 6     if file.nil?
 7       puts $control.list_all.blue # list []
 8     elsif item.nil?
 9       begin
10         puts $control.list_help(file) # list [file]
11       rescue StandardError => e
12         puts e.to_s.red
13       end
14     else
15       begin
16         puts $control.show_item(file, item) # list [file] [item]
17       rescue StandardError => e
18         puts e.to_s.red
19       end
20     end
21   end
```

4 行目の file と 5 行目の item には my_help 起動時の引数 (args) が入っている．my_help の List コマンドを使用する時には，ターミナルで，my_help list [file] [item] と入力する．こ

の時の file と item に何を入れるかによって，出力される結果が異なる．List メソッドのテストは，それぞれに対して正常に動いているかのテストを作成する為に，

- file 無し item 無しの場合
- file 有り item 無しの場合
- file 有り item 有りの場合

の 3 つの場合に分けた．

1. file 無し item 無しの場合

   以下のコードは，5.4.1 に記載してある lib/my_help/my_help.rb コードの 6 行目から 7 行目部分にあたる．

   ```
   1  if file.nil?
   2  puts $control.list_all.blue # list []
   ```

   この場合は，my_help list (nil) (nil) を意味しており，その時には，2 行目の Control.new.list_all が呼ばれる．my_help での出力では colorize 機能が付加されているが，テストを書く時に，カラーでの出力は重要視していない為，colorize 機能を付加していない．この場合に，List コマンドを使用した時の入出力結果は以下の通りである．

   ```
   > my_help list
   List all helps
         ruby: - ruby
          org: - emacs org-mode の help
         todo: - my todo
   my_help_test: - my_help_test
   sample_help: - ヘルプのサンプル雛形
        emacs: - Emacs key bind
     new_help: - ヘルプのサンプル雛形
    new_help1: - ヘルプのサンプル雛形
   ```

2. file 有り item 無しの場合

以下のコードは，5.4.1 に記載してある lib/my_help/my_help.rb コードの 8 行目から 13 行目部分にあたる．

```
1  elsif item.nil?
2      begin
3        puts $control.list_help(file) # list [file]
4      rescue StandardError => e
5        puts e.to_s.red
6      end
```

この場合は，my_help list (ruby) (nil) を意味しており，その時には，3 行目の Control.new.list_help('ruby') が呼ばれる．今回は file 部分に ruby を用いているが，上記の list all helps 内にある file のどれか 1 つをテストすれば良い．1 つのテストが通れば，残りのテストも通るからである．この場合に，List コマンドを使用した際の入出力結果は以下の通りである．

```
1  < my_help list ruby
2  - ruby
3        , head            : head
4        , license         : license
5      -p, puts_%          : puts_%
```

3. file 有り item 有りの場合

以下のコードは，5.4.1 に記載してある lib/my_help/my_help.rb コードの 8 行目から 13 行目部分にあたる．

```
1  else
2      begin
3        puts $control.show_item(file, item) # list [file] [item]
4      rescue StandardError => e
```

```
5       puts e.to_s.red
6     end
```

この場合は，my_help list (todo) (-d=”./test”) を意味しており，その時には，Control.new.show_item(’todo’,’-d=”./test”’) が呼ばれる．今回も同様に，file 部分に todo を用いているが，上記の list all helps 内にある file のどれか１つをテストすれば良い．item 部分には，-d=”./test”を用いているが，この部分には，ターミナル上で my_help list todo とした際に出力される中からどれか１つを選んでテストすれば良い．しかし，-d オプションが省かれると，それぞれの環境の’~/.my_help’ に置かれた help.org が読み込まれる．そこで，-d に’./test’ と指定する事で，環境に依存せずにあらかじめ用意したディレクトリ’my_help/test/.my_help’ を参照するようになる．この場合に，List コマンドを使用した際の入出力結果は以下の通りである．

```
1  < my_help list todo -d="./test"
2  - my todo
3  -----
4  daily
5  - ご飯を食べる
6  - 10 時には寝床へ入る
```

以上の 1.2.3 を参照しながらテストを作成した．

5.4.2 最終テストコード

最終的なテストコードは以下のようになった．

```
1  sub_test_case "List" do
2   test "List all" do
3     expected = /List all helps/
4     conf_path = File.join(Dir.pwd, "test")
5     assert_match expected, Control.new(conf_path).list_all
6    end
```

```
 7
 8   test "List 'ruby' returns RuntimeError" do
 9     conf_path = File.join(Dir.pwd, "test")
10     assert_raise WrongFileName do
11       Control.new(conf_path).list_help("ruby")
12     end
13   end
14
15   test "List 'todo','-d'" do
16     expected = /^- my todo/
17     conf_path = File.join(Dir.pwd, "test")
18     assert_match expected, Control.new(conf_path).show_item("todo", "-d")
19   end
20 end
```

これは，Unit Test の典型的な記述である．テストコードの内容を 3 つの場合に分けて，以下で詳しく説明する．

1. file 無し item 無しの場合 (List all のテスト)

   この場合のテストは，5.4.1 の 1 のテストである．以下のコードは，上記の test/my_help_test.rb コードの 2 行目から 6 行目部分にあたる．

   ```
   1  test "List all" do
   2    expected = /List all helps/
   3    conf_path = File.join(Dir.pwd, "test")
   4    assert_match expected, Control.new(conf_path).list_all
   5  end
   ```

   2 行目は，変数 expected に/List all helps/を代入している．/List all helps/は，ターミナル上で my_help list と入力した時の出力結果 (文字列の集合) を正規表現 (1 つの文字列で表現する方法) を用いて表したコードである．4 行目の assert_match は,「第

一引数の正規表現の値と第二引数の値がマッチすれば成功になるメソッド」である[6]．このメソッドで，先程の期待値 (expected) と，lib/my_help_controll.rb の list_all メソッドを呼ぶ新しいオブジェクト Controll がマッチするかを調べている．

2. file 有り item 無しの場合 (List 'ruby' returns RuntimeError のテスト)

   この場合のテストは，5.4.1 の 2 のテストである．以下のコードは，上記の test/my_help_test.rb コードの 8 行目から 13 行目部分にあたる．

```
1  test "List 'ruby' returns RuntimeError" do
2        conf_path = File.join(Dir.pwd, "test")
3        assert_raise WrongFileName do
4     Control.new(conf_path).list_help("ruby")
5        end
6  end
```

   3 行目の assert_raise の役割は以下の通りである．「Passes if the block raises one of the expected exceptions[7].」つまり，3 行目は，WrongFileName が予想される例外の 1 つを発生させた場合に合格するという事を実行するように宣言している．

3. file 有り item 有りの場合 (List 'todo','-d="./test"' のテスト)

   この場合のテストは，5.4.1 の 3 のテストである．以下のコードは，上記の test/my_help_test.rb コードの 15 行目から 19 行目部分にあたる．

```
1  test "List 'todo','-d'" do
2   expected = /^- my todo/
3   conf_path = File.join(Dir.pwd, "test")
4   assert_match expected, Control.new(conf_path).show_item("todo", "-d")
5  end
```

   上記のコード内容は，1 と同様の為，省略する．

5.4.3 my_help の改良と pull_request

テスト実行におけるエラーと pull_request を記す．List メソッドに関する my_help 本体の変更は，西谷が行った．最初に書いたテストコードは次の通りであった．

```
1  test "List 'ruby'" do
2          conf_path = File.join(Dir.pwd, "test")
3          assert_raise NameError do
4           Control.new(conf_path).list_help("ruby")
5          end
6        end
```

assert_raise のサンプルに書かれていた NameError をそのまま残してテストを走らせた所，以下のエラーが返ってきた．

```
<NameError> expected but was
<MyHelp::Control::WrongFileName(<No help named 'ruby' in the directory ''.>)
```

このエラーは,「NameError が期待されたが、my_help_controll.rb では WrongFileName（ディレクトリに'ruby' という名前のヘルプはない.）が返って来ました.」というメッセージである．これらのコードで問題となる例外クラスの意図は次の通りである．まず，NameError は,「未定義のローカル変数や定数を参照した場合 [8]」に定義する例外クラスである．次に，RuntimeError は,「特定の例外クラスには該当しないエラーが発生した場合や例外クラスを省略した raise の呼び出し [8]」と厳密には解説されているが，もっと大雑把に説明すると，以下のような説明となる．

> ほとんどのプログラム言語では，アプリケーションを走らせているとき (runtime) あるいは実行中 (execution) の間に起こる例外を runtime error とゆるく定義しています．これは定義としてはあまり有益ではないけれど，実行する前から言語のコンパイラで捕捉される CompileError などの他の例外のカテゴリと簡単に区別することができます (和訳)[9].

Ruby の NameError は CompileError と同様にコードの実行前に判明するエラーである．実行中に間違った File name を指定した時のエラーを意図している WrongFileName は，例

外クラスを my_help 内で独自に定義した例外クラスである．lib/my_help/my_help_controll .rb で定義しているコードは以下の通りである．

