Cyclomatic Complexity

We tried to explore different tools to calculate cyclomatic complexity, but were unable to integrate it easily. We decided to calculate the cyclomatic complexities of the different methods in the modern way:

> Assign one point to account for the start of the method
> Add one point for each conditional construct, such as an "if" condition
> Add one point for each iterative structure
> Add one point for each case or default block in a switch statement
> Add one point for any additional boolean condition, such as the use of && or ||
> With exceptions, you can add each throws, throw, catch or finally block as a single point when calculating the McCabe cyclomatic complexity metric.

The ideal McCabe complexity is 4 or 5, but 1-10 is considered well-structured and highly testable. For the sake of brevity, we will include the calculations for the Database file, as well as the number of methods with complexity over 10, and the percentage of that file's methods that are low complexity.

Overall, there were 5 complex and 2 very complex methods. This was due to list management and different cases that we had to explore. We could have split the method up further to reduce complexity, but we found the increased complexity to still be very readable and logical, so we let those methods have their higher complexity. The project has 86.5% low complexity, with an average cyclomatic complexity of 4.40 per method, which is an indicator of well-structured code with high testability.

Java Files

Database: 1 complex, 2 very complex. 70% low complexity.

checkUser(String): Boolean	3 (start of the method, if statement, catch)
getPasswordInfo(String username): String[]	4 (start, if, while, catch)
getUserID(String username): Int	3 (start, if, catch)
createUser(String username, String passwordHash, String salt): Boolean	3 (start, if, catch)
getLists(int userID, String listname): ArrayList<Info>	12 (start, 8 if/elseif, 2 while, catch)
addToList(int userID, Boolean isRecipe, String	21 (start, 19 if/elseif, catch)

listname, Info i): Boolean	
removeFromList(int userID, Boolean isRecipe, String listname, Info i): Boolean	21 (start, 19 if/elseif, catch)
updateLists(int userID, Boolean add, String listname, Info i): Boolean	3 (start, if, &&)
getPrevSearch(int userID): ArrayList<Searches>	3 (start, while, catch)
addPrevSearch(int userID, String testSearch, int radius, int results): Boolean	3 (start, if, catch

ListServlet: 1 complex. 50% low complexity.

doGet(HttpServletRequest request, HttpServletResponse response): void	8 (start, throws, 2 if, for, 3 &&)
doPost(HttpServletRequest request, HttpServletResponse response): void	16 (start, throws, 6 if/ifelse, 3 &&, 4 cases, catch)

LoginServlet: 1 complex. 0% low complexity

doPost(HttpServletRequest request, HttpServletResponse response): void	11 (start, throws, &&, 3 cases, 4 if, catch)

PrevSearchServlet: 0 complex. 100% low complexity.

doGet(HttpServletRequest request, HttpServletResponse response): void	3 (start, if, throws)

SearchServlet: 1 complex. 90% low complexity.

doGet(HttpServletRequest request, HttpServletResponse response): void	7 (start, 4 if, ||, throws)
getJSONResponse(String url): String	3 (start, while, catch)
recipeSearch(String query, int numResults, List<Info> doNotShowList, List<Info> favoritesList): ArrayList<RecipeInfo>	11 (start, 3 if/elseif, 4 for, 3 catch)
helperSort(Boolean isRecipe, ArrayList<Info> r, List<Info> favoritesList): ArrayList<Info>	3 (start, for, if)
restaurantSearch(String query, int numResults, int radius, List<Info> doNotShowList, List<Info> favoritesList): ArrayList<RestaurantInfo>	9 (start, 3 for, 3 if, while, catch)
getDistances (ArrayList<RestaurantInfo> restaurants): Map<RestaurantInfo, String>	3 (start, for, if)

getDriveTimes(ArrayList<RestaurantInfo> restaurants): void	3 (start, if, for)
helperDriveTime(String driveTimeURL, ArrayList<RestaurantInfo> restaurants): JsonArray	3 (start, if, for)
getPhoneAndURL(ArrayList<RestaurantInfo> restaurants): void	3 (start, for, catch)
getImageURLs(String query): ArrayList<String>	2 (start, for)

PasswordHashing: 0 complex. 100% low complexity

hashPassword(String password, String salt): String	2 (start, catch)
getRandomSalt(): String	2 (start, while)

Info: 0 complex. 100% low complexity

equals(Object other): Boolean	1 (start)

RecipeInfo: 0 complex. 100% low complexity

compareTo(RecipeInfo other): Int	1 (start)
equals(Object other): Boolean	3 (start, 2 if)

RestaurantInfo: 0 complex. 100% low complexity

compareTo(RestaurantInfo other): Int	1 (start)
equals(Object other): Boolean	3 (start, 2 if)

Searches: 0 complex. 100% low complexity

equals(Object other): Boolean	4 (start, if, 2 &&)

SortLists: 0 complex. 100% low complexity

sortAlphabetically(List<Info> list): List<Info>	2 (start, if)
sortByRating(List<Info> list): List<Info>	2 (start, if)

RatingComp (nested in SortLists): 0 complex. 100% low complexity

compare(Info i1, Info i2): Int	2 (start, if)

AlphabeticalComp (nested in SortLists): 0 complex. 100% low complexity

compare(Info i1, Info i2): Int	2 (start, if)

JSP Files

loginPage: 0 complex. 100% low complexity

sendData()	2 (start, if)
hexString(buffer)	1 (start)

listPage: 1 complex. 50% low complexity

(not inside a function)	13 (9 if, 4 for, ||)
setStoredItem(i)	1 (start)

searchPage: 0 complex. 100% low complexity

(not inside a function)	1 (for)
populateSearch()	2 (start, if)

signupPage: 0 complex. 100% low complexity

sendData()	3 (start, 2 if)
hexString(buffer)	1 (start)

JS Files

ListClient: 0 complex. 100% low complexity

addItem(listName, item)	1 (start)
removeItem(listName, item)	1 (start)
reorderResults(order, listName)	1 (start)
resetLists()	1 (start)
getList(listName)	1 (start)

loginChecker: 0 complex. 100% low complexity

checkLoggedIn()	3 (start, 2 if)

parseQuery: 0 complex. 100% low complexity

parseQuery(queryString)	3 (start, for, ||)

recipePage: 0 complex. 100% low complexity

(not inside a function)	5 (3 if, 2 for)

restaurantPage: 0 complex. 100% low complexity

(not inside a function)	3 (3 if)

resultPage: 0 complex. 100% low complexity

(not inside a function)	9 (4 if, 5 for)

All files not included have no methods or functionality that would fall under complexity.