Iteration 2 and Code Coverage

Fantasy Football Analyst

Joey Bauer, Ben Chang, Dylan Fiedler, Isaac Hoffmann, Alejandro Puente, Jake Trotman

Write a short summary for iteration 2. Your summary should address at least the following issues:

1. **What were the main difficulties so far?**

For iteration 2 our group faced difficulties in parallel programing. Our version control repo became disorganized and took significant time and effort to correct. We have started to implement more consistent methods in terms of creating branches, when to commit and push new features, and maintaining a consistently working branch that does not have any errors in order to avoid these issues.

Our original scrapers have provided us with sufficient data to test with. However, they still experience some issues in regards to pulling specific ESPN pages and returning valid data. We also need to add additional scrapers for more detailed data about players and leagues, without disrupting our current ones. We have implemented tests to verify that our scraped data has updated and MySql schema to handle the additional data that we will pull for later iterations.

Our team has also been working in different local environments (Windows vs. OS. vs. CS Lab Machines) and we often run into issues building new commits on different OS's. We have updated the README with additional details as to how to run the project within Eclipse, via command line, and any additional steps you may need to do so depending on your environment.

Furthermore, while we have successfully implemented various JUnit tests we need to continue adding to our test suite that tests additional features as they are added. We have implemented a new policy to always create tests for whatever feature you build before committing those features to the git branch. By creating both the enhancement and the respective tests, it is less likely that mistakes will be made. Additionally, by doing so other team members can run the new tests before making additional ones.

We also are using multiple tools, languages, and dependencies that not all team members are familiar with. We have focused on spending time sharing with each other what has been implemented, how it works, and how it may be useful to others for what they are working on. We have also started recording "template code" for certain features (for example, angularjs forms and API calls, basic queries for scrapers with JSoup) so that others have a reference to build from.

Overall, maintaining communication as to who is doing what and when it needs to be done has been the biggest challenge we have experienced. Often times all of our group members cannot meet at the same time so when substantial progress has been made, we must update team members via Slack, finding times for additional meetings, and through our git repo so that everyone is on the same page. Now that we have defined who is responsible for what and set a system in place for how and when commits are made to the git repo we have resolved a lot of these issues.

2. **Were there any features you did not implement as planned, and why? Are you pushing some features to later iterations, and if so, why?**

We pushed a couple of our low priority tools to the last iteration. These tools include *League Polls*, *Waiver Wire Assistant,* and *Add a League.* We spent significant time this iteration cleaning up and finishing tasks originally assigned to iteration one. We also initially planned to do the majority of our testing on the third iteration. However, given the requirements for running code coverage we had to pivot and implement testing for iteration two. The tools we are pushing to the last iteration are considered less of a priority and will be balanced with cleaning up the current code before submitting for testing.

We've currently implemented 6 of the 9 tools: Fair Trade Judge, Weekly Scores, Composite Rankings, Draft Buddy, Smart Rankings, and Insult Generator. These tools are accurately pulling data and have main functionality implemented, however, we will continue to enhance them with future iterations. For these tools we implemented numerous tests, but will continue to do so the more we develop and further we get in our project. Also, our scrapers are pulling sufficient data for all our tools and purposes but we plan to expand upon them in future iterations.

Lastly, given the extensive amount of backend work, our front end user page is not as clean and refined as we would like. We intend on spending substantial time cleaning up our views and html code once we are certain all the functionality for our tools and streams is reliable and consistent.

3. **What tests did you prepare for this iteration, and what are they covering? Did the tests you wrote deviate from your plan? What features are you not testing yet? Did you use any test frameworks, such as JUnit, the Android Monkey, Selenium, etc.?**

We used the JUnit testing framework to run unit test on our Java code. We wrote sanity checks on our data as well as check the functionality of our tools. JUnit is already built into Eclipse, thus it did not take us any additional time to set up and we were able to start building tests right away. It is also easy to implement JUnit test cases while developing the specific tool being built. This allows us to make sure every bit of our code is being tested before any other testing is performed or git updates are made. Additionally, JUnit can easily verify any new functionality did not break any previous code. As we add to our test suite, we can confirm that our new features and code does not break any previous JUnit tests. Lastly, one of the largest benefits of JUnit testing is that it is very easy for other team members to read and understand the tests performed in each test. It also allows us to create a comprehensive test suite by using the old tests as templates.

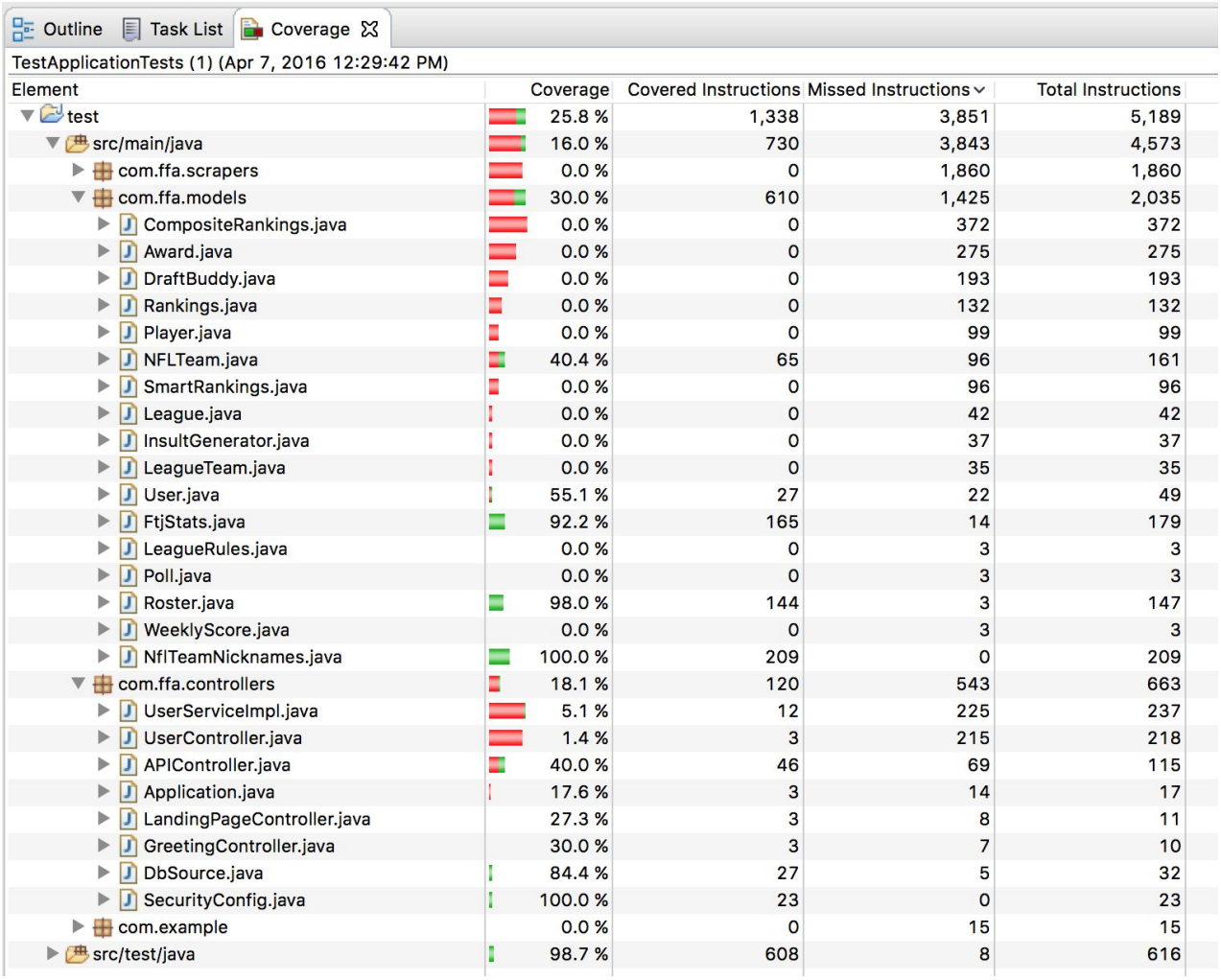
Using JUnit also has numerous disadvantages. Some include in inability to do dependency testing with JUnit. This has not been an issue yet, but we can see it being an issue in the final project. Also, our tests have started to become somewhat redundant. We plan to modify our test suite to make it as efficient and complete as possible. Lastly, JUnit does not test our HTML. To account for this we plan on using Selenium at the end of iteration 3 once we have completed our fully developed user interface.

4. **Give a sample of code coverage tool output both before and after adding to your test suite. What did you learn from the code coverage data? How did you use this information to expand your test suite and improve coverage?**

Below are examples of two runs using a code coverage tool. After running our test suite using the code coverage tool we realized we were missing significant tests for our applications

controllers. After adding additional tests to our suite we were able to improve our results by almost one hundred perecent. We have determined that our models contain our most essential business logic so we directed most of our tests to those classes. Overall the code coverage tool also allows us to ensure that our critical functions are working properly through each update and iteration.

Code Coverage First Attempt:

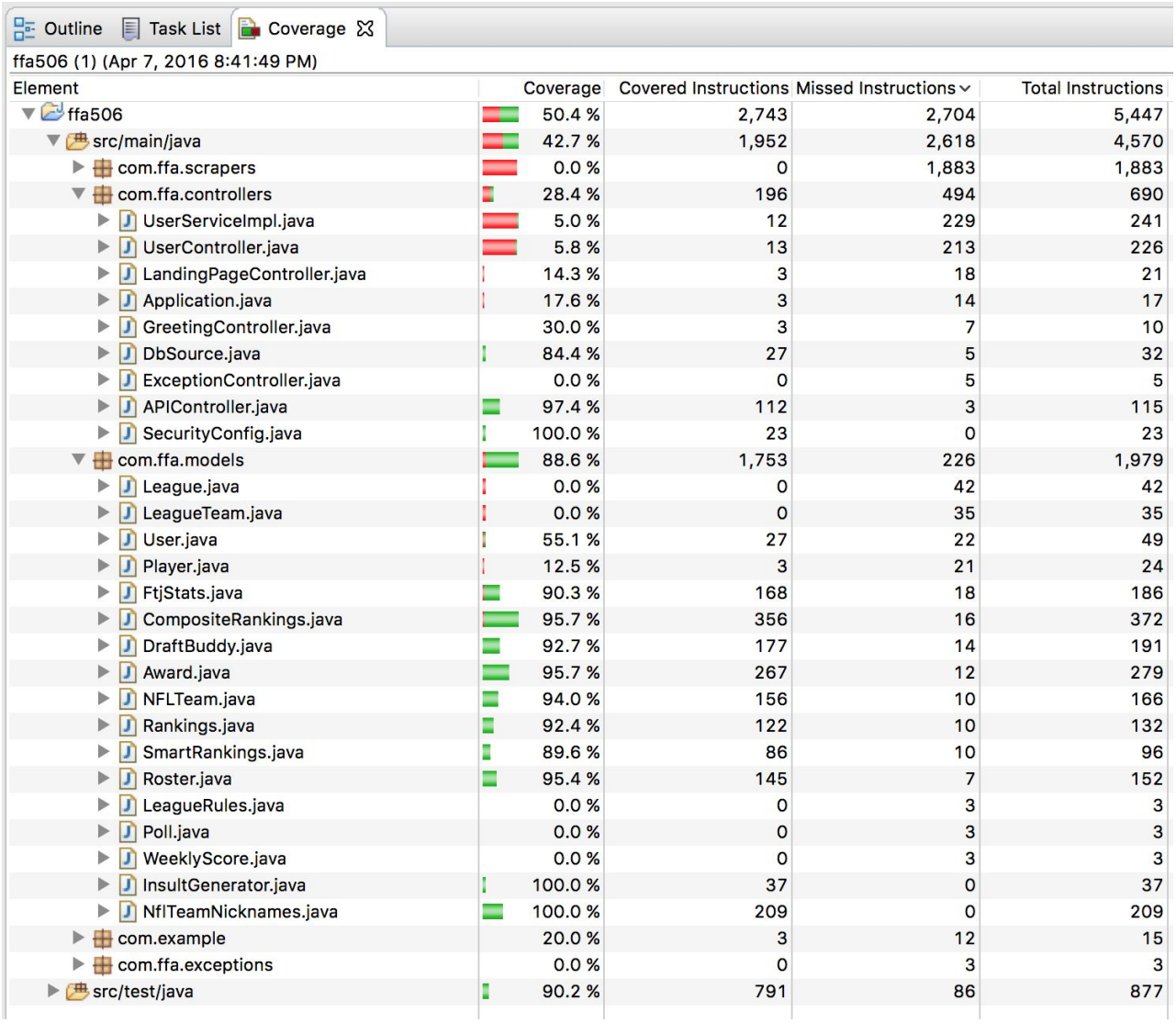


Outline | Task List | Coverage

TestApplicationTests (1) (Apr 7, 2016 12:29:42 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
test	25.8 %	1,338	3,851	5,189
src/main/java	16.0 %	730	3,843	4,573
com.ffa.scrapers	0.0 %	0	1,860	1,860
com.ffa.models	30.0 %	610	1,425	2,035
CompositeRankings.java	0.0 %	0	372	372
Award.java	0.0 %	0	275	275
DraftBuddy.java	0.0 %	0	193	193
Rankings.java	0.0 %	0	132	132
Player.java	0.0 %	0	99	99
NFLTeam.java	40.4 %	65	96	161
SmartRankings.java	0.0 %	0	96	96
League.java	0.0 %	0	42	42
InsultGenerator.java	0.0 %	0	37	37
LeagueTeam.java	0.0 %	0	35	35
User.java	55.1 %	27	22	49
FtjStats.java	92.2 %	165	14	179
LeagueRules.java	0.0 %	0	3	3
Poll.java	0.0 %	0	3	3
Roster.java	98.0 %	144	3	147
WeeklyScore.java	0.0 %	0	3	3
NflTeamNicknames.java	100.0 %	209	0	209
com.ffa.controllers	18.1 %	120	543	663
UserServiceImpl.java	5.1 %	12	225	237
UserController.java	1.4 %	3	215	218
APIController.java	40.0 %	46	69	115
Application.java	17.6 %	3	14	17
LandingPageController.java	27.3 %	3	8	11
GreetingController.java	30.0 %	3	7	10
DbSource.java	84.4 %	27	5	32
SecurityConfig.java	100.0 %	23	0	23
com.example	0.0 %	0	15	15
src/test/java	98.7 %	608	8	616

Code Coverage Second Attempt:



Outline | Task List | Coverage

ffa506 (1) (Apr 7, 2016 8:41:49 PM)

Element	Coverage	Covered Instructions	Missed Instructions	Total Instructions
ffa506	50.4 %	2,743	2,704	5,447
src/main/java	42.7 %	1,952	2,618	4,570
com.ffa.scrapers	0.0 %	0	1,883	1,883
com.ffa.controllers	28.4 %	196	494	690
UserServiceImpl.java	5.0 %	12	229	241
UserController.java	5.8 %	13	213	226
LandingPageController.java	14.3 %	3	18	21
Application.java	17.6 %	3	14	17
GreetingController.java	30.0 %	3	7	10
DbSource.java	84.4 %	27	5	32
ExceptionController.java	0.0 %	0	5	5
APIController.java	97.4 %	112	3	115
SecurityConfig.java	100.0 %	23	0	23
com.ffa.models	88.6 %	1,753	226	1,979
League.java	0.0 %	0	42	42
LeagueTeam.java	0.0 %	0	35	35
User.java	55.1 %	27	22	49
Player.java	12.5 %	3	21	24
FtjStats.java	90.3 %	168	18	186
CompositeRankings.java	95.7 %	356	16	372
DraftBuddy.java	92.7 %	177	14	191
Award.java	95.7 %	267	12	279
NFLTeam.java	94.0 %	156	10	166
Rankings.java	92.4 %	122	10	132
SmartRankings.java	89.6 %	86	10	96
Roster.java	95.4 %	145	7	152
LeagueRules.java	0.0 %	0	3	3
Poll.java	0.0 %	0	3	3
WeeklyScore.java	0.0 %	0	3	3
InsultGenerator.java	100.0 %	37	0	37
NflTeamNicknames.java	100.0 %	209	0	209
com.example	20.0 %	3	12	15
com.ffa.exceptions	0.0 %	0	3	3
src/test/java	90.2 %	791	86	877

5. **Optionally give a URL and instructions for using your application in the current stage. This makes sense for purely web-based projects, but it may be impractical for projects that must be installed on a client device.**

Tag a branch or revision in your repository to identify the code you want to submit for iteration 2. The date and time of this revision should be *before* the deadline. This branch should include a README file with detailed instructions on how to build and run the application and tests.

Dashboard URL: http://ffa506-env.us-west-2.elasticbeanstalk.com/dashboard

1- Main page of dashboard displays tools available for use

2- *Fair Trade Judge*

- Select two teams one from each drop down menu
- Select two players using the two option lists
- Click Compare Players
- Output of trade result will appear

3- *Awards*
- Displays weekly awards
- Use drop down selection to pick week

4- *Draft Buddy*
- Shows current players available in draft
- Implementation coming in next iteration (need to update scrapers to pull rankings)

5- *Add A League*
- Implementation coming in next iteration

6- *Insult Generator*
- Select a team to insult from drop down selection
- Returns an insult based on team (currently dummy data)

7- *League Poll*
- Implementation coming in next iteration

8- *Smart Rankings*
- Displays "smart" (computer generated rankings) based on week
- Change week using drop down selection

9- C*omposite Rankings*
- Ranks opponents on results based on points each week
- Select week using drop down selection

Signup URL: http://ffa506-env.us-west-2.elasticbeanstalk.com
- Frontend for signup and a login
- Implementation to be finalized in next iteration (currently 90% of the way done)

See README.md for detailed instructions on building application and running JUnit tests, and Code Coverage Tools

Instructions to Checkout and Launch

Checkout repo:

Clone the repo at: https://github.com/trotman23/ffa506.git

Launch app:

Mac OSX Terminal:

1. install homebrew: http://brew.sh/
2. open terminal and run 'brew install maven'
3. Git checkout Iteration2 branch
4. In terminal go to project root directory and run 'mvn spring-boot:run'
5. Open browser and go to localhost:8080/dashboard for main dashboard page

Via Eclipse:

1. import project into eclipse (file->import->git and add the above link to path)
2. Checkout Iteration2 branch
3. Right click project directory and select Run As..->Maven Build… (notice the …)
4. In Goals add "spring-boot:run" and select apply
5. Open browser and go to localhost:8080/dashboard for main dashboard page