```
WrongFileName = Class.new(RuntimeError)
```

であったが，置き場所の階層が適切ではなかった為，当初 NameError が出ていたと思われる．エラーメッセージの原因は，期待されるエラーコードが NameError になっている事であった．これは，上記の NameError の説明通り，一般的な変数が定義されていない時に返るエラーコードである．そこで，例外クラスで定義した WrongFileName と書き換えてテストが通った．

第6章　総括

本研究では，my_help の発展に向け，以下の開発を行った．

- Unit Test を用いた test/my_help_test.rb に Version メソッド，Set_Editor メソッド，Local_Help_Entries メソッド，及び List メソッドのテスト作成
- lib/my_help/my_help.rb の Version メソッド及び Set_Editor メソッドの改良
- lib/my_help/my_help_controll.rb の Set_Editor メソッド及び Init_Help メソッドの改良

以上の通り，Unit Test での my_help のテストを作成した為，今後，my_help に新たな機能が追加された場合でも，my_help が想定通りに動いている事を，このテストで確認する事ができる．

開発にあたって，知っておく必要があると考える点を以下に 2 つ挙げる．先ず，テスト作成にあたって，my_help の理解を深める為，実際にコマンド実行する必要がある．コマンド実行には，Ruby の知識が必要となる．その為，事前に習得しておく必要がある．次に，本研究のテスト作成に用いた Unit Test は，速く実行ができ，テスト失敗時の原因究明も非常に容易であるが，クラスやメソッドを単体でテストする手法である為，Unit Test のみではソフトウェアの品質全てを保証できないという事だ．以上 2 つの問題点を理解した上で，テスト作成を進める方が効率的であると考える．

最後に，今後の課題としては，今回テストを見送った new コマンドや git コマンドのテストの作成を進める事である．更に，List メソッドのテスト作成過程で課題となった例外処理関連については，raise(例外) への refactoring の解説が Refactoring Ruby の「10.13 エラーコードから例外へ」に記されている．今後，この方針に従って例外処理をまとめていく必要がある [10].

謝辞

本論文の作成にあたり，終始適切な助言と丁寧な指導をして下さった西谷教授に心より深く感謝致します．また，研究にあたり，様々な助言や知識の供給，精神的な支えを頂きました同研究室のメンバーおよび先輩方に深く感謝致します．誠にありがとうございました．

参考文献

[1] MyHelpGitHub, https://github.com/daddygongon/my_help, (accessd on 20 Jan 2022).

[2] ラス　オルセン, ”明解!Ruby-奥深い Ruby の文化を身に付けるテクニック”, (ピアソン桐原, 2012).

[3] 助田雅紀, ”Ruby を 256 倍使うための本”, (アスキー，2001).

[4] TDD, https://qiita.com/daddygongon/private/c367f2acb1a194489370, (accessd on 20 Jan 2022).

[5] Ruby の exit の使い方を現役エンジニアが解説 [初心者向け],
https://techacademy.jp/magazine/19931, (accessd on 20 Jan 2022).

[6] [Rails]assert_match の使い方,
https://qiita.com/GalaxyNeko/items/8982354dfd62fed0e58a, (accessd on 20 Jan 2022).

[7] Module:Test::Unit::Assertions,
https://www.rubydoc.info/github/test-unit/test-unit/Test/Unit/Assertions, (accessd on 20 Jan 2022).

[8] Ruby の例外処理についてまとめ,
https://qiita.com/nyan_tech_24/items/337d19500c56cb905112, (accessd on 20 Jan 2022).

[9] Ruby Exception Handling:RuntimeError,
https://airbrake.io/blog/ruby-exception-handling/runtimeerror, (accessd on 20 Jan 2022).

[10] ジェイ・フィールズ他, ”リファクタリング:Ruby エディション”, (復刊ドットコム，2020).