Homework #4

CECS 550

Dr. Ragade

April 17, 2014

Team WikiMapper

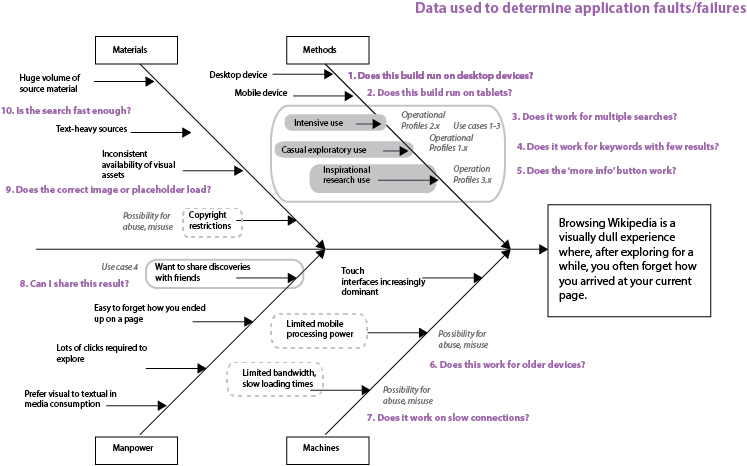
Ashley Revlett

Everett Rush

Sarah Mullins

**Task 1)**

**Diagram 1. Data Used to Determine Application Faults/Failures**



The statistics we gathered to determine faults and failures in our application are highlighted above. Each question was a point of failure that we identified.

**Table 1a. Function Points, Errors and Hours**

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **ID #** | **Function Descrip.** | **Use Case** | **Func. Pts.** | **Error IDs** | **Effort - AR** | **Effort - ER** | **Effort - SM** | **TOTAL HOURS** | **Hrs/ Function Pt** |
| **0** | planning, research and architecture design | n/a | n/a | 1 and 19 | 18.5 | 2 | 13 | 33.5 | n/a |
| **1** | create window | 1 | 1 |  |  | 1 |  | 1 | 1.00 |
| **2** | accept user search terms | 1 | 3 | 21 | 1 | 8 |  | 9 | 3.00 |
| **3** | display nodes | 2 | 5 | 5,6,7,9,10,13,14 |  | 35.5 |  | 35.5 | 7.10 |
| **7** | create model to store data | 2 | 12 |  | 3 |  |  | 3 | 0.25 |
| **12** | retrieve page data asynchronously | 2 | 12 |  | 17 |  |  | 17 | 1.42 |
| **10** | parse page for most important links | 2 | 7 | 18,19,20, 22, 23 |  |  | 38 | 28 | 4.00 |
| **11** | extract image from page correctly | 2 | 5 |  |  |  | 8 | 4 | 0.80 |
| **8** | record data from network | 2 | 7 |  | 6 |  |  | 3 | 0.43 |
| **6** | pass messages between display, model, and network | 1 | 12 | 3,17,24 | 6 | 4.5 | 5.5 | 16 | 1.33 |
| **5** | button interactions | 1 | 7 |  |  | 20 |  | 16 | 2.29 |
| **4** | display more info box | 3 | 7 | 8 and 12 |  | 8 | 5 | 12 | 1.71 |
| **13** | sanitize query | 2 | 3 | 16 | 0.5 |  |  | 0.5 | 0.17 |
| **14** | handle network errors | 2 | 5 | 15 | 1 |  |  | 1 | 0.20 |
| **15** | share on facebook link | 4 | 7 |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  | Total: | Avg: |
| **TOTALS** |  |  | **93** |  | **53** | **79** | **69.5** | **201** | **1.82** |

**Table 1b. Use Cases IDs referenced in Table 1a**

|  |  |  |
| --- | --- | --- |
| **Use Case ID #** | **Use Case Description** | **Priority** |
| **1** | User searches for Wikipedia article | 1 |
| **2** | User browses related Wikipedia articles | 2 |
| **3** | User views more detailed information about Wikipedia article | 3 |
| **4** | User shares Wikipedia article on social media | 4 |

**Table 1c. Error IDs referenced in Table 1a**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **ID** | **Github ID** | **Error Cat.** | **Error Description** | **Location** | **Est. Hours to fix** |
| **1** |  | 2 | UI Architecture doesn’t make sense. | Display | 50 |
| **2** |  | 10 | UI Team broke Network’s working code | Network | 0.2 |
| **3** |  | 2 | Parent-Child relationship was left out of architecture | Controller, Model, View | 2 |
| **4** |  | 11 | Parser returns incorrect set | Parser | 0.75 |
| **5** |  | 2 | UI Node Mouse Movement Bug | Display | 2 |
| **6** |  | 2 | UI Controls Translation during panning | Display | 2 |
| **7** |  | 2 | UI Controls don't Scale correctly | Display | 2 |
| **8** |  | 2 | Cannot Remove Non Type UI pop up | Display | 4 |
| **9** |  | 2 | Pre-existing Node parent bug | Display | 5 |
| **10** |  | 2 | The Picture moves faster than background | Display | 3 |
| **11** |  | 2 | Summary Pop Up doesn’t appear and throws exception | Display | 2 |
| **12** |  | 2 | RstDocument is an inappropriate choice for Text Summary Pop Up | Display | 2 |
| **13** |  | 2 | The Page Layout causes the scatter plane to reset to (0,0) | Display | 2 |
| **14** |  | 2 | The text box distorts text | Display | 2 |
| **15** |  | 11 | Network doesn't handle zero search results case | Network | 2 |
| **16** |  | 11 | Network doesn't fully sanitize and urlencode query | Network | 1 |
| **17** |  | 2 | Parser accepts incorrect parameters | Parser | 0.5 |
| **18** | 9 | 11 | Import strip\_tags in parser fails | Parser | 0.25 |
| **19** | 8 | 2 | Parser class architecture refactoring needed | Parser | 2 |
| **20** | 7 | 2 | Parser parameter change | Parser | 0.5 |
| **21** | 5 | 11 | App crashes when search field is clicked | Display | 0.5 |
| **22** | 4 | 11 | Codec unable to encode character | Parser | 1 |
| **23** | 3 and 1 | 11 | Set not removing duplicates | Parser | 3 |
| **24** | 2 | 11 | Module import failures | All | 5 |
|  |  |  |  | **TOTAL** | **94.7** |

**Table 1d. Error Codes referenced in Table 1c.**

|  |  |
| --- | --- |
| **Error Code** | **Error Descrip.** |
| **1** | Library Limitations |
| **2** | Architectural Design |
| **3** | UI Requirement Change |
| **4** | Module Miscommunication |
| **5** | Network Communication |
| **6** | Security Vulnerabilities |
| **7** | Legal Responsibilities |
| **8** | Hardware Limitations |
| **9** | Performance |
| **10** | Lack of Communication between Teams |
| **11** | Implementation Errors |

In Homework 1, we estimated a total of 191 hours required to complete the project. As of April 16, we have spent 201 hours so far, with the application about 75% complete. We are obviously going to spend more time than the hours initially estimated before the project is complete. We did use a formal model – the agile method of allocating story points to form an estimate. We broke the requirements down into large chunks of user-required functionality, assigned each chunk story points based on task complexity, then estimated the time required to complete each chunk of functionality based on those points.

Based on our experience this project and the data we’ve collected, a few observations are possible using data mining techniques, but since our dataset is relatively small, simple deduction will suffice. Lesson one, user interface development is hard, and UI functionality should be broken down into very small tasks for planning, estimation and testing. Lesson two, initial design time takes longer than estimated, and will need to be revisited at least once later during development, as the architecture evolves due to development needs. Using our data we can also see an emergence of certain classes of developer time use – people involved in planning and architecture invest a lot of time up front in research and design, but their time tapers off as others implement the design. Similarly, developers who are more involved with testing and QA will spend little time in the beginning, but that time increases as the project progresses.

**Task 2)**

I)

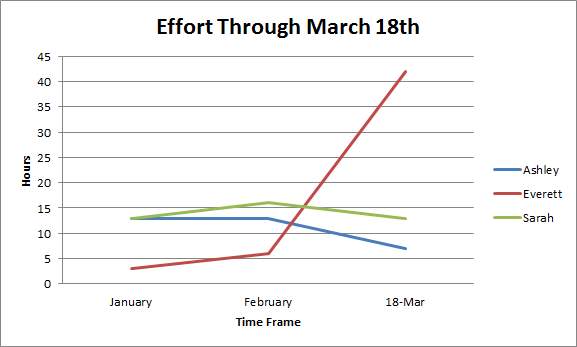
The following table shows the effort hours spent by each team member by month:

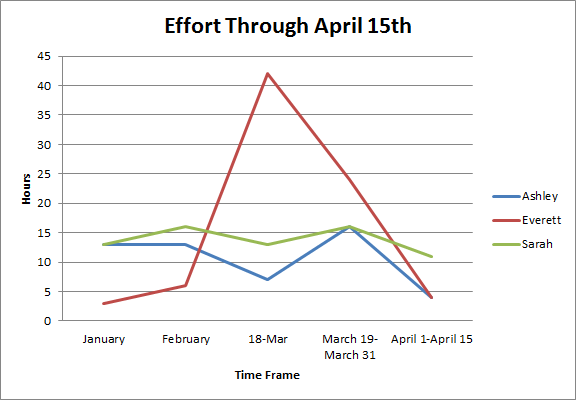
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Month | Jan. | Feb | Mar18 | Mar19-Mar31 | Apr1-Apr15 | **Total** |
| Ashley | 13 | 13 | 7 | 16 | 4 | 53 |
| Everett | 3 | 6 | 42 | 24 | 4 | 79 |
| Sarah | 13 | 16 | 13 | 16 | 11 | 69 |
| **Sub-Total** | 29 | 35 | 62 | 56 | 19 |  |

**Total effort through March 18:** 126 hours

**Total effort through April 15:** 201 hours

We did not track testing effort hours separately, but based on the code composition and total effort hours shown below, we can estimate that approximately 16 hours were spent on testing.

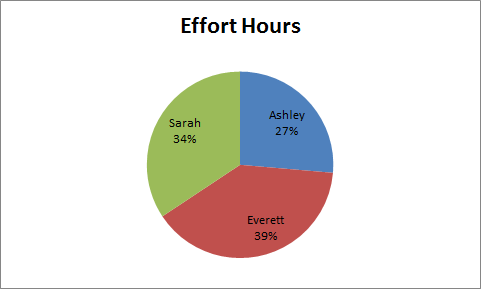




The total effort hours was close to the number predicted at the beginning of the project (184) hours. We can find the MMRE (Mean Magnitude of Relative Error) with the formula

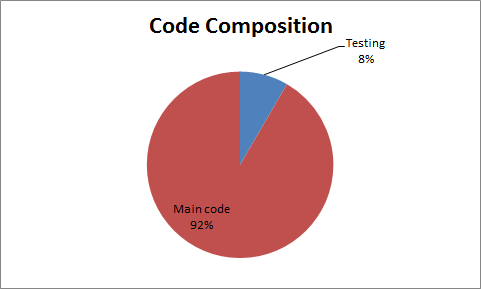
|Actual Effort - Estimated Effort| / Actual Effort = 0.09

The following pie-chart breaks down the total effort hours by team member:



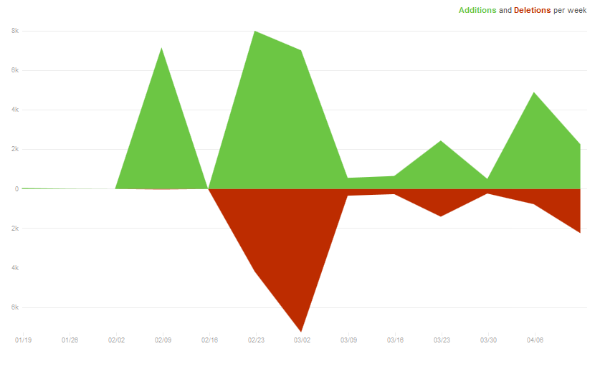
Each team member made a fair contribution to the project, considering the nature of their work. Everett was given the responsibility of creating the UI for Wikiviz; this required learning the intricacies of the open source library Kivy, which very likely contributed to his increased effort hours. Ashley was the team leader, which required time for tasks above and beyond those of her portion of the project, the network. Sarah’s main task (the text parser) was relatively straight-forward, but additional time was needed to become more familiar with Python.

The following pie-chart shows the code composition of Wikiviz:



