SPRINT 2 DOCUMENT

Team E

Varshitha Bachu
Mitali Karmarkar
Arpita Khare
Tanuja Mohan
Maya Ravichandran
Prerana Reddy

Executive Summary

The Sprint 2 Document details the Scrum and XP Practices we used in Sprint 2 of this project. In this document, we describe and justify the different processes used in these methodologies and how we followed through with each of these methodologies. The first part of the document talks about what each scrum entails, including the roles, the documents, and the ceremonies. The 3 roles in the Scrum process are the scrum master, product owner, and team members. The 4 documents used in the processes are the product backlog, sprint backlog, task chart, and burndown chart. In this section, each document is accompanied by a justification of the information in each document. Evidence for each document is provided. The next section describes the 5 ceremonies in the scrum process: sprint planning, sprint, daily scrums, sprint review, and the retrospective. After the Scrum practices are addressed, the XP Practices we used are described and evidence is also given where applicable. The XP Practices that we used in this sprint were pair programming, simple design, sustainable pace, developing tests first, collective code ownership, refactoring, continuous acceptance and unit testing, and creating a test for every bug found. Proof for pair programming was provided using the selfies that were taken at each meeting. A simple design was justified through component, sequence, data-flow, and component diagrams. Sustainable pace was accounted for during sprint planning by narrowing down which tasks would be completed during this sprint and which would be carried out during sprint 3. We addressed developing tests first by making sure the unit tests and acceptance tests were completed first, and then the code was implemented by a pair of team members. The collective code ownership XP practice was fulfilled by making sure a pair of team members did not work on a part of the code they had already worked on. We reflected on our Sprint 1 designs and changed any designs based on the features planned to be implemented in Sprint 2 for refactoring. Continuous acceptance and unit testing was completed by running the unit and acceptance tests every time a new task was completed and before the next scrum. The creating a test for every bug found practice was completed every time a new bug was found. The screenshots and unit and acceptance test files are linked for that bug to show how it was found and fixed. All of these practices describe the processes used to complete Sprint 2 of Project 2.

Contents

1. Introduction

1.1 Purpose

The purpose of this Sprint 2 Document is to detail all the essential agile practices held and used while completing Sprint 2. This document also explains how we incorporated XP practices such as pair programming, simple design, sustainable pace, and developing tests first, collective code ownership, refactoring, continuous testing, and, finally, review and retrospective. We also provided proof that we completed seven scrums during the second sprint through GitHub links with all of our documents, pictures, and code. To justify simple design, we provided explanations for each of the designs describing what purpose each design fulfilled. We also provide evidence for all of the XP and Scrum practices in the form of pictures and time stamped GitHub commits.

1.2 Intended Audience

This product's intended audience is the CPs, TA, Professor Halfond, and anyone else who wants to interact with research paper information in a visual way. Anyone can use this product as long as they have an interest in creating a Word Cloud based on research paper.

1.3 Definitions

Term	Definition
CP	Course Producer
CSS	Cascading Style Sheets
Detailed Design Document or Design Document	Design Document previously created by Team 2
HTML	Hyper Text Markup Language
JavaScript	Programming language of HTML and the Web
PHP	Hypertext Preprocessor
PM	Project Manager
PMP	Project Management Plan
QA	Quality Assurance
SRS	Software Requirements Specification
TA	Teaching Assistant

1.4 References

https://en.wikipedia.org/wiki/Scrum_(software_development)
https://en.wikipedia.org/wiki/Sprint_(software_development)
https://blackboard.usc.edu/bbcswebdav/pid-4641165-dt-content-rid-13688743_2/courses/20171_csci_310_29967/Project%202%20-%20Requirements.pdf

2. Scrum

2.1 Roles

2.1.1 Scrum Master – Varshitha Bachu

Varshitha, our scrum master, was in charge of overseeing our scrum meetings. During each of our scrum meetings, Varshitha asked each pair of programmers one at a time the three questions: "What have you accomplished since the last meeting?", "Are there any obstacles in the way of meeting your goal?", and "What will you accomplish before the next meeting?". Varshitha kept track of these scrum meetings using Google documents. If a group expressed that there was an obstacle in their way, then Varshitha would ask them for more details about what they have already tried to overcome the obstacle. Accordingly, Varshitha would either spend some time helping the team find a new solution or come up with new resources that would help the team overcome the obstacle. Also, if the team needed help in clarifying what their task was, Varshitha would go to office hours to help the team get a better understanding of what needed to be done. Varshitha made sure that everyone had what they needed in order to successfully work on their respective parts of the project.

2.1.2 Product Owners – Stakeholders (CPs, TAs, and Professor Halfond)

Professor Halfond, our CSCI 310 CPs, and our 310 TAs were our product owners for this project and represented the stakeholders of this project. Professor Halfond was the one who gave us the requirements for this new project. These requirements went into our backlog. The team members (see below) would then attend the various office hours provided to clarify the requirements and to gather more detailed information on what the requirements encompassed. By speaking with the product owners, the team members were able to gather more information on the requirements in our backlog.

2.1.3 Team Members – Varshitha Bachu, Mitali Karmarkar, Arpita Khare, Tanuja Mohan, Maya Ravichandran, Prerana Reddy

Our team members have all been in charge of coding, designing, testing, and document writing. We have accomplished these four tasks by meeting in groups.

2.2 Four Documents

In Section 3.1 of the Design Document, we described our Data Flow Diagram. In this diagram, we demonstrated the flow of data throughout our system. In our implementation, we very closely followed this diagram. There are very few deviations from this diagram. The following describe each of the requests that are detailed in the Data Flow Diagram and how they are shown in our implementation. They are originally found in Design Document Sections 3.2.1 to 3.2.8.

2.2.1 Product Backlog

To create this product backlog, we took the 12 requirements provided by the Product Owners and prioritized them according to what our team thought was of highest priority and risk.

- At the top of the list we decided that the highest priority was getting the papers from the ACM and IEEE database and creating a word cloud from the data from those papers. We decided this was of highest priority because the whole Research Paper Word Cloud Generator revolved around creating a word cloud.
- The second requirement in the Product Backlog was lower priority than the first one because you need to be able to create the world cloud before any input can be given to the word cloud.
- The third requirement which discusses clicking on a word cloud is next in the product backlog because without words in the word cloud, a user cannot click on a word in the word cloud.
- The fourth requirement that we prioritized was the paper's abstract showing up with highlighted words. This requirement could only be done once the word cloud was created to be able to access the papers related to each word cloud. The initial page also has to be implemented before because the user has to input their author/keyword and then the word cloud has to be generated using ACM and IEEE papers. The third requirement also has to be implemented because in order to click on a paper's title, a list of papers has to show up.
- The fifth requirement has to do with authors. When an author is clicked, the author will then become the search criteria. For this to happen, the word cloud has to be completed and the

papers have to already be generated. This requirement comes after clicking on a paper's task because the paper's have more priority than the author's.

- The sixth requirement says that for each paper, there has to be links to download the paper and access its bibtex. This was prioritized after the requirements that discussed the list of papers and what happens when you click on a paper.
- The seventh requirement says the papers can be exported as PDFs and plain text. This requirement has lower priority than accessing the papers, making a paper clickable, and downloading the papers. The previous requirement has to do with one requirement whereas the seventh requirement deals with the whole list of papers.
- The eighth requirement is ordered here because for this requirement to be completed, the paper list and word cloud generation functionalities have to be completed. The priority is less than those requirements before it.
- The ninth requirement allows the user to download an image of the word cloud. As a group, we thought that the user would rather interact with the information about the papers and word cloud rather than download the image, thus it is ranked as ninth. This functionality could also be done by taking a screenshot, which the user could accomplish.
- The tenth requirement says when a conference is clicked, the other papers from that conference will be clicked. This functionality won't be as important as downloading papers or the word cloud image. Also, looking up other papers by authors is more informational and useful.
- The eleventh requirement allows the user to access previously entered searches. This requirement would act like an added feature for the user, which would help the user experience, but is not as important as the functionality of the search, word cloud, or paper list.
- The twelfth requirement shows a status bar between when the user searches for an author/keyword and the word cloud being generated. This requirement was listed as low priority because the status bar would only add to user experience, but wouldn't take away from any functionality of the Research Word Cloud Generator.
- The thirteenth requirement allows the user to sort alphabetically by author, title, and conference on both the paper list page and the conference page. This requirement has very low priority because sorting is a very small functional feature in comparison to the other ones. It is more important that we have the table correctly populated and then we can return back to sort the table alphabetically.
- The fourteenth requirement allows the user to see the frequency of the selected word for each paper shown in the paper list. This requirement has very low priority because we want to make sure that we are populating the table with the correct information and implementing other core functionality before coming back to this small requirement.
- The fifteenth requirement involves us, the team members, to update the simple design diagrams we made during sprint 1. Although, design is what drives our implementation, which may cause some to believe that this requirement should have had a higher priority, we saw this requirement in a different light. Because we made a very thorough design diagrams during sprint 1, which we adhered to fully, we had very minimal changes to our design diagrams. These changes only had to do with the user interface and not to do with any of the backend

functionality. Also, since our front end did not change during sprint 2, we did not have to make any updates to our design diagrams. We did make sure that all our diagrams were still up to date and accurate.

- The sixteenth requirement involves us, the team members, writing PHPunit tests for any white-box php code that we would be writing for this sprint. This requirement has very low priority because we had already written all the PHPunit tests during sprint 1 and we were confident that the test cases written for sprint 1 were thorough and correct for sprint 2 as well.
- The seventeenth requirement involves us, the team members, writing Behat tests for any black-box test cases for this sprint. This requirement has very low priority because we had already written all the Behat tests during sprint 1 and were confident that the test cases written for sprint 1 were thorough and correct for sprint 2 as well.
- GitHub link: https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/ProductBacklog.pdf

2.2.2 Sprint Backlog

- Reasoning for each requirement in the order displayed in the product backlog ('✓ ') indicates that it is in the sprint 2 log and ('✗ ') indicates that it did not make the sprint 2 log)
  - R1 (✓): This requirement made it into our sprint 2 backlog because setting the number for X allowed us to limit the number of papers we needed to search to gather words for our word cloud. Rather than searching through all the papers that are returned by the search, we are able to use this user inputted value of X to limit the papers we search through. Because this requirement made our runtime faster and used less memory to read all the words, we included this requirement in sprint 2's backlog. In sprint 1, we hard coded this value to always be 10 and for sprint 2 we want to have the user to be able to modify this value using the drop down value selector we implemented in sprint 1.
  - R2 (✗): This requirement did not make it into our sprint backlog because it was completed in sprint 1.
  - R3 (✗): This requirement did not make it into our sprint 2 backlog because it was completed in sprint 1.
  - R4 (✗): This requirement did not make it into our sprint 2 backlog because it was completed in sprint 1.
  - R5 (✗): This requirement did not make it into our sprint 2 backlog because it was completed in sprint 1.
  - R6 (✗): This requirement did not make it into our sprint 2 backlog because it was completed in sprint 1.
  - R7 (✗): This requirement did not make it into our sprint 2 backlog because it was completed in sprint 1.
  - R8 (✓): This requirement did make it into our sprint 2 backlog because we are slightly unsure how this implementation needs to look like. Therefore, by attempting its

implementation in sprint 2, we can use our stakeholder's feedback to implement this correctly for sprint 3. Our goal for this requirement is to start on the earlier side and then use feedback to make the implementation match what our stakeholders want.

- R9 (✗): This requirement did not make it into our sprint 2 backlog because we had to implement a very similar requirement in our project 1 and believe that we can implement this very quickly in sprint 3. Since we know this will be an easier implementation, we want to use sprint 2 to focus on the other requirements that will require more time and more feedback from our stakeholders.
- R10 (✗): This requirement did not make it into our sprint 2 backlog because it was completed in sprint 1.
- R11 (✓): This requirement did make it into our sprint 2 backlog because we are slightly unsure how this implementation needs to look like. We are specifically confused about what the user interface for this feature should like and where the user should be able to access this search history. Therefore, by attempting its implementation in sprint 2, we can use our stakeholder's feedback to implement this correctly for sprint 3. Our goal for this requirement is to start on the earlier side and then use feedback to make the implementation match what our stakeholders want.
- R12 (✓): This requirement did make it into our sprint 2 backlog because we wanted to finish all the transitions between the pages for this sprint. Since the status bar deals with the transition between the page where the user enters the search query and the page that displays the generated word cloud we decided to include this into our sprint 2 backlog.
- R13 (✓): This requirement did make it into our sprint 2 backlog because we are slightly unsure how this implementation needs to look like. We are specifically confused about what the user interface for this feature should like and if the way we envisioned it was the way the stakeholders wanted it to be done. Therefore, by attempting its implementation in sprint 2, we can use our stakeholder's feedback to implement this correctly for sprint 3. Our goal for this requirement is to start on the earlier side and then use feedback to make the implementation match what our stakeholders want.
- R14 (✓): This requirement did make it into our sprint 2 backlog because we thought this would be a more challenging task to complete than some of the other ones. By working on this requirement's implementation in sprint 2, we still have sprint 3 to make any modifications and changes needed.
- R15 (✓): This requirement did make it into our sprint 2 backlog because we wanted to make sure our design was still up to date before continuing implementing any further features. Our designs help guide our implementation process and, therefore, it is crucial that we confirm that they are up to date before we code a new feature.
- R16 (✓): This requirement did make it into our sprint 2 backlog because the XP practice of develop tests first states that we should have our test files completed first before we write code for the functionality. For this reason, this requirement is on our sprint 2 backlog so that we have the tests made which then allow us to code certain functionalities.

  - R17 (✓): This requirement did make it into our sprint 2 backlog because the XP practice of develop tests first states that we should have our test files completed first before we write code for the functionality. For this reason, this requirement is on our sprint 2 backlog so that we have the tests made which then allow us to code certain functionalities.

- Evidence: https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/ProductBacklog.pdf

  https://github.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/SprintBacklog.pdf

2.2.3 Task Chart

- Sprint Task 1: Give the user an option for number of papers to generate based on search criteria
  - To accomplish this task we need to read the value the user inputs into the input box the user types a value into. Then we need to take this value and place it in the section of code where we have hardcoded the number of papers to search through. By putting the user's value into this place, the user is able to customize how many papers are searched through to generate the word cloud.
- Sprint Task 2: For the paper list, allow users to select a subset to generate a new word cloud from
  - To accomplish this task we need to create a user interface where the user can select which papers he/she wants to generate the word cloud from. Then, we need to pass these papers that were selected into the original search function that generated the word cloud using X papers, where X is the number the user inputted in the first search page. By using the functionality we implemented earlier we should be able to generate a word cloud with the subset of papers the user selected.
- Sprint Task 3: Access previously entered searches on Search Page
  - To accomplish this task we need to create a user interface element where the user can see all of his or her previous searches. The user should be able to access this user interface element in the Word Cloud Page, the Paper List Page, and the Conference page. We then need to make sure that when the user selects one of the previous searches that we generate word cloud for the selected search.
  - Tasks Broken Down
    - Access previously entered searches on Word Cloud Page
    - Access previously entered searches on Paper List Page
    - Access previously entered searches on Conference Page
- Sprint Task 4: Create a status bar
  - To accomplish this task we need to insert a gif of a loading search bar with the percentage of how much the page has been loaded. We then need to insert this gif in whenever the user is waiting for the word cloud to be generated.

  - Tasks Broken Down
    - Measure progress with status bar
- Sprint Task 5: Sort by author column alphabetically on Paper List Page
  - To accomplish this task we need to read each value in the specified column and then using a sorting algorithm to sort all the values alphabetically. Then, we need to display this new sorted column and make sure that all the information in each value's row stays with the column values when the values move around. This needs to be done for the title column and conference column on the Paper List Page as well as with the frequency, author, title, author, and conference column on the Conference Page.
  - Tasks Broken Down
    - Sort by title column alphabetically on Paper List Page
    - Sort by conference column alphabetically on Paper List Page
    - Sort by frequency and author column alphabetically on Conference Page
    - Sort by title column alphabetically on Conference Page
    - Sort the conference column alphabetically on Conference Page
- Sprint Task 6: Fix the frequency column on the paper list page to show accurate frequency
  - To accomplish this task we need to accurately parse the paper and locate everywhere we see the selected word. Each time we see the selected word we need to increase a counter so that we have a final count of how many times that word showed up in the paper. Then, we need to display this frequency next to the paper.
- Sprint Task 7: Update design
  - To accomplish this task we need to go through all of our design diagrams and make sure that they match up with how we envision our implementation.
- Sprint Task 8: Add necessary PHPunit tests
  - To accomplish this task we need to use PHPunit to write all the white-box test cases for our implemented code.
- Sprint Task 9: Add necessary Behat tests
  - To accomplish this task we need to use Behat to write black-box test cases for the code that we will be implementing.
- Evidence
  - During each scrum meeting we also updated the task chart to reflect what new tasks had been accomplished since the last time we met and to clearly see what tasks were left to complete. These task charts are located within their respective scrum folders and the time stamp for each chart's commit message shows that they were in fact updated during each scrum meeting. The links to each task chart is provided below.
  - Scrum 1 Task Chart

        - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum1/Task%20Chart.pdf
    - Scrum 2 Task Chart
        - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum2/Task%20Chart%20Scum%202.pdf
    - Scrum 3 Task Chart
        - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum3/Task%20Chart%20Scrum%203.pdf
    - Scrum 4 Task Chart
        - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum4/Task%20Chart%20Scrum%204.pdf
    - Scrum 5 Task Chart
        - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum5/Task%20Chart%20Scrum%205.pdf
    - Scrum 6 Task Chart
        - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum6/Task%20Chart%20Scrum%206.pdf
    - Scrum 7 Task Chart
        - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum7/Task%20Chart%20Scrum%207.pdf

2.2.4 Burndown Chart

- Scrum 1 Burndown Chart
    - For scrum 1, our team stated that the ideal number of tasks to have remaining at this point would be 18 tasks, meaning we would have completed. Since this is our first scrum for sprint 2, we had not accomplished any of the tasks and, therefore, our actual remaining tasks is 18 as well. Instead of completing tasks from our task chart, we used this time to reestablish ourselves by working on the new sprint backlog, updating the product backlog, creating the new burndown chart for sprint 2, and writing the review and retrospective from sprint 1.
    - Evidence
        - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum1/Burndown%20Chart.pdf
- Scrum 2 Burndown Chart
    - For scrum 2, our team stated that the ideal number of tasks to have remaining at this point would be 15 tasks, meaning we would have completed 3 more tasks since our last scrum. The three tasks we completed consists of us writing Behat black-box test, writing

PHPunit white-box tests, and updating our design diagrams for this sprint. We maintained a sustainable pace by adhering to our burndown chart.

- Evidence
  - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum2/Burndown%20Chart%20Scrum%202.pdf

- Scrum 3 Burndown Chart
  - For scrum 3, our team stated that the ideal number of tasks to have remaining at this point would be 12 tasks, meaning we would have completed 3 more tasks since our last scrum. The three tasks we completed revolved around making the previous search history accessible to the user. We maintained a sustainable pace by adhering to our burndown chart.
  - Evidence
    - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum3/Burndown%20Chart%20Scrum%203.pdf
- Scrum 4 Burndown Chart
  - For scrum 4, our team stated that the ideal number of tasks to have remaining at this point would be 9 tasks, meaning we would have completed 3 more tasks since our last scrum. The three tasks we completed involved allowing the user to select the number of papers to generate the word cloud from, showing the search history on the Conference page, and finished sorting the frequency author column alphabetically on the Paper List page. We maintained a sustainable pace by adhering to our burndown chart.
  - Evidence
    - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum4/Burndown%20Chart%20Scrum%204.pdf
- Scrum 5 Burndown Chart
  - For scrum 5, our team stated that the ideal number of tasks to have remaining at this point would be 6 tasks, meaning we would have completed 3 more tasks since our last scrum. The three tasks we completed involved sorting the tables in our project. Specifically, we finished giving the user the option to sort the title column, author column, conference column on the Paper List page. We maintained a sustainable pace by adhering to our burndown chart.
  - Evidence
    - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum5/Burndown%20Chart%20Scrum%205.pdf
- Scrum 6 Burndown Chart
  - For scrum 6, our team stated that the ideal number of tasks to have remaining at this point would be 3 tasks, meaning we would have completed 3 more tasks since our last scrum. The tasks we completed for this scrum involved sorting the tables in our project.

Specifically, we finished sorting the title column on the Conference Page, fixed the frequency column on the Paper List Page, and finished sorting the conference column on the Conference Page. We maintained a sustainable pace by adhering to our burndown chart.

  - Evidence
    - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum6/Burndown%20Chart%20Scrum%206.pdf

- Scrum 7 Burndown Chart
  - For scrum 7, our team stated that the ideal number of tasks to have remaining at this point would be 0 tasks, meaning we would have completed 3 more tasks since our last scrum. In reality we were only able to complete 2 more tasks. These two tasks we completed were that we finished working on measuring the progress with the status bar and that we finished creating a status bar to measure the progress. We started working on allowing the user to select a new subset of papers to generate a word cloud from. We were unable to finish this task and will continue working on it for sprint 3. We maintained a sustainable pace by adhering very closely to our burndown chart.
  - Evidence
    - https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/scrum7/Burndown%20Chart%20Scrum%207.pdf

2.3 Five Ceremonies

2.3.1 Sprint Planning

All team members met on 4/6/17 which was the day before our first scrum when we were reading up and reviewing what needed to be done for agile development and specifically the new XP practices that were introduced for this sprint. In other words, we analyzed all the requirements for the software. Although we didn't have a formal meeting with the product owner as part of the sprint planning, the scrum master facilitated a meeting for the entire team. For our sprint planning, our team got together to move tasks from the product backlog to the sprint backlog. The team decided the tasks that should be moved from the product log to the sprint log based on the risks associated with the tasks and the relative importance of a particular feature as determined by the scrum master and the team members. This means that the tasks that had higher risks associated with them and the features that were decided to be more important (as based on prior meetings with the product owners), were moved to the sprint backlog.

2.3.2 Sprint

The scrum master determined each sprint to be 2 weeks long. Each sprint comprised taking a specific task and breaking it down into subtasks. Each task was broken down into subtasks so that any one team of pair programming doesn't have to work on entire task, and multiple teams can take on a single task and finish it faster. In this way, tasks are completed faster and more efficiently. Standups also took place as part of the sprint.

2.3.3 Sprint 2 Review and Retrospective

The review ceremony will be completed with the feedback given by our class graders. We will put the software we have developed in front of our stakeholders on 4/17/17. When we display our software in front of the stakeholders, we will ask and receive feedback. Based on this data, we will be able to perform the retrospective ceremony for this sprint as well.

2.3.4 Scrum

There were 7 scrums as part of the sprint. In each scrum the team discussed what each pair programming accomplished since the last meeting, if there were any obstacles since their last meeting and what they will get done in the next meeting.

2.3.4.1 Scrum 1

Arpita and Maya worked on the new sprint backlog and the retrospective for sprint 1. Prerana and Varshi finished the task chart and the burndown chart for sprint 2. Tanuja and Mitali finished the new product backlog and a review for sprint 1. None of the programming pairs faced any obstacles during this scrum.
https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/scrum1

2.3.4.2 Scrum 2

Arpita and Maya worked on updating the unit tests in order to complete the tests for functionality completed in sprint 1. Tanuja and Mitali finished updating the accepting tests to complete the tests for functionality completed in Sprint 1. They also updated the designs to correspond to functionality in Sprint 2. Varshi and Prerana finished updating the designs for sprint 2. None of the programming pairs faced any obstacles.
https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/scrum2

2.3.4.3 Scrum 3

Arpita and Maya finished allowing the user to access the previously entered searches on the Search Page. Tanuja and Mitali finished letting the user access the previously entered searches on the Paper List Page. Varshi and Prerana worked on the showing previously entered searches on the Word Cloud Page. None of the programming teams faced any obstacles.
https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/scrum3

2.3.4.4 Scrum 4

Arpita and Maya finished allowing the user to access the previously entered searches on the Search Page. Tanuja and Mitali finished letting the user access the previously entered searches on the Paper List Page. Varshi and Prerana worked on the showing previously entered searches on the Word Cloud Page. None of the programming teams faced any obstacles.
https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/scrum3

2.3.4.5 Scrum 5

Arpita and Maya finished working on sorting the title column. Tanuja and Mitali finished sorting the author column. Varshi and Prerana finished sorting the conference column on the Paper List page. None of the programming teams faced any obstacles.
https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/scrum5

2.3.4.6 Scrum 6

Arpita and Maya finished working on sorting the title column on the Conference Page alphabetically. Tanuja and Mitali finished fixing the frequency column on the Paper List Page to be accurate. Varshi and Prerana finished sorting the conference column alphabetically on the Conference Page. None of the programming teams faced any obstacles.
https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/scrum6

2.3.4.7 Scrum 7

Arpita and Maya finished working on measuring the progress with the status bar. Tanuja and Mitali worked on allowing the user to select a new subset of papers to generate a new word cloud from. Varshi and Prerana finished creating a status bar to measure the progress. Tanuja and Mitali ran into an obstacle and discussed the next steps with the scrum master.
https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/scrum7

2.3.5 Sprint 1 Review and Retrospective

2.3.5.1 Review

We met with the stakeholders on Monday April 3rd to discuss the progress that we have made with the product. We met with a CP, David, who looked at all of the products that we had completed. For every requirement that he checked, he also looked at the unit tests and acceptance tests that were related to that particular requirement. After meeting with the stakeholders, we received some valuable feedback. Any feedback that involves any changes to our features will require us to put those features back onto our sprint log. For example, on the list of papers, our Word Frequency did not accurately calculate the correct frequency of the word in the paper. Because this feature was not fully implemented, we will be putting it back on the Sprint Backlog. One of the first observations our stakeholders made was related to our black box acceptance tests. Our stakeholder told us that we did not have blackbox tests for certain features that we had implemented. These features that we had implemented but did not have black box tests are: clicking on the title pops up the abstract, having a search history that displays previous searches, and looking at PDFs in new text files. We had black box tests for almost all of the features that we had completed, but only a few had been not completed. This is something that we can easily fix for the next sprint. Our stakeholder also realized that one of our black box tests related to the feature of displaying the word frequency of the clicked word in the word cloud was incorrectly done. We had designed the test case so that it would pass for the specific way we envisioned its implementation rather than designing the test case specifically to the what the requirement wanted. Because of this, the test case would pass even though the requirement was not correctly implemented as the requirement specified. This means for future features and requirements, the black box tests must accurately reflect what the product can do. Some of our black box tests will fail while others will pass. During our discussion, the stakeholder also revealed another additional feature to us. He told us that we need to give the user options to sort the columns alphabetically for any table displayed in the project. This means the author column, the paper title column, and the conference column must be sortable from A to Z as well as Z to A. Finally, one small change the stakeholder mentioned was to change the title of the column “Source” to “Conference.” Overall, the stakeholder praised us on how much we were able to accomplish in Sprint 1 and mentioned that we were very ahead of schedule. He also mentioned that our product was more complete than other team’s products.

2.3.5.2 Retrospective

During Sprint 1, we were successful in accomplishing most of our goals. First, we kept a successful sustainable pace and made sure to get high-risk features done early in the sprint process. By finishing the high-risk features, it helped us finish implementing the features during Sprint 2 and getting confirmation about features during the Sprint 2 Review. During Sprint 3, we can clean up the code and improve any features during Sprint 3. Professor Halfond, a stakeholder, liked this pace and how we are doing a bulk of the features during Sprint 1 and Sprint 2. Also, during Sprint 1, frequent stand up meetings that were decided at the beginning of the sprint was helpful because everyone knew about the six standups at the beginning of Sprint 1. This way everyone could make sure they were available for the standup meetings and knew about the expectations. Another successful part of Sprint 1 was our sustainable pace. Although we felt that most things went well in the first sprint, there are still some areas that need more improvement. There are some changes that we need to make and during one of our initial meetings in sprint 2, we discussed and finalized how we are going to make these changes to improve the various processes in sprint 2. We are going to work on having more effective communication. Sometimes, although team members finished coding a particular component, they forgot to push their code or their pair programming selfies. We want to have more effective communication between team members so that they can remind each other when they forgot to push code or any other forms of evidence like the selfies. We also decided that there should be more communication between the scrum master and the team members so that the scrum master can make sure none of the team members have forgotten to do some kind of work. This means that, although none of the team members didn't forget to complete their tasks on time, they sometimes forgot to push their evidence such as their pair programming selfies or their group selfies at the scrum, or they forgot to push their code more regularly so that the code files on GitHub are regularly updated. Although we were mostly successful with our sustainable pace, we could still improve and make it better. In our burndown chart, there was one steep curve, and we want to get rid of any peaks in the burndown chart to attain a perfect sustainable pace.

2.4 XP Practices

2.4.1 Coding – Pair Programming

Throughout all the processes, the coding ceremony and all the research that was done before the actual implementation of any requirement was always done in pairs. In order to prove that we have been pair programming throughout this assignment, we decided to take a picture of each pair or programmers every time they met to work on the assignment or to do research. We have pushed these images to GitHub so that there would be a time stamp associated with each picture. This time stamp provides evidence that each selfie is taken every time that we pair programmed. Whenever the pair of programmers accomplished significant progress in the coding process, the commit message of the selfie contained some information about the progress that was made that day. For example, one of the selfie commit messages was "tanuja mitali day 1 done, acceptance tests prepared for early testing".

GitHub link to the pair programming selfies for each day:
https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/pairprogrammingselfies

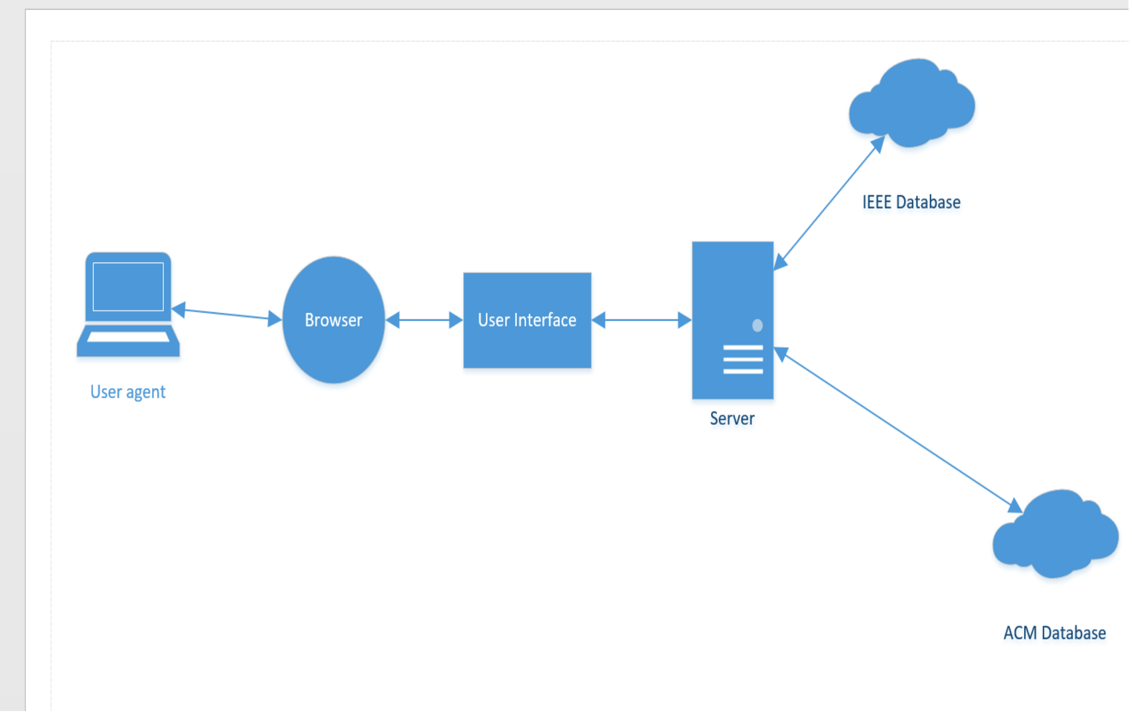
2.4.2 Designing

Designing is an XP Practice which makes it easy to visualize the system from different viewpoints. The diagrams in this section describe the functionality implemented during Sprint 2. In this section we included Architecture, Component, Data Flow, and Sequence Diagrams. All of these diagrams together sufficiently show the designs for the implementation. The designs only represent the implementation from Sprint 2 because using the XP Practices, we only focused on functionality we were completing during Sprint 2.

For simple design, we just make the system do the intended functionality and only that.

The following diagrams strive to keep the simplest designs by making the diagrams as easily understandable and explainable. By being easily understandable and explainable, it will be easier to refactor and add functionality for later sprints. By being easily explainable, new people viewing the designs can figure out how the system works with limited explanation. Refactoring is also part of the Design XP Practice and ensures that functionality can be easily added at increments.

2.4.2.1 Architecture Diagram



The User agent is the way the user interacts with the system. The User agent first opens up a browser to initiate the interaction. The Browser includes a User interface in which the user will will interact with the product. The User Interface communicates with the Server which communicates with two databases. The server communicates with the IEEE Database and the ACM Database. Through this interaction, the user is able to use the full potential of the Research Paper Word Cloud.

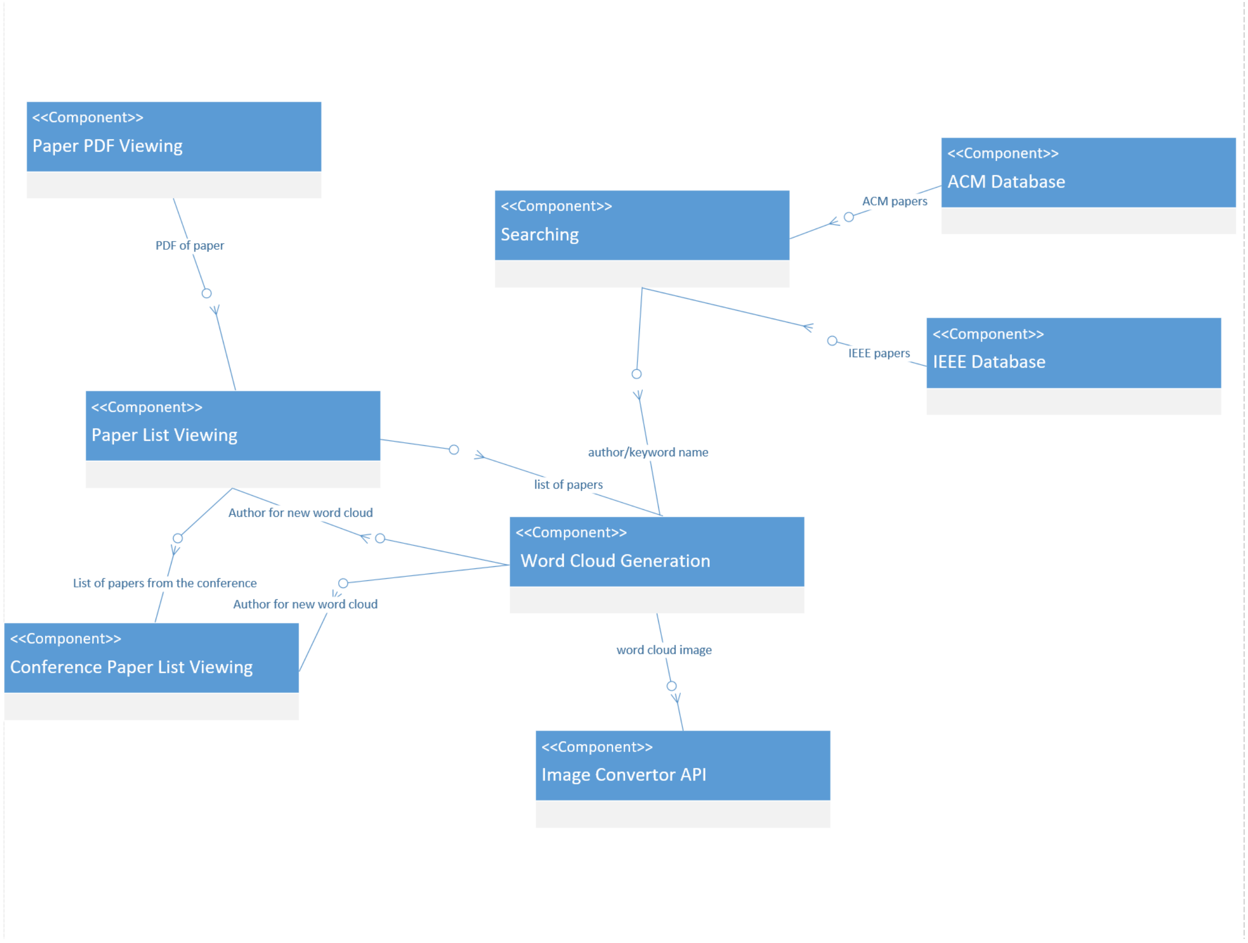
This Architecture Diagram conveys a simple design because of its understandability. There are very few components and connections, 6 components and 5 connectors. This emphasizes the simplicity in the diagram. The flow of information goes between the user communicating with the User interface and the server communicating with the User interface to relay information back and forth.

- Abstraction
  - The System Architecture Diagram focuses on the relationship between the various components and separates the individual functions of each component in this manner. The user does not need to understand the specific functions that are in each component, which we have abstracted with geometric shapes. By abstracting these specifics, we are following the user to emphasize how the components will interact with each other and where data will be passed around.
- Modularity
  - Each component in the System Architecture Diagram is given its own geometric shape with no intersection with any other component. This allows us to group the functionality of each component as individual to all other component functionalities. The cohesion between a given pair of components is represented by an arrow between the two components which signifies the relationship between the two components. The cohesion, or strength, between two modules is determined by the significance of the

data being passed back and forth between the two components. Our System Architecture Diagram models Sequential Cohesion along the component path of User Agent → Browser → User Interface → Server and then to both IEEE Database and ACM Database. The diagram also model Communicational Cohesion in that when the user types in that both the IEEE database and the ACM database are processing the author/keyword at the same time. This communicational cohesion is modeled in the component of Server to both IEEE Database and ACM Database.

- Information Hiding
  - The System Architecture Diagram utilizes information hiding in that when each component generates a piece of data, that component only passes it along to those components that need to use the data. For example, when the server only passes on the author/keyword to the IEEE Database and the ACM Database, the only two components that need to know what the search was. We follow this same logic when creating the relationships between all the components so that each time a new piece of data is created, it is only sent to the components who actually need it and the other components are unaware that this data exists.
- Simplicity
  - The System Architecture Diagram utilizes "A uses B" and "B uses A" intermodular complexity. The diagram shows the number of dependencies in the overall system by illustrating the individual dependencies between each pair of components. Within each of these dependencies we are passing a simple and small amount of data between each component making the complexity of each dependency very small. Some of the pieces of data that we are passing around include a String for the searched for author/keyword to the IEEE Database and to the ACM Database.
- Hierarchy
  - The System Architecture Diagram follows a chaos hierarchy because the flow of information passing between components is not restricted by the layer (layer being the number of components away from the User Agent component). For example, the Browser component (layer 2) can pass data to the User Interface component (layer 3) and this relationship is bidirectional. In this example, we see an upper level passing data to a lower level and we also see the ability of a lower level to pass data to an upper level. The bidirectional relationship between the components is very important because each component is making a request and waiting for a return from the component the request was made to, creating a chaos hierarchy.

2.4.2.2 Component Diagram



Each component sends or receives information from other components. Starting with drop down searching, the search feature gets the data from the ACM and IEEE Database. The Drop Down Searching Component receives these papers from their applicable Components. The World Cloud Generation Component then takes the author/keyword name to create the World Cloud. From the Word Cloud Generation component, the Image Convertor API component can be accessed to create an image version of the Word Cloud. The Word Cloud Generation Component can lead to the Paper List Viewing Component allowing for a list of papers to be displayed. The Paper List Viewing Component sends information about an author if the user clicks on an author on the Paper List page to the Word Cloud Generation Component. The Word Cloud Generation Component then regenerates the Word Cloud. The Conference Paper List Viewing asks for a list of papers from the conference from the Paper List Viewing Component to display. To view the PDF of a paper, the Paper List Viewing Component asks for the pdf version from the Paper PDF Viewing Component.

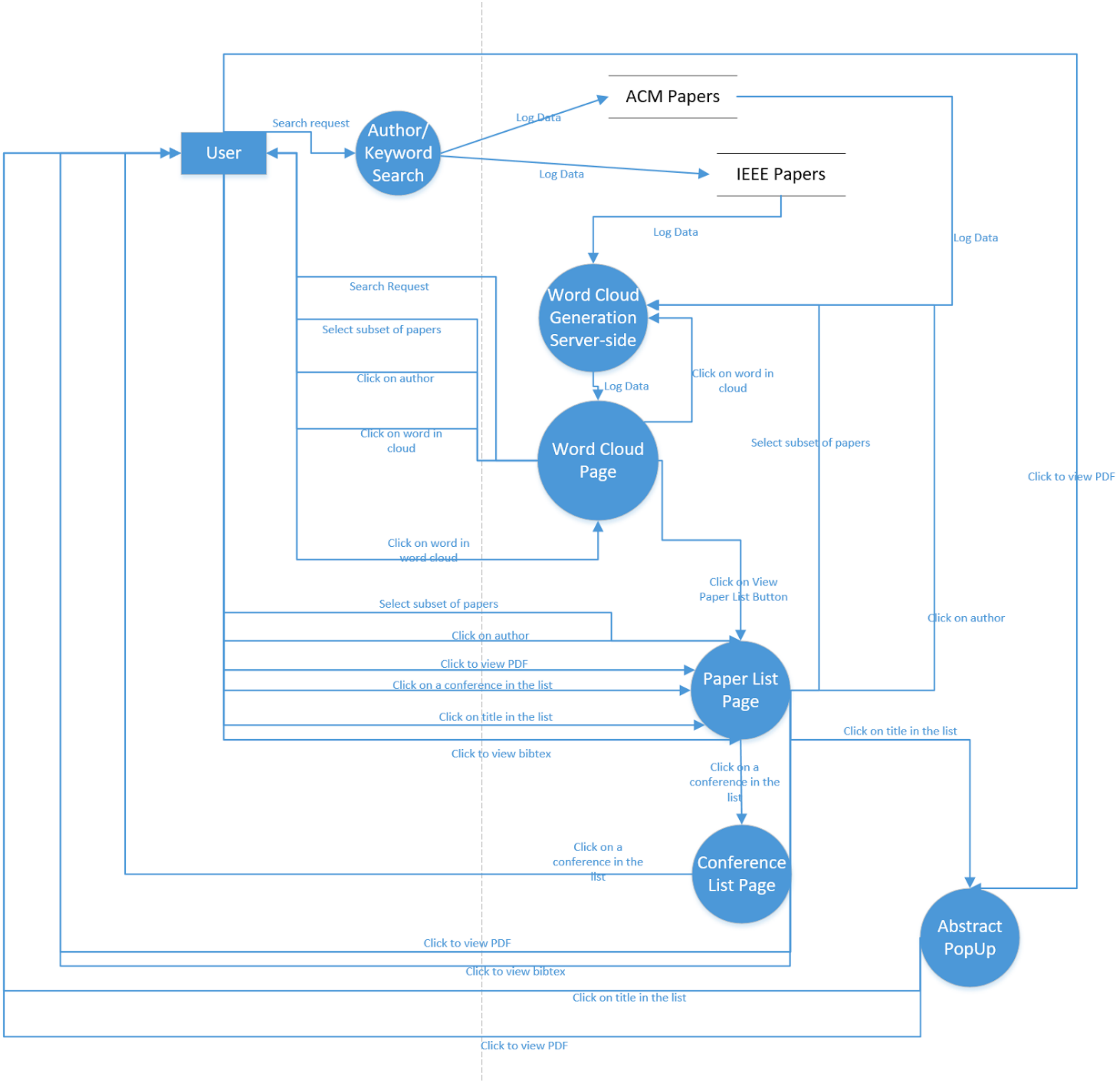
The Component Diagram conveys a simple design because of its understandability and clear structure. There are only the necessary number of components, involving 7 components and 8 connectors. The flow of the information is demonstrated clearly from each component to the other components.

- Abstraction
    - Our Component Diagram utilizes abstraction in order to consider each component individually from its attributes and inner functions. By abstracting away these attributes

and functions, we are able to focus solely on the relationship between the various components, specifically which components share interfaces between each other. The edges between the components allow us to see that which components require data or information from another component, without getting caught up in the specifics of the data or information being passed between the two.

- Modularity
  - Our Component Diagram utilizes modularity in that each component in our system is given its own geometric shape. Each component consists of its own attributes and functions that have been abstracted from the user. Between each component, which are the modules, there is a connection which signifies if the two components require data or information from one another. This edge, which is the sharing of an interface and passing of information, is the cohesion between two components. These edges represent sequential cohesion in that one component requires the output data of another component in order to begin its own processes. Following from this idea, the relationship between the components use content coupling, which means the one component directly affects another component, which represents how our components work with each other. For example, the data from the WordCloud Generation component is directly fed into the Image Convertor API component. Any time the data in the WordCloud Generation component is changed, this directly affects the output of the Image Convertor API component.
- Information Hiding
  - Our Component Diagram utilizes information hiding in that each component determines who needs its data output and selectively sends that data to the components that need it. No other components in the diagram receive the information other than the components that need it. We can see this transfer of information by the edges between the components. The APIs are only connected to the components that actively use the information they provide.
- Simplicity
  - The Component Diagram has inter-module complexity. All of the components in the diagram are not nested and do not contain other modules or components. The components instead depend on information from the other components. For example, the Word Cloud Generation is connected to the Image Convertor API. The Word Cloud Generation components gives information directly to the Image Convertor API component.
- Hierarchy
  - The Component Diagram uses the chaos hierarchy. There is no root node in the graph because lots of components are connected to nodes on different levels. There are also many loops that are formed. For example, the WordCloud generation component and the image convertor API component are both connected directly to each other.

2.4.2.3 Data Flow Diagram



The data flow diagram shows the way data is passed through the system. To start, the user sends a search request containing either the author or a key phrase causing the system to retrieve data from the ACM papers as well as the IEEE papers. That data gets sent to the system which allows the system to generate a word cloud using the related papers. This generation gets displayed on the Word Cloud page. From the Word Cloud page, the user can click on the View Paper List Button making the system display the Paper List Page. If the user clicks on a conference within the Paper List Page, the system will direct the user to the Conference List Page. While on the Word Cloud Page, the user has a variety of actions to choose from. The user can start another search request, causing a new Word Cloud generation. Also, the user can select a subset of papers, which would lead to the Paper List Page. The user can also click on the author or word in the word cloud which also would lead to the Paper List Page. Should the user decide to send the view PDF request, a PDF of the paper will be displayed. If the user clicks on the title in the list of the Paper List Page, an Abstract will pop-up. On the Paper List Page itself, the user can also view the BibTex of a paper.

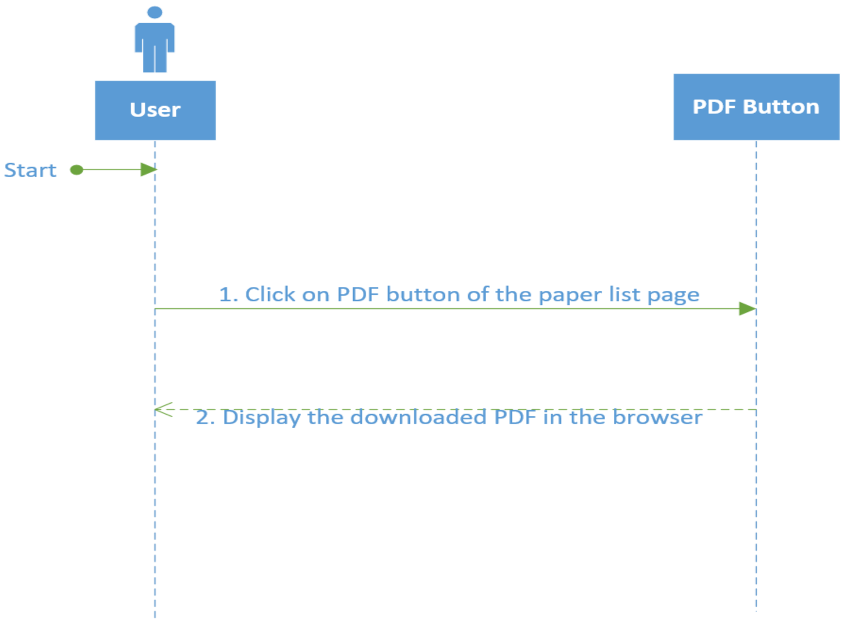
The Data Flow Diagram conveys a simple design because it is easy to understand and view the different ways in which data flows through the different components of the system. Since this diagram is data-oriented, it also refers to the different APIs used as well as emphasizes the specific data being based through the system. Viewing this diagram gives a clear idea of the many possible interactions the user can have with the data and the various results possible for the user actions. To understand all possible ways data is passed in the system, this diagram would be most helpful to refer to.

- Abstraction
    - The Data Flow Design focuses only on how data travels through the system. This means that certain parts of the system like user interface are not indicated in this diagram. The diagram merely focuses on the parts of the system that handles the data directly like different APIs. There are some parts of the user interface like the Search Bar that contains the Author/Keyword search, are necessary instruments in the flow of data, so these are indicated in the design.
- Modularity
    - Each component in the diagram is given its own shape. In this diagram, all of the pages are given a circular shape and the user is given a rectangular shape. Each of the nodes in the diagram are connected because of the data that is passed through the edges. This means that there is communicational cohesion between the nodes because they all operate on the same data that is being passed through. The diagram shows common coupling because each component directly shares data with the next component in the diagram. A Word Cloud cannot be generated without data that is being transformed and supplied.
- Information Hiding
    - In this diagram, we did not show any details or classes in which the data is manipulated in. We wanted to show the high-level components that handle the data in the system. The paths simply show where the data travels, but it does not show how the data is manipulated. We did not show this information because it was not necessary in understanding how the data moves in the system
- Simplicity
    - This diagram uses inter-module complexity. All of the components, the nodes in the graph, are separate and do not contain other modules. Each component uses the other components in order to complete its function. For example, the Word Cloud will be generated using the WordCloud generation server side functionality and the Author/Keyword Search will be done using the data that is passed from the user and is typed into Search Bar.
- Hierarchy
    - The hierarchy in this diagram is chaos hierarchy. While it looks like user could be the root node, it in fact has many paths that lead back to it. Since the data flow is a circular process, there is no one starting point and all of the nodes are connected in a very complicated fashion. The User node can be accessed at any point because there are so many different requests that can occur.

2.4.2.4 Sequence Diagrams

The sequence diagrams describe the order of events as a timeline. The diagram is organized by objects and explains the relationships between all of the objects. This diagram also describes the return messages if necessary. Every action will be tied to an object to show the specific connections. Each object is a lifeline to each message.

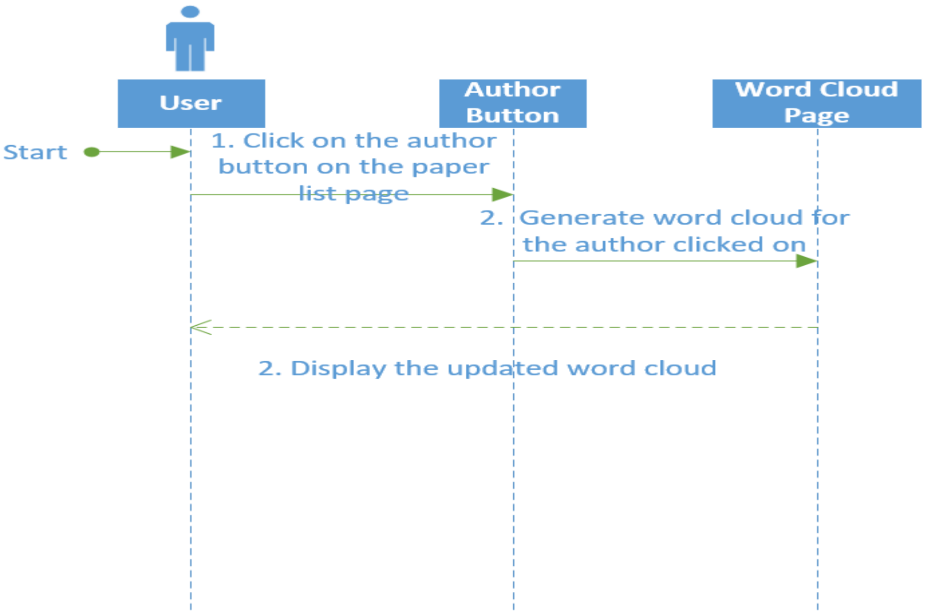
2.4.2.4.1 Sequence Diagram 1


User
PDF Button
Start
1. Click on PDF button of the paper list page
2. Display the downloaded PDF in the browser

This diagram describes the events that take place when the user presses the PDF button of the paper list page. The diagram starts with the scenario when a user clicks on a word in the WordCloud, and a page containing a table with the names of ten research papers that have the highest frequency of occurrence of the word that was clicked is generated. This page that contains the information about all these papers, is referred to as the paper list page. The table also contains a “download PDF” button and once the button is pressed, a PDF version of the paper is downloaded and the downloaded PDF is also displayed in the browser. As we can see in the diagram, there are only 2 life lines. The life lines are the user who is the actor and the PDF is the component. In addition, there are only two messages that are being sent back and forth. The first message is sent from the actor to the component. The second is sent from the component to the actor when the user clicks on the “download” button. The second message displays the downloaded PDF in the browser.

Requirement Satisfied	**Represented Feature**
Requirement 7	The requirement discusses the download features on the paper list page. However, although the requirement specifies download as plain text and PDF, this diagram only depicts the download of the PDF.

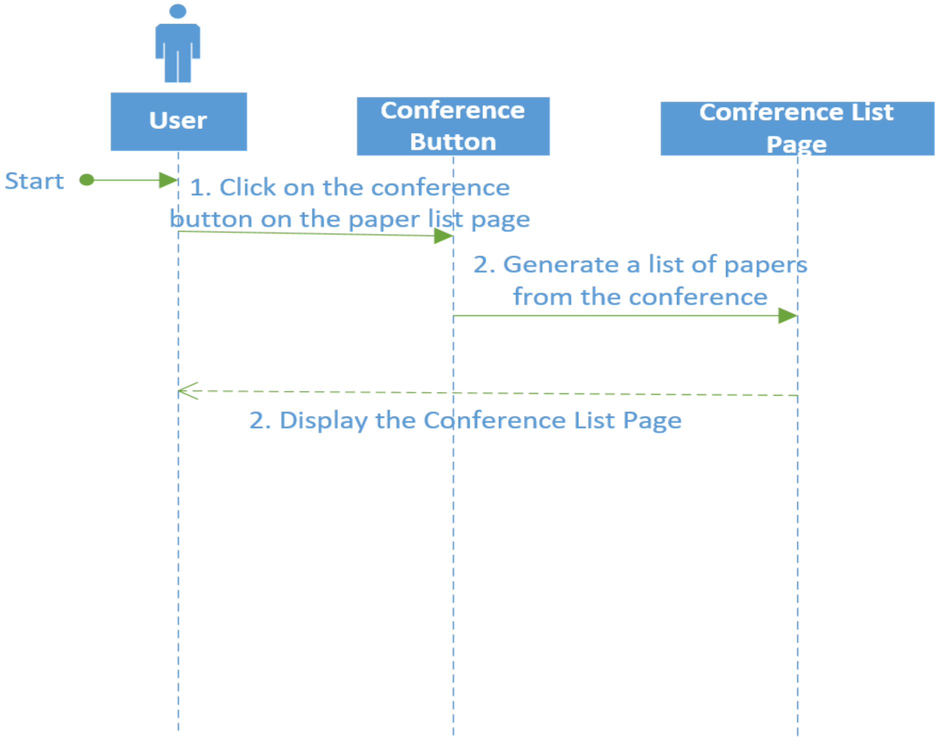
2.4.2.4.2 Sequence Diagram 2


User
Author Button
Word Cloud Page
Start
1. Click on the author button on the paper list page
2. Generate word cloud for the author clicked on
2. Display the updated word cloud

This diagram describes the sequence of events that occur when a user wants to click on an author in the Paper List Page. The scenario starts when the user is on the Paper List Page. In table of all of the papers, the user can click on any author of any paper; all of the authors are buttons. As soon as the "Author Button" is clicked, the system will generate a new Word Cloud based on the the author that was clicked and the user will be navigated to the Word Cloud Page. This diagram shows a very simple viewpoint. There are only three lifelines, two components, and one actor in the diagram. There are three messages that are being passed through the lifelines. The diagram does not show any of the backend operations that go into creating the Word Cloud.

Requirement Satisfied	**Represented Feature**
Requirement 5	This diagram shows how clicking an author in the author list per paper will generate a new Word Cloud based on the clicked author.

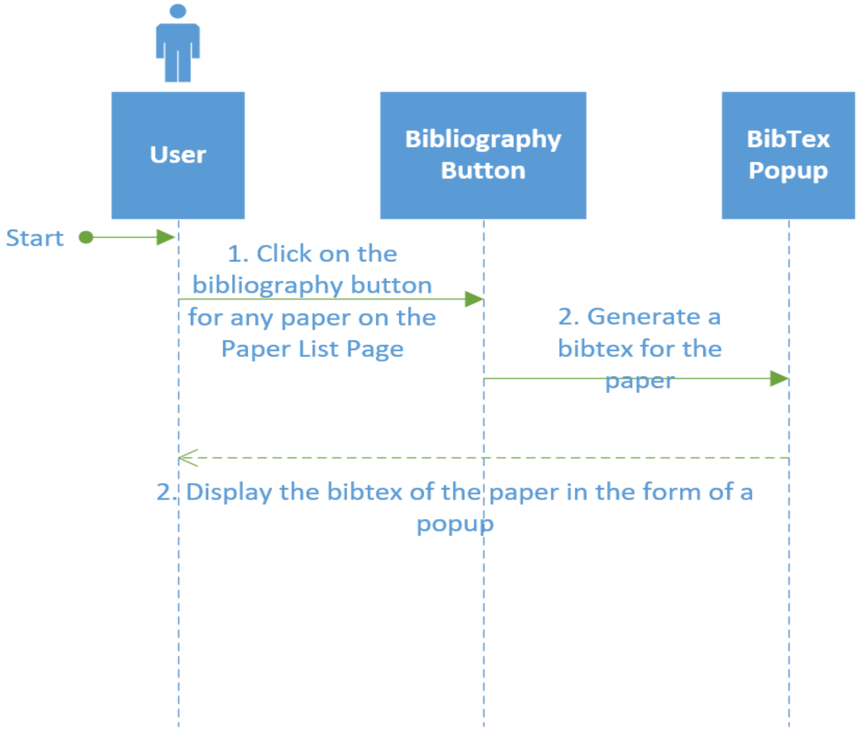
2.4.2.4.3 Sequence Diagram 3


User
Conference Button
Conference List Page
Start
1. Click on the conference button on the paper list page
2. Generate a list of papers from the conference
2. Display the Conference List Page

This diagram describes the sequence of events that occur when the user clicks on the conference button on the paper list page. The diagram starts with the scenario that the user is on the paper list page. The paper list page contains a table of top ten papers with the highest frequencies of the word that was clicked on the Word Cloud. Apart from other relevant information such as the frequency of occurrence of the clicked word and the download PDF button, the table also contains the conference where the paper was presented. The name of the conference acts as a “conference button”. When the conference button is clicked on the paper list page, a list of papers from the conference is generated. The user is then navigated to the Conference List Page where the information about other research papers that were presented at the same conference is displayed. This diagram has one actor, which is the user, and two components, which are the conference button and the conference list page. Only 2 messages are sent in this diagram: a message is sent from the actor to the conference button when the button is clicked and another message is sent from the conference button to the conference list page.

Requirement Satisfied	**Represented Feature**
Requirement 10	This requirement states that for each paper, clicking on a paper’s conference name will list other papers from that conference.

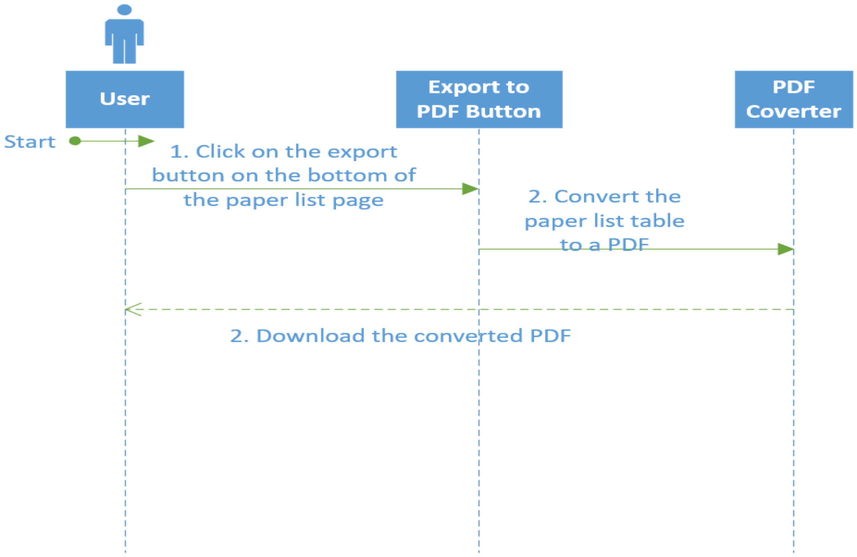
2.4.2.4.4 Sequence Diagram 4


User
Bibliography Button
BibTex Popup
Start
1. Click on the bibliography button for any paper on the Paper List Page
2. Generate a bibtex for the paper
2. Display the bibtex of the paper in the form of a popup

This diagram describes the sequence of events that occur when a user wants view the bibtex of a paper in the Paper List Page. If the user is on the Paper List Page, the user can go to any paper that is in the list displayed and click on the Bibliography Button that is next to each paper. The system will then generate a bibtex for the paper that the user wants to view. Then, the system will display the bibtex that was generated in a popup to the user. This simple diagram has three lifelines, one actor, and two components. There are a total of three messages that are passed between lifelines. The first message is sent from the user to the Bibliography Button. The second message is sent from the Bibliography Button to the BibTex Popup. Finally, the last message is a display message that is sent from the BibTex Popup to the user.

Requirement Satisfied	**Represented Feature**
Requirement 6	This diagram shows the feature that allows the user to click on a link and gain access to a paper's bibtex.

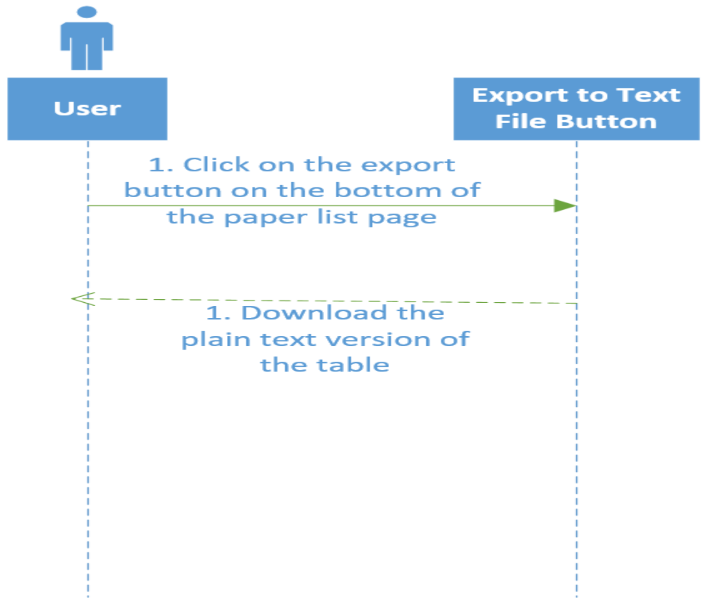
2.4.2.4.5 Sequence Diagram 5


User
Export to PDF Button
PDF Coverter
Start
1. Click on the export button on the bottom of the paper list page
2. Convert the paper list table to a PDF
2. Download the converted PDF

This diagram describes the events that take place when the user presses the "export to PDF button" of the paper list page. The diagram starts with the scenario when a user clicks on a word in the WordCloud, and a page containing a table with the names of ten research papers that have the highest frequency of occurrence of the word that was clicked is generated. This page that contains the information about all these papers, is referred to as the paper list page. The paper list page contains an export to PDF button at the bottom. When the user presses this button, a message is sent from the user, who is the actor, to the button, which is a component the paper list table is converted to a PDF. The net event that takes place is that another message is sent from the button to the PDF convertor, which is another component. In this diagram, there are 2 components, 1 actor and 2 messages being sent.

Requirement Satisfied	**Represented Feature**
Requirement 7	The requirement discusses the download features on the paper list page. However, although the requirement specifies download as plain text and PDF, this diagram only depicts the download of the table of papers as a PDF.

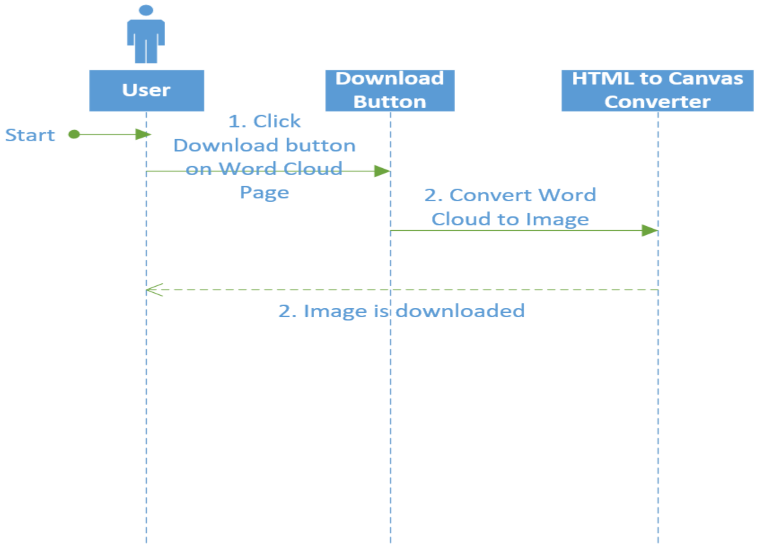
2.4.2.4.6 Sequence Diagram 6


User
Export to Text File Button
1. Click on the export button on the bottom of the paper list page
1. Download the plain text version of the table

This diagram describes the sequence of events that occur when a user wants to convert the list of papers on the Paper List Page into plain text. If the user is on the Paper List Page, the user can click on the Export button at the bottom of the page to convert the list to plain text. Once the user clicks on this button, the system will generate a document in plain text of the table. User will then see the document downloaded on their computer. This diagram has one actor and one component as well as two lifelines. There are only two messages that are passed through lifelines. The first message is from the user to the Export button. The second message is from the Export button back to the user.

Requirement Satisfied	**Represented Feature**
Requirement 7	This diagram shows the feature that allows the user to export the list of papers into plain text.

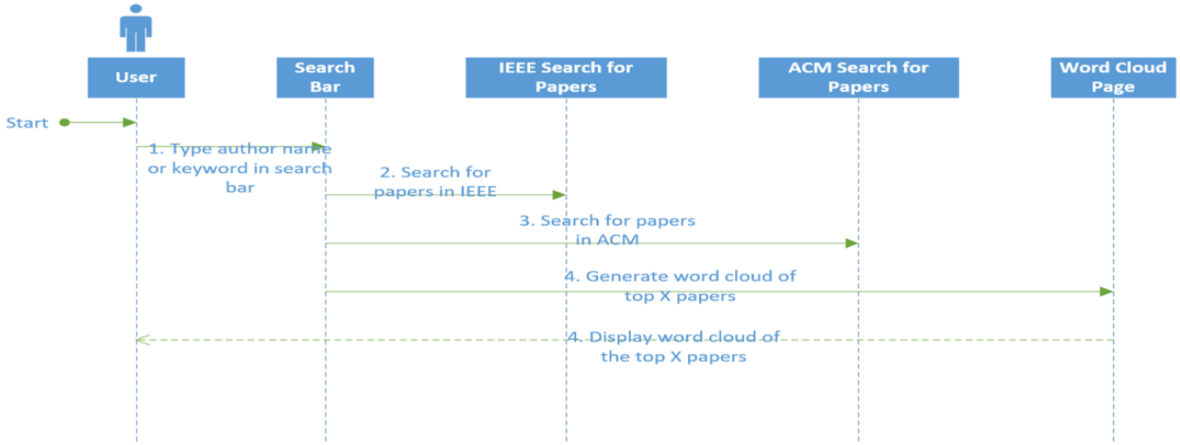
2.4.2.4.7 Sequence Diagram 7


User
Download Button
HTML to Canvas Converter
Start
1. Click Download button on Word Cloud Page
2. Convert Word Cloud to Image
2. Image is downloaded

This diagram describes the sequence of events that occur when a Word Cloud is generated and the user clicks on the download button. The diagram starts with the scenario that a WordCloud has been generated and the user presses the Download button on the WordCloud page. Once the button is presses, the WordCloud is converted to an image with the help of an HTML to canvas convertor. After it is converted into an image, the WordCloud is then downloaded. There is one actor (user) and two components (download button and the HTML to canvas convertor) used in this diagram. There are only 2 messages sent in the diagram: when the button is presses a message is sent from the user to the download button and another message is sent from the download button to the HTML to canvas convertor when the WordCloud needs to be converted to an image.

Requirement Satisfied	**Represented Feature**
Requirement 9	This diagram shows the feature that allows for the downloading of an image of the generated WordCloud.

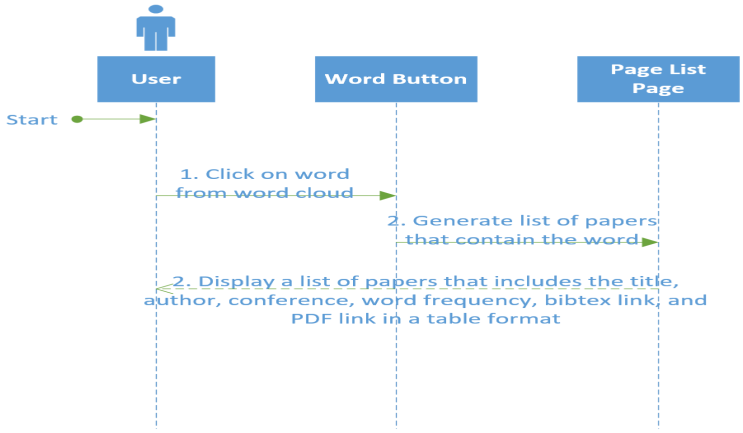
2.4.2.4.8 Sequence Diagram 8


User
Search Bar
IEEE Search for Papers
ACM Search for Papers
Word Cloud Page
Start
1. Type author name or keyword in search bar
2. Search for papers in IEEE
3. Search for papers in ACM
4. Generate word cloud of top X papers
4. Display word cloud of the top X papers

This diagram describes the sequence of events that occur when the user wishes to generate a Word Cloud based on a searched author or keyword. The user can start this sequence of actions when the user is on the Search Page. The user can first type a keyword or author into the Search Bar. The system will then search for papers in the IEEE database and the ACM database that match the searched author or keyword. The system will then generate a Word Cloud based on the papers that it got from the two databases. The Word Cloud of the searched author or keyword will finally be displayed to the user. This simple design has one actor, four components, and five lifelines. There are a total of five messages that are sent through the lifelines.

Requirement Satisfied	**Represented Feature**
Requirement 2	This diagram shows the feature that allows a user to input search criteria based on the author (researcher's last name) or a keyword phrase.
Requirement 1	This diagram shows that when a search of the author's name or a keyword is submitted, the system will create a Word Cloud using papers from the IEEE and ACM digital libraries.

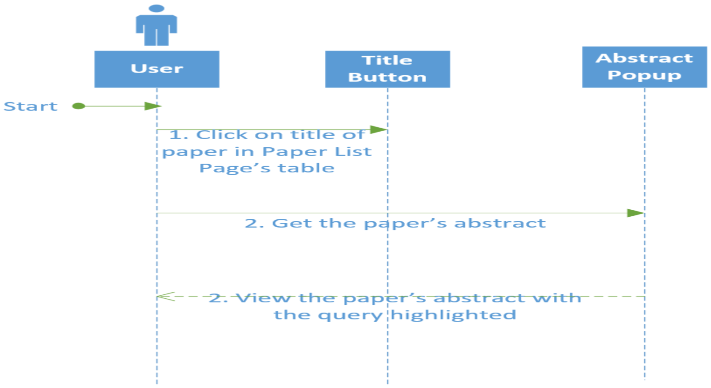
2.4.2.4.9 Sequence Diagram 9


User
Word Button
Page List Page
Start
1. Click on word from word cloud
2. Generate list of papers that contain the word
2. Display a list of papers that includes the title, author, conference, word frequency, bibtex link, and PDF link in a table format

This diagram describes the sequence of events that occur when the user clicks on a word on the WordCloud page. The diagram starts with the scenario that a WordCloud has been generated and the user clicks on one of the words in the WordCloud. Since each word on the WordCloud is clickable, they are referred to as functioning as “Word Buttons” in the diagram. When a word button is pressed, a list of papers that contain the largest frequencies of the clicked word are generated and the user is navigated to the Page List Page that contains a table with the information of all these papers that contain the clicked word. The table displays the title, author, conference, word frequency of the clicked word, bibtext link and PDF link of all the relevant papers. This diagram contains one actor (the user), and 2 components (Word Button and the Page List Page) and 2 messages are sent in this diagram.

Requirement Satisfied	**Represented Feature**
Requirement 2	This requirement ensures that when a word in the WordCloud is presses, a list of papers that mention that word should be returned.

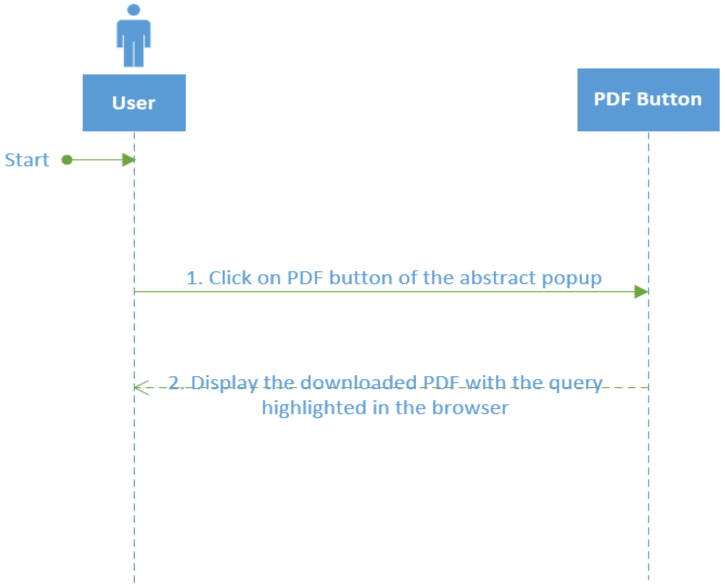
2.4.2.4.10 Sequence Diagram 10


User
Title Button
Abstract Popup
Start
1. Click on title of paper in Paper List Page's table
2. Get the paper's abstract
2. View the paper's abstract with the query highlighted

This diagram describes the sequence of events that occur when the user wants to see the abstract of a paper in the Paper List Page. This sequence starts when the user is currently on the Paper List Page. The user can then click on the title of any of the papers that are in the Paper List Page. As soon as the user clicks on a title, the system will then fetch the abstract of the clicked paper and display the abstract in a pop up window. Before displaying the abstract, the system shall go through the abstract and highlight the clicked term in the abstract. When the user views the abstract in a popup window, the user will be able to see the clicked term highlighted in the abstract. The abstract popup is displayed to the user. This diagram has one actor, two components, and three lifelines. There are a total of three messages that are sent through the lifelines. The first message is from the user to the Title Button. The second message is from the Title Button to the Abstract Popup. The final message is sent from the Abstract Popup back to the user.

Requirement Satisfied	**Represented Feature**
Requirement 4	This diagram shows the feature that for each paper, if you click on the paper's title, the user can view the abstract of the paper with the clicked word highlighted in the abstract.

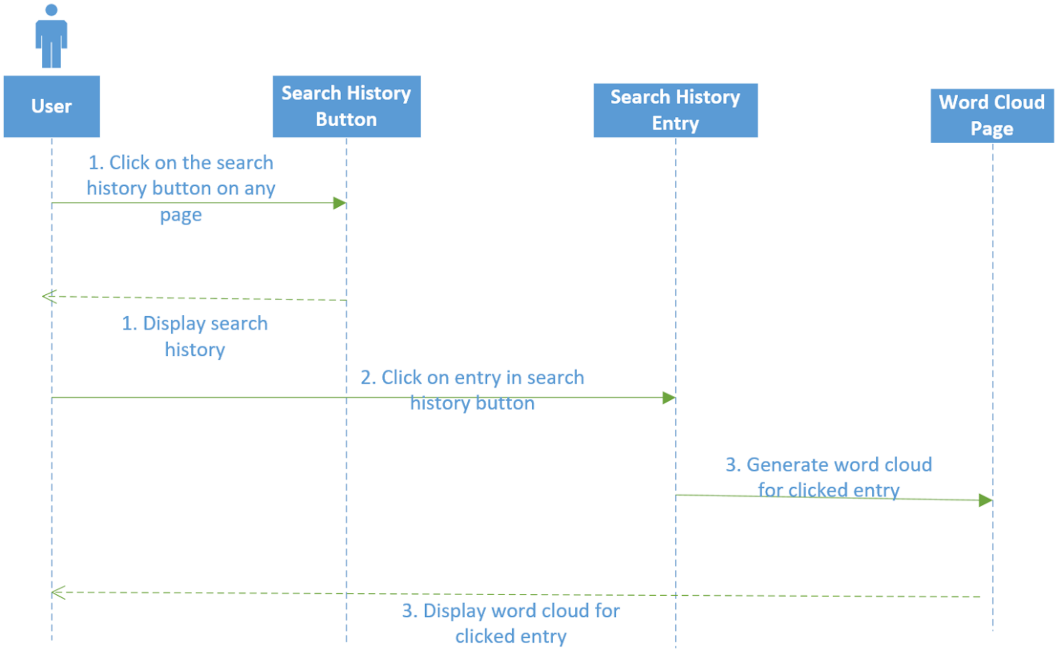
2.4.2.4.11 Sequence Diagram 11


User
PDF Button
Start
1. Click on PDF button of the abstract popup
2. Display the downloaded PDF with the query highlighted in the browser

This diagram describes the sequence of events that occur when the user wants to download the PDF version of a research paper. All occurrences of the clicked word will be highlighted in this version of the research paper. The diagram starts with the scenario that the user is on the Paper List Page after clicking on a specific word on the Word Cloud. When the user clicks on a title of any of the papers in the table, a popup with the abstract of that paper is generated. The abstract will contain a “PDF button”. When the user clicks on the PDF button on the popup, a PDF with all the occurrences of the word on the WordCloud that was clicked, highlighted, is downloaded. The downloaded PDF is then displayed with the query highlighted on the browser. There are is one actor (the user) and one component (PDF button), in this diagram and only one message is generated and sent from the user to the PDF button when the user clicks on it

Requirement Satisfied	**Represented Feature**
Requirement 4	This diagram shows the feature that allows the user to download the PDF of the clicked paper. This feature is on the same popup as the abstract.

2.4.2.4.12 Sequence Diagram 12


User
Search History Button
Search History Entry
Word Cloud Page
1. Click on the search history button on any page
1. Display search history
2. Click on entry in search history button
3. Generate word cloud for clicked entry
3. Display word cloud for clicked entry

This diagram describes the sequence of events that occur when the user wants to access his or her search history. There will be user interface element on the screen and when the user clicks on this button-like element, a drop down will appear that shows the previous searches. The diagram starts with the scenario that the user clicks on the history button on any page of the project. The user can then click on any of the previous search entries. Once the user clicks on an entry, the user is redirected to the Word Cloud page where he or she can see a word cloud generated for that search. On this Word Cloud Page, the user can once again view the search history.

Requirement Satisfied	**Represented Feature**
Requirement 11	This diagram shows the feature that allows the user to access his or her search history. This feature is accessible on all pages.

2.4.2.4.13 Quality Metrics

We decided that the Sequence Diagram is useful because it shows how objects work with one another as well as in what order. In terms of simplicity, the Sequence Diagram shows only the messages and components that are specifically used when a user is doing an action. This allows our diagram to focus only on the feature that is being diagramed. In terms of content, it is very similar to the Communication Diagram but based on its structure, the Sequence Diagram is more likely to be used if one wants to simply analyze which interactions take place overall and the events in which they happen. The sequence diagram puts emphasis on the timeline of the events. It is therefore very useful if one wants to examine the order in which the individual parts of different actions take place.

- Abstraction
    - All of the Sequence Diagrams show the user's interaction with the system at a high level using objects. Each message describes multiple low level processes. By grouping multiple processes together and showing it as one, unified message the ideas are more abstract. The objects are also a very high level representation of the program's different entities. By making the Sequence Diagrams high level, it helps the team understand the overall flow of the different actions a User can take.
- Modularity
    - The objects in the Sequence Diagrams represent different modules. The diagrams further this by showing the different relationships between all of the modules. The diagrams use sequential cohesion since messages are sent and returned between modules. The diagrams show common coupling because each time a message is sent from a lifeline, it shares data with the receiving lifeline. The data is sent from one object to the next.
- Information Hiding
    - The user does not know the information within each object. The user does not interact with some of the objects in the Sequence Diagrams. For example, the user does not directly interact with the digital libraries like IEEE and ACM. Those are there to help with the functionality. The user only interacts with the components displayed on the web page. The cohesion of each module keeps it together and separate from other objects. For each message, the coupling is not as strong because of the independence of each module. The strength between modules decreases.
- Simplicity
    - The Sequence Diagrams are inter-modular because all of the relationships are between objects. The messages are sent between different objects. In the diagrams, the messages are never sent within an object. Each message shows how the final lifeline uses the initial lifeline.
- Hierarchy
    - The Sequence Diagrams follow a regular hierarchy because all of the events happen sequentially and none of the messages overlap. The User is the root of the hierarchy because each Sequence Diagram starts with the User. The Sequence Diagrams are User focused. The different objects would be different nodes and the messages would be different relationships between them.

2.4.3 Testing – Acceptance and Unit Tests

- Throughout the software processes, acceptance and unit tests were developed before the code for a particular requirement or feature was written. Although, acceptance tests are typically written by the customer, we had to write these in our case. We therefore wrote the acceptance tests from the perspective of the customer who will be using the developed software. Unit tests were designed so that all the backend code is being tested. The various testing processes ensured that no bugs were present in the code.
- GitHub links to Acceptance Tests: https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/AcceptanceTests
- Github links to Unit Tests: https://github.com/mkarmark/USC_CSCI_310_Project_2/tree/master/code/app/test/white

2.4.3.1 Develop Tests First

During the software processes, the first things developed were the acceptance and unit tests for the features we decided to implement for Sprint 2. The reason they were developed before implementation is to be aware of all possible ways in which our system can be used by the user. This test-driven approach aided us in optimizing the functionality of our code with minimal errors in the process. The link to our tests can be viewed in Section 2.4.3.3.
GitHub links to the commits that verify that the tests were developed first:
https://github.com/mkarmark/USC_CSCI_310_Project_2/commit/f83940fb413590176ece9d96c6638f3cdf48c966
https://github.com/mkarmark/USC_CSCI_310_Project_2/commit/82cbd51f77891cafb0d0c7e6a425ac0974ca8d5a

2.4.3.2 Continuous Testing

When we added new features for our project, we wrote unit and acceptance tests for all of the new features. As we added code to the different features we were working on, we would run our test suites and see which test cases passed and failed. We made sure to never write code without having a test case for it first, as tests were ran constantly. Running tests was a continual process in this sprint. We took screenshots of the results of the tests when the tests were run.
Evidence for running black box tests in every pair programming episode can be found here:
https://github.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/acceptanceTests
Evidence for running relevant white box tests in every pair programming episode can be found here:
https://github.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/unitTests

2.4.3.3 Creating Tests for Bugs

Whenever a bug was found, a new test was made to address the bug. We made sure that we ended up passing this new test in the sprint. The links provided have screenshots of the initial test that failed that drew attention to the bug, the new tests we wrote that further addressed the bug, and the screenshots of the tests being solved. Updated feature files and unit tests can also be found in the acceptance test and unit test folders, for which links have been provided above.

2.4.3.3.1 Status Bar Error (Acceptance Test)

For this error, we saw that the status bar that we had implemented did not show up properly when we had added the new feature. We had no errors in our code, but the status bar did not appear. We created this acceptance test scenario to check if the status bar appeared every time we altered our code. Once we altered the code, the status bar began to appear in our user interface.

GitHub Link: https://GitHub.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/Bug1

2.4.3.3.2 Conference Column Error (Acceptance Test)

For this error, we saw that the conference column was not sorting correctly. We had gotten our other columns to sort numerically and alphabetically, but the conference column had a slight issue. We created this acceptance test to check if the column was sorting correctly. Once we fixed the bug, the acceptance test started passing.

GitHub Link: https://github.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/Bug2

2.4.3.3.3 Creation of Paper Error (Unit Test)

For this error, our word frequency column was getting the frequency of the clicked word in the abstract of the paper, not the actual paper itself. The numbers we were showing on the abstract column were much smaller than the actual number. In our unit test, we made sure that the value that we were using for the frequency came from the actual paper text and not the abstract text like we had before.

GitHub Link: https://github.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/Bug3

2.4.3.3.4 Search History on Paper List Page Error (Acceptance Test)

For this error, we noticed that the sidebar that held all of the previous search history for the application was not visible on the Paper List Page. The user could access it on the Search Page, but not the Paper List Page. We added this acceptance test to check every time we made a fix if the sidebar appeared on the Paper List Page.

GitHub Link: https://github.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/Bug4

2.4.3.3.5 Search History on Word Cloud Page Error (Acceptance Test)

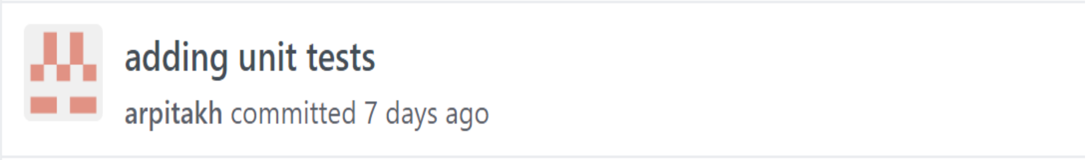
Like Bug 4, we noticed that this was also happening on the Word Cloud Page. We noticed that the sidebar that held all of the previous search history for the application was not visible on the Word Cloud Page. The user could access it on the Search Page, but not the Word Cloud Page. We added this acceptance test to check every time we made a fix if the sidebar appeared on the Word Cloud Page. We were able to fix both of the errors.

GitHub Link: https://github.com/mkarmark/USC_CSCI_310_Project_2/tree/master/Sprint2/Bug5

2.4.4 Refactoring

2.4.4.1 Arpita and Maya Day 1

- Commit



adding unit tests

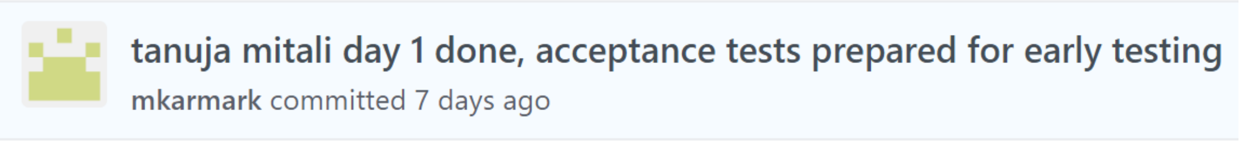
arpitakh committed 7 days ago

  - In this pair-programming episode, Arpita and Maya updated the unit tests for the new features being introduced in Sprint 2.
- Refactoring Explanation
  - The new requirements for Sprint 2 required the existence of search history and sorting. The current unit tests were not sufficient to test this functionality, so Arpita and Maya had to insert more unit tests to account for this new functionality. This is therefore an

example for refactoring because the sniffed problem was the absence of relevant unit tests, and the problem was resolved by adding more relevant unit tests.

2.4.4.2 Mitali and Tanuja Day 1

- Commit



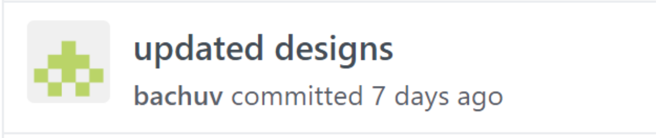
tanuja mitali day 1 done, acceptance tests prepared for early testing

mkarmark committed 7 days ago

  - In this pair programming episode, Tanuja and Mitali updated the acceptance tests for the new features being introduced in Sprint 2.
- Refactoring Explanation
  - The new requirements for Sprint 2 required the existence of search history and sorting. The current acceptance tests were not sufficient to test this functionality as they did not account for the new buttons these features would introduce, so Tanuja and Mitali had to insert more feature files to account for this new functionality. This is therefore an example for refactoring because the sniffed problem was the absence of relevant feature files, and the problem was resolved by adding more relevant feature files.

2.4.4.3 Prerana and Varshi Day 1

- Commit



updated designs

bachuv committed 7 days ago

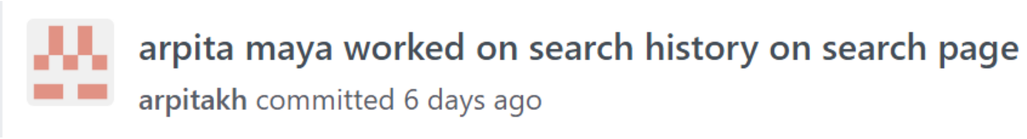
  - In this pair programming episode, Prerana and Varshi updated the designs for the new features being introduced in Sprint 2.

- Refactoring Explanation

  - The new requirements for Sprint 2 require the existence of search history. The current sequence diagrams did not account for this behavior, Prerana and Varshi made a new sequence diagram to introduce this new behavior. The current component diagram also did not account for communication from the Conference list page to the Word cloud page (when an author is clicked), so they updated the existing component diagram to include this feature. This is therefore an example for refactoring because the sniffed problem was the absence of certain behavior in the pre-existing diagrams, and the problem was resolved by adding a new diagram and modifying the existing diagram.

2.4.4.4 Arpita and Maya Day 2

- Commit



arpita maya worked on search history on search page

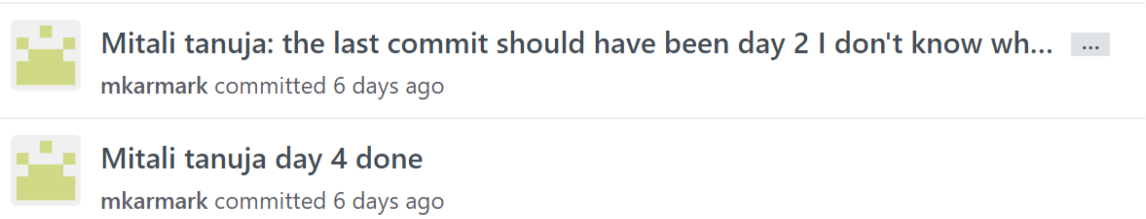
arpitakh committed 6 days ago

  - In this pair programming episode, Arpita and Maya added in the search history feature on the search page.

- Refactoring Explanation
  - To add the search history feature, a new button had to be added in onto the search page. By adding this button in the sidebar, a click on the button prompts a new window to display the past searches made on the website in the browser session. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the button triggers on the search page. This functionality was thus inserted in during the refactoring process. This is therefore an example for refactoring because the sniffed problem was that the absence of a search history button on the search page prevented the users from looking up search history, and the problem was resolved by adding in the search history button, allowing users to view search history by clicking the button.

2.4.4.5 Mitali and Tanuja Day 2

- Commit



Mitali tanuja: the last commit should have been day 2 I don't know wh... ...
mkarmark committed 6 days ago

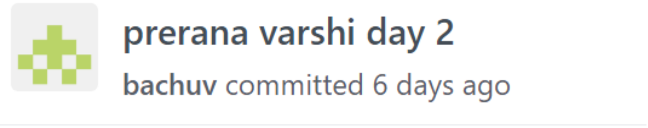
Mitali tanuja day 4 done
mkarmark committed 6 days ago

  - In this pair programming episode, Tanuja and Mitali added in the search history feature on the Paper List page.

- Refactoring Explanation
  - To add the search history feature, a new button had to be added in onto the paper list page. By adding this button in the sidebar, a click on the button prompts a new window to display the past searches made on the website in the browser session. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the button triggers on the paper list page. This functionality was thus inserted in during the refactoring process. This is therefore an example for refactoring because the sniffed problem was that the absence of a search history button on the paper list page prevented the users from looking up search history, and the problem was resolved by adding in the search history button, allowing users to view search history by clicking the button.

2.4.4.6 Prerana and Varshi Day 2

- Commit



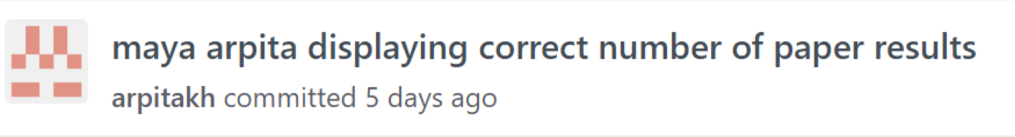
prerana varshi day 2
bachuv committed 6 days ago

  - In this pair programming episode, Prerana and Varshi added in the search history feature on the Word Cloud Page.

- Refactoring Explanation
  - To add the search history feature, a new button had to be added in onto the word cloud page. By adding this button in the sidebar, a click on the button prompts a new window to display the past searches made on the website in the browser session. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the button triggers on the word cloud page. This functionality was thus inserted in during the refactoring process. This is therefore an example for refactoring because the sniffed problem was that the absence of a search history button on the word cloud page prevented the users from looking up search history, and the problem was resolved by adding in the search history button, allowing users to view search history by clicking the button.

2.4.4.7 Arpita and Maya Day 3

- Commit



maya arpita displaying correct number of paper results

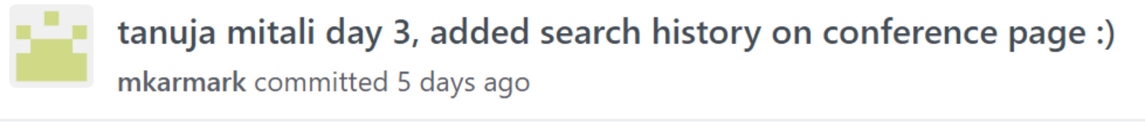
arpitakh committed 5 days ago

  - In this pair programming episode, Arpita and Maya added the feature to select the correct X number of papers.

- Refactoring Explanation
  - Before this feature was implemented, 10 papers were always searched for by default. Therefore, the search page had no such input field that let the user choose how many papers the user wants to search through. When they added in this feature into Sprint 2, they needed a way to gather user input on how many papers to search through. To accomplish this, they added a drop-down to the search page that has numbers from 1 to 100, letting the user choose how many papers to search through. They also fed this information to the word cloud page and the paper list page. There were two instances of refactoring that took place in this pair programming episode. The first sniffed problem was that the absence of an input tool prevented the users from providing the number of papers to search from, and this problem was resolved by providing a drop-down option to accomplish this task. The second sniffed problem was that there was no priorly designed communication that was passing an integer from the search page to the word cloud page or paper list page. This communication was needed to communicate to the other pages how many papers to search from. This problem was resolved by setting up such communication through session variables.

2.4.4.8 Tanuja and Mitali Day 3

- Commit



tanuja mitali day 3, added search history on conference page :)

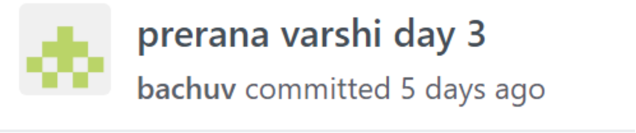
mkarmark committed 5 days ago

- In this pair programming episode, Tanuja and Mitali added in the search history feature on the Conference List page.

- Refactoring Explanation
    - To add the search history feature, a new button had to be added in onto the conference list page. By adding this button in the sidebar, a click on the button prompts a new window to display the past searches made on the website in the browser session. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the button triggers on the conference list page. This functionality was thus inserted in during the refactoring process. This is therefore an example for refactoring because the sniffed problem was that the absence of a search history button on the conference list page prevented the users from looking up search history, and the problem was resolved by adding in the search history button, allowing users to view search history by clicking the button.

2.4.4.9 Prerana and Varshi Day 3

- Commit



prerana varshi day 3

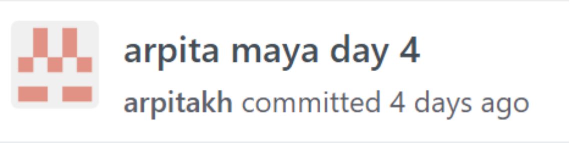
bachuv committed 5 days ago

- In this pair programming episode, Prerana and Varshi added in the functionality to sort the frequency and author column on the paper list page.

- Refactoring Explanation
    - To sort the frequency and author column, two new button had to be added to the paper list on the paper list page. The first button would sort the list by ascending frequency when clicked and descending frequency when clicked again. The second button would sort the list by ascending author names when clicked and descending author names when clicked again. The previous code from Sprint 1 did not account for these additional buttons and on-click functionality that the buttons triggers on the paper list page. The previous code also did not have in the design a sorting algorithm that sorts the papers by word frequency or author names. Therefore, there were two instances of refactoring that took place in this pair programming episode. The first sniffed problem was the absence of the buttons that triggered the sorting of the frequency column and author column, and this problem was resolved by providing a button in the frequency column that sorts the papers by frequency (ascending on one click and descending on two clicks) and a button in the author column that sorts the papers by author names (A-Z on one click and Z-A on two clicks) on the paper list page. The second sniffed problem was that there was no priorly designed sorting algorithm that sorted the papers by frequency and author name, and this problem was resolved by setting up this algorithm to run across the searched papers.

2.4.4.10 Arpita and Maya Day 4

- Commit



arpita maya day 4
arpitakh committed 4 days ago

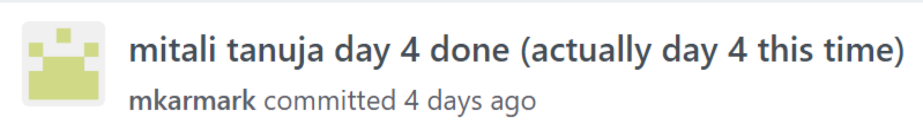
  - In this pair programming episode, Arpita and Maya added in the functionality to sort the title column on the paper list page.

- Refactoring Explanation

  - To sort the title column, a new button had to be added to the paper list on the paper list page. By adding this button, a click of the button can sort the list by ascending title names and another click of the button can sort the list by descending title names. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the button triggers on the paper list page. The previous code also did not have in the design a sorting algorithm that sorts the papers by paper title. Therefore, there were two instances of refactoring that took place in this pair programming episode. The first sniffed problem was the absence of a button that triggered the sorting of the title column, and this problem was resolved by providing a button in the title column that sorts the papers by title (A-Z on one click and Z-A on two clicks) on the paper list page. The second sniffed problem was that there was no priorly designed sorting algorithm that sorted the papers by title, and this problem was resolved by setting up this algorithm to run across the searched papers.

2.4.4.11 Mitali and Tanuja Day 4

- Commit



mitali tanuja day 4 done (actually day 4 this time)
mkarmark committed 4 days ago

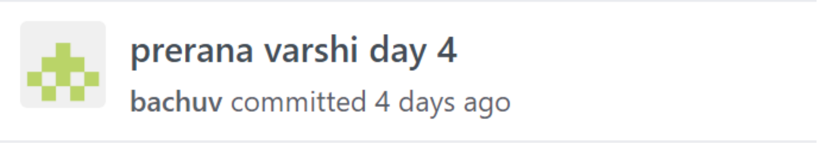
  - In this pair programming episode, Mitali and Tanuja added in the functionality to sort the frequency and author columns on the conference list page.

- Refactoring Explanation

  - To sort the author column and frequency column, new buttons had to be added to the conference list on the conference list page. A click of one button can sort the list by ascending frequency on one click and descending frequency on another and a click of the second button can sort the list by ascending author names and another click of the button can sort the list by descending author names. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the buttons triggers on the conference list page. Therefore, this was an example of refactoring because the sniffed problem was the absence of a button that triggered the sorting of the author column and the absence of a button that triggered the sorting of the frequency column, and this problem was resolved by providing a button in the author(s) column that sorts the papers by author (A-Z on one click and Z-A on two clicks) and a button in the frequency column that sorts the paper by frequency (ascending on one click, descending on two clicks) on the conference list page.

2.4.4.12 Prerana and Varshi Day 4

- Commit



prerana varshi day 4

bachuv committed 4 days ago

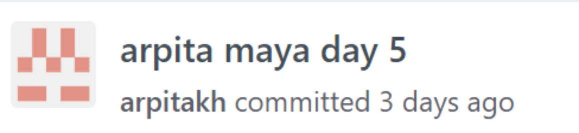
  - In this pair programming episode, Prerana and Varshi added in the functionality to sort the conference column on the paper list page.

- Refactoring Explanation

  - To sort the conference column, a new button had to be added to the paper list on the paper list page. By adding this button, a click of the button can sort the list by ascending conference names and another click of the button can sort the list by descending conference names. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the button triggers on the paper list page. The previous code also did not have in the design a sorting algorithm that sorts the papers by conference names. Therefore, there were two instances of refactoring that took place in this pair programming episode. The first sniffed problem was the absence of a button that triggered the sorting of the conference column, and this problem was resolved by providing a button in the conference column that sorts the papers by conference (A-Z on one click and Z-A on two clicks) on the paper list page. The second sniffed problem was that there was no priorly designed sorting algorithm that sorted the papers by conference names, and this problem was resolved by setting up this algorithm to run across the searched papers.

2.4.4.13 Maya and Arpita Day 5

- Commit



arpita maya day 5

arpitakh committed 3 days ago

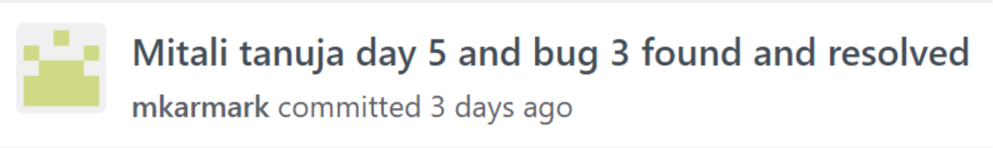
  - In this pair programming episode, Arpita and Maya added in the functionality to sort the title column on the conference list page.

- Refactoring Explanation

  - To sort the title column, a new button had to be added to the paper list on the conference list page. By adding this button, a click of the button can sort the list by ascending title names and another click of the button can sort the list by descending title names. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the button triggers on the conference list page. The previous code also did not have in the design a sorting algorithm that sorts the papers by paper title. Therefore, this is an example of refactoring because the sniffed problem was the absence of a button that triggered the sorting of the title column, and this problem was resolved by providing a button in the title column that sorts the papers by title (A-Z on one click and Z-A on two clicks) on the conference list page.

2.4.4.14 Mitali and Tanuja Day 5

- Commit



Mitali tanuja day 5 and bug 3 found and resolved
mkarmark committed 3 days ago

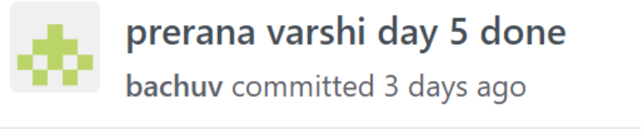
  - In this pair programming episode, Tanuja and Mitali added the functionality of adding accurate frequencies of the word occurrences in the paper in the frequency column on the paper list page.

- Refactoring Explanation

  - In Sprint 1, frequencies reported in the frequency column of the paper list page reported the number of occurrences of the searched term in the *abstracts* of the searched papers. In reality, we needed to change this to the number of occurrences of the searched term in the *full text* of the searched papers. With the design of the last sprint, we were not extracting the text of the full papers. Thus to display accurate frequencies, we had to extract the text of the PDF document, and then tabulate the frequencies of the desired terms. Therefore, this is an example of refactoring. The sniffed problem was the inability to process the full text of a paper to tabulate accurate frequencies, and the problem was solved by designing a new way to extract the text from the PDF and tabulate correct frequencies while searching for the papers.

2.4.4.15 Prerana and Varshi Day 5

- Commit



prerana varshi day 5 done
bachuv committed 3 days ago

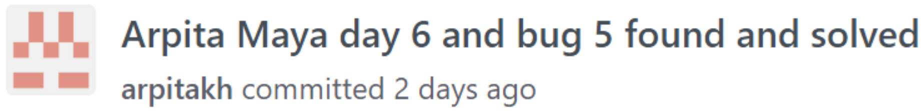
  - In this pair programming episode, Prerana and Varshi added in the functionality to sort the conference column on the conference list page.

- Refactoring Explanation

  - To sort the conference column, a new button had to be added to the paper list on the conference list page. By adding this button, a click of the button can sort the list by ascending conference names and another click of the button can sort the list by descending conference names. The previous code from Sprint 1 did not account for this additional button and on-click functionality that the button triggers on the conference list page. Therefore, there was an instance of refactoring in this pair programming episode. The sniffed problem was the absence of a button that triggered the sorting of the conference column, and this problem was resolved by providing a button in the conference column that sorts the papers by conference (A-Z on one click and Z-A on two clicks) on the conference list page.

2.4.4.16 Arpita and Maya Day 6

- Commit



Arpita Maya day 6 and bug 5 found and solved

arpitakh committed 2 days ago

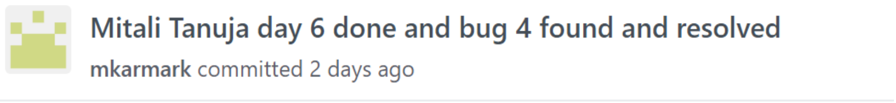
  - In this pair programming episode, Arpita and Maya measured the progress of the page during loading.

- Refactoring Explanation

  - To measure the progress of the page loading, communication had to occur between the browser and the page. This was not necessary and thus did not exist in Sprint 1. To accomplish this in this sprint, we introduced a means of communication between the browser and the page that accurately measured the percent a web page is loaded. This is therefore an example of refactoring. The sniffed problem was the lack of communication between the browser and web page, and the problem was resolved by introducing this communication in the design.

2.4.4.17 Mitali and Tanuja Day 6

- Commit



Mitali Tanuja day 6 done and bug 4 found and resolved

mkarmark committed 2 days ago

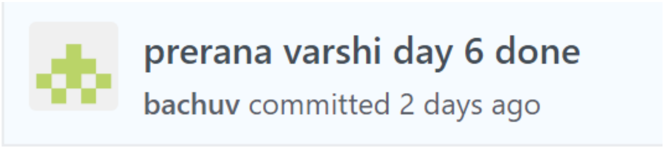
  - In this pair programming episode, Mitali and Tanuja added the feature of re-searching for a subset of papers from the paper list page.

- Refactoring Explanation

  - To search a subset of papers from the paper list page, there needed to be a mechanism for indicating the desired subset. For this checkboxes were needed in each row. These checkboxes did not exist in Sprint 1, but since they were the only way to collect this subset data from the user, they were introduced into the design to complete this task. Once the desired paper list is selected, the papers need to be searched for specifically. This specific search mechanism also did not exist in Sprint 1 and had to be designed in Sprint 2. There were thus two instances of refactoring in this pair programming episode. The first sniffed problem was that there was no means for the user to select a subset of papers, and this problem was resolved by providing the users with checkboxes in each row. The second sniffed problem was that there was no means of searching for specific papers, and this problem was resolved by adding a new search query with which to search by paper through ACM and IEEE.

2.4.4.18 Prerana and Varshi Day 6

- Commit



prerana varshi day 6 done

bachuv committed 2 days ago

  - In this pair programming episode, Prerana and Varshi designed a status bar to use during page loading.

- Refactoring Explanation
  - To create a status bar, a status bar GIF has to be visible during the loading period. This GIF was not necessary in Sprint 1 and thus did not exist in Sprint 1. To accomplish this in this sprint, we introduced a new page that displays a GIF when the browser is loading. This is therefore an example of refactoring. The sniffed problem was the absence of a needed GIF during page loading, and the problem was resolved by introducing this status bar GIF in the design.

2.4.5 Sustainable Pace

Sustainable pace was a very important factor when we were doing sprint planning. During sprint planning, we split up the tasks for each sprint in order to ensure that there was no chance of any team member burning out or working overtime. We also made sure that there was an equal balance of work to make sure that there is high quality code. The sustainable pace ensured that we were working efficiently and to our full capacity. This means we ensured that we didn't work overtime when we were exhausted and tired.

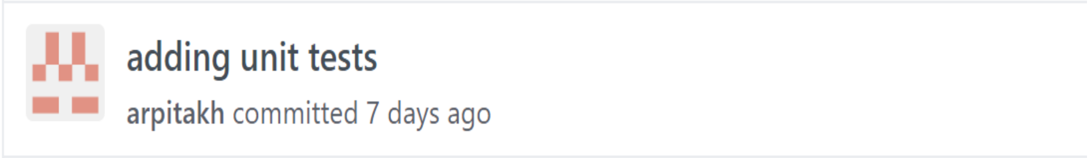
This link shows the final burndown chart, which shows how we gradually completed the tasks: https://github.com/mkarmark/USC_CSCI_310_Project_2/blob/master/Sprint2/Burndown%20Chart.pdf

Finally, the code commits in the following section will also serve as evidence to show our incremental development in order to preserve sustainable pace.

2.4.6 Code Commits and Collective Code Ownership

The following section describes the XP practice of collective code ownership with respect to all the code commits that we made in this sprint. In other words, we made sure that programmers don't work on the same code that they worked on in the previous sprint to ensure that the practices of collective code ownership are upheld and any developer can change code to add functionality, fix bugs and improve designs or refactor.

2.4.6.1 Maya and Arpita Day 1



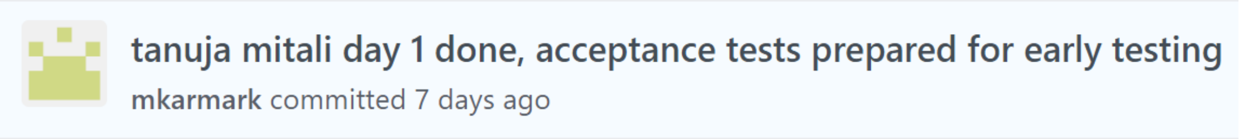
adding unit tests

arpitakh committed 7 days ago

In the image above, a commit from Arpita and Maya is shown. Their task for the day was to write unit tests for the new tasks in Sprint 2.

- Collective Code Ownership
  - Their task for the day upheld the collective code ownership principles since they hadn't written the white box tests in Sprint 1. Prerana and Varshi had worked on these white-box tests in the previous sprint.

2.4.6.2 Tanuja and Mitali Day 1



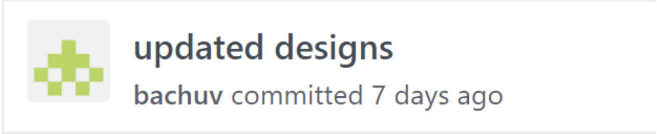
tanuja mitali day 1 done, acceptance tests prepared for early testing

mkarmark committed 7 days ago

In the image above, a commit from Tanuja and Mitali is shown. Their task for that day was to work on writing the new acceptance tests for the new tasks in Sprint 2.

- Collective Code Ownership
  - Their task for the day upheld the collective code ownership principles since they hadn't written black-box tests in Sprint 1. Maya and Arpita wrote the black-box tests in Sprint 1.

2.4.6.3 Prerana and Varshi Day 1



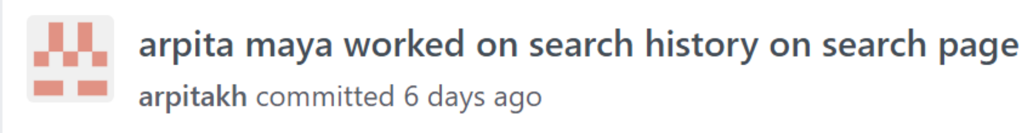
updated designs

bachuv committed 7 days ago

In the image above, a commit from Prerana and Varshi is shown. Their task for that day was to work on updating the designs, accounting for the new features in Sprint 2.

- Collective Code Ownership
  - Their task for the day upheld the collective code ownership principles since they hadn't worked on the designs in the last sprint. Mitali and Tanuja created the designs for the last sprint.

2.4.6.4 Arpita and Maya Day 2



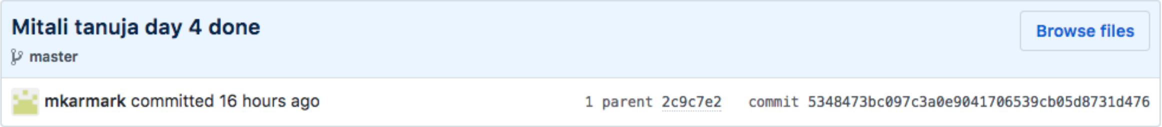
arpita maya worked on search history on search page

arpitakh committed 6 days ago

In the image above, Arpita and Maya's commit for the day is shown. Their task for the day was adding the search history feature on the search page.

- Collective Code Ownership
  - Their task for the day upheld the collective code ownership principles since they hadn't touched the search page in the last sprint. Prerana and Varshi had worked on the search page in the previous sprint.

2.4.6.5 Mitali and Tanuja Day 2

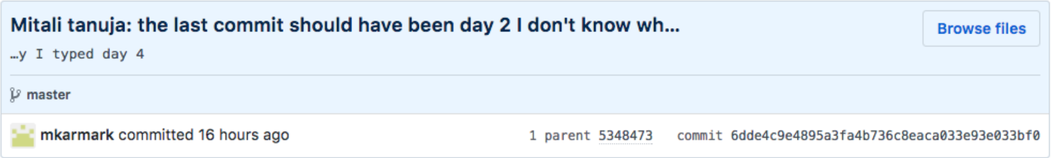


Mitali tanuja day 4 done

master

mkarmark committed 16 hours ago

```
1 parent 2c9c7e2    commit 5348473bc097c3a0e9041706539cb05d8731d476
```



Mitali tanuja: the last commit should have been day 2 I don't know wh...

```
…y I typed day 4
```

master

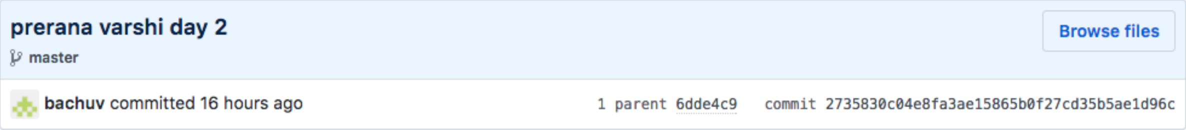
mkarmark committed 16 hours ago

```
1 parent 5348473    commit 6dde4c9e4895a3fa4b736c8eaca033e93e033bf0
```

In the image above, a commit from Tanuja and Mitali is shown. However, as seen from the above screenshots, a mistake was made by Mitali as she named the commit message "day 4 done" instead of day 2. But she made another commit to correct the error. Their task for the day was to add the search history feature on the paper list page.

- Collective Code Ownership
    - Their task for the day upheld the collective code ownership principles since they hadn't worked on the paper list page in the last sprint. Arpita and Maya had worked on the paper list page in Sprint 1.

2.4.6.6 Prerana and Varshi Day 2

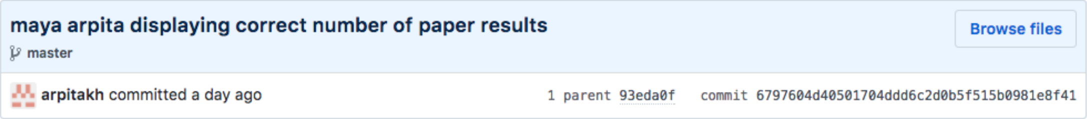


prerana varshi day 2
master
bachuv committed 16 hours ago 1 parent 6dde4c9 commit 2735830c04e8fa3ae15865b0f27cd35b5ae1d96c

In the image above, a commit from Varshi and Prerana is shown. Their task for the day was to add the search history feature on the word cloud page.

- Collective Code Ownership
    - Their task for the day upheld the collective code ownership principles since they hadn't worked on the word cloud page in the last sprint. Tanuja and Mitali had worked on the word cloud page in Sprint 1.

2.4.6.7 Maya and Arpita Day 3

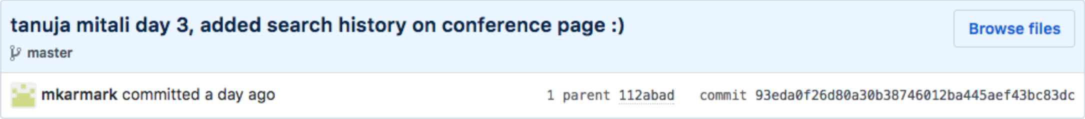


maya arpita displaying correct number of paper results
master
arpitakh committed a day ago 1 parent 93eda0f commit 6797604d40501704ddd6c2d0b5f515b0981e8f41

In the image above, a commit from Maya and Arpita is shown. Their task for the day was to give the user an option for the number of papers to generate a search on.

- Collective Code Ownership
    - Their task for the day upheld the collective code ownership principles since they hadn't touched the search page in the last sprint. Prerana and Varshi had worked on the search page in the previous sprint.

2.4.6.8 Tanuja and Mitali Day 3

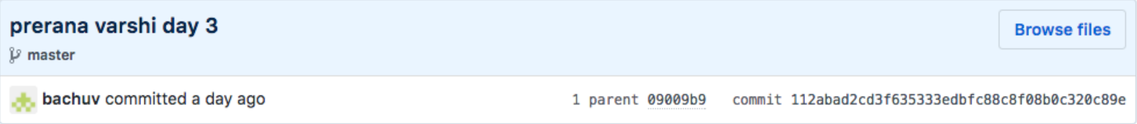


tanuja mitali day 3, added search history on conference page :)
master
mkarmark committed a day ago 1 parent 112abad commit 93eda0f26d80a30b38746012ba445aef43bc83dc

In the image above, a commit from Tanuja and Mitali is shown. Their task for the day was to work on the previously entered searches on the Conference List Page.

- Collective Code Ownership
    - Their task for the day upheld the collective code ownership principles as in the previous sprint, Varshi and Prerana had worked on the code to satisfy the requirement of accessing previously entered searches on the Conference List Page.

2.4.6.9 Prerana and Varshi Day 3

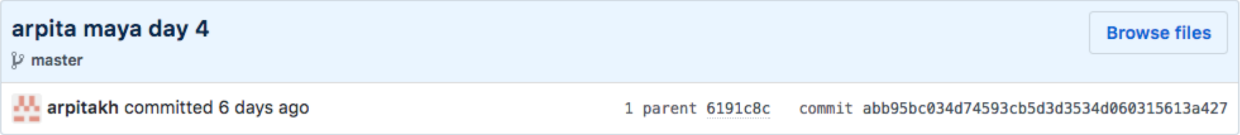


prerana varshi day 3
master
bachuv committed a day ago 1 parent 09009b9 commit 112abad2cd3f635333edbfc88c8f08b0c320c89e

In the image above, Varshi and Prerana finished the task of sorting by frequency author column alphabetically on Paper List Page.

- Collective Code Ownership
    - Through this task, Varshi and Prerana demonstrated collective code ownership because only Tanuja and Mitali had worked on the code for this task in the previous sprint and Varshi and Prerana were completely unaware of the specifics of this piece of code.

2.4.6.10 Maya and Arpita Day 4



arpita maya day 4

master

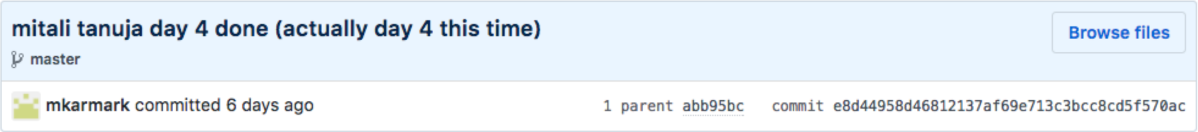
arpitakh committed 6 days ago

1 parent 6191c8c commit abb95bc034d74593cb5d3d3534d060315613a427

In the image above, it shows Maya and Arpita committing code for day 4. Their task was to sort by title column alphabetically on Paper List Page. They finished their task at the appropriate time.

- Collective Code Ownership
    - Through this task, Arpita and Maya demonstrated collective code ownership because only Tanuja and Mitali had worked on the code for this task in the previous sprint and Maya and Arpita were completely unaware of the specifics of this piece of code.

2.4.6.11 Tanuja and Mitali Day 4



mitali tanuja day 4 done (actually day 4 this time)

master

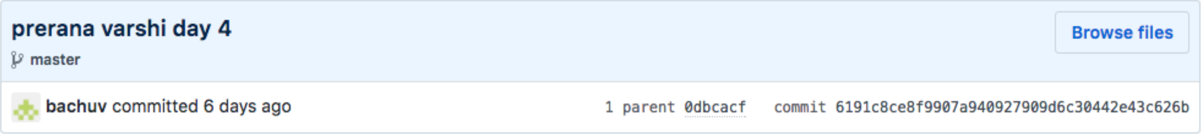
mkarmark committed 6 days ago

1 parent abb95bc commit e8d44958d46812137af69e713c3bcc8cd5f570ac

In the image above, Tanuja and Mitali completed their task for Day 4. They finished working on the task of sorting by author column alphabetically on Conference Page. They successfully completed this task.

- Collective Code Ownership
    - Through this task, Tanuja and Mitali demonstrated collective code ownership because only Maya and Arpita had worked on the code for this task in the previous sprint and Tanuja and Mitali were completely unaware of the specifics of this piece of code.

2.4.6.12 Prerana and Varshi Day 4



prerana varshi day 4

master

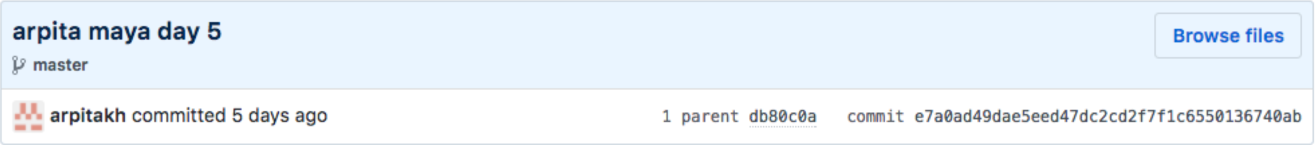
bachuv committed 6 days ago

1 parent 0dbcacf commit 6191c8ce8f9907a940927909d6c30442e43c626b

In the image above, a commit from Varshi and Prerana is shown. Their task for the day was to work on sort by conference column alphabetically on Paper List Page.

- Collective Code Ownership
    - Through this task, Varshi and Prerana demonstrated collective code ownership because only Tanuja and Mitali had worked on the code for this task in the previous sprint and Varshi and Prerana were completely unaware of the specifics of this piece of code.

2.4.6.13 Maya and Arpita Day 5



arpita maya day 5

master

arpitakh committed 5 days ago

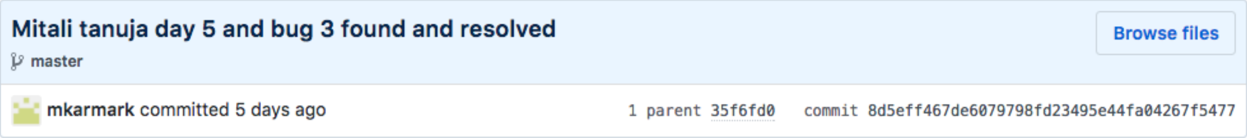
1 parent db80c0a commit e7a0ad49dae5eed47dc2cd2f7f1c6550136740ab

In the image above, a commit from Maya and Arpita is shown. Their task for the day was to work on sorting by title column alphabetically on Conference Page.

- Collective Code Ownership

  - Collective code ownership was shown as in the previous sprint the task of writing code to sort by title column alphabetically on Conference Page was done by Tanuja and Mitali. Prior to working on this code in this sprint Maya and Arpita had not worked on this piece of code at all.

2.4.6.14 Tanuja and Mitali Day 5

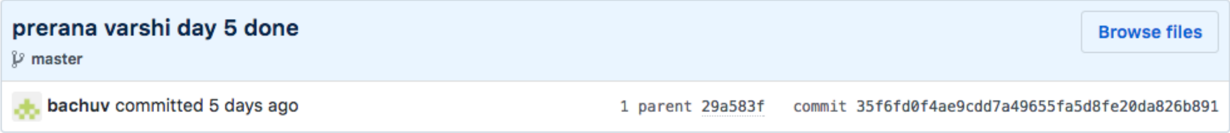


Mitali tanuja day 5 and bug 3 found and resolved
master
mkarmark committed 5 days ago 1 parent 35f6fd0 commit 8d5eff467de6079798fd23495e44fa04267f5477

In the image above, a commit from Tanuja and Mitali is shown. Their task for the day was to work on fixing the frequency column on the paper list page to show accurate frequency.

- Collective Code Ownership
  - Collective code ownership is shown as in the previous sprint, Prerana and Varshi had written code to fix the frequency column on the paper list page to show accurate frequency. Tanuja and Mitali had no prior experience of working on the piece code and it was completely new to them.

2.4.6.15 Prerana and Varshi Day 5

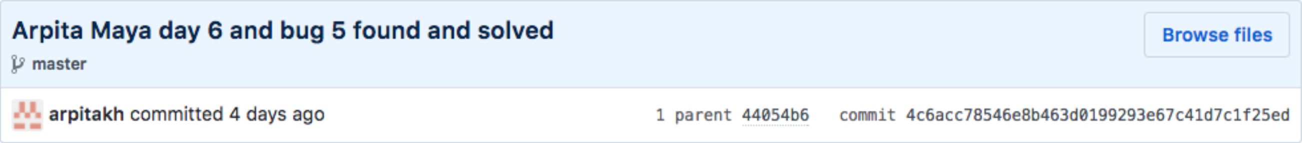


prerana varshi day 5 done
master
bachuv committed 5 days ago 1 parent 29a583f commit 35f6fd0f4ae9cdd7a49655fa5d8fe20da826b891

In the image above, a commit from Varshi and Prerana is shown. Their task for the day was to work on sorting by conference column alphabetically on Conference Page

- Collective Code Ownership
  - Collective code ownership was shown as in the previous sprint, Tanuja and Mitali had written code for the task of sorting by conference column alphabetically on Conference Page.

2.4.6.16 Maya and Arpita Day 6

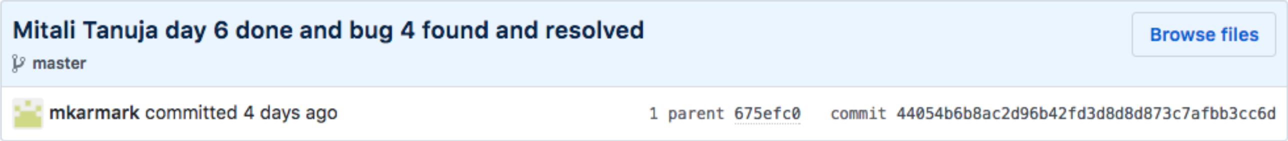


Arpita Maya day 6 and bug 5 found and solved
master
arpitakh committed 4 days ago 1 parent 44054b6 commit 4c6acc78546e8b463d0199293e67c41d7c1f25ed

In the image above, a commit from Maya and Arpita is shown. Their task for the day was to measure the loading progress of the page.

- Collective Code Ownership
  - This was a new task with no prior relevant code in Sprint 1. Therefore, Maya and Arpita touched code no one, including themselves, was familiar with the code.

2.4.6.17 Tanuja and Mitali Day 6



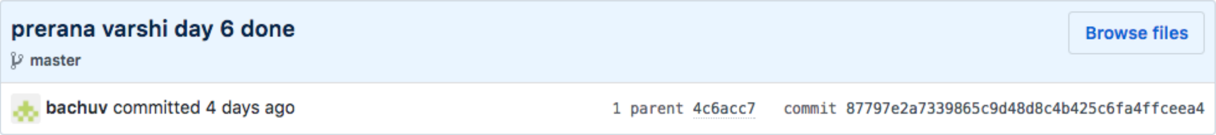
Mitali Tanuja day 6 done and bug 4 found and resolved
master
mkarmark committed 4 days ago 1 parent 675efc0 commit 44054b6b8ac2d96b42fd3d8d8d873c7afbb3cc6d

In the image above, a commit from Tanuja and Mitali is shown. Their task for the day was to work on allowing users to select a subset of papers on the paper list page to generate a new word cloud from.

- Collective Code Ownership

  - Maya and Arpita made and filled accurately the columns of the paper list page in the previous sprint. Therefore, Tanuja and Mitali had not worked on this piece of code before this sprint.

2.4.6.18 Prerana and Varshi Day 6



prerana varshi day 6 done

master

bachuv committed 4 days ago

1 parent 4c6acc7 commit 87797e2a7339865c9d48d8c4b425c6fa4ffceea4

In the image above, a commit from Varshi and Prerana is shown. Their task for the day was to work on creating a status bar for page loading periods.

- Collective Code Ownership
  - This was a new task with no prior relevant code in Sprint 1. Therefore, Prerana and Varshi touched code no one, including themselves, was familiar with the code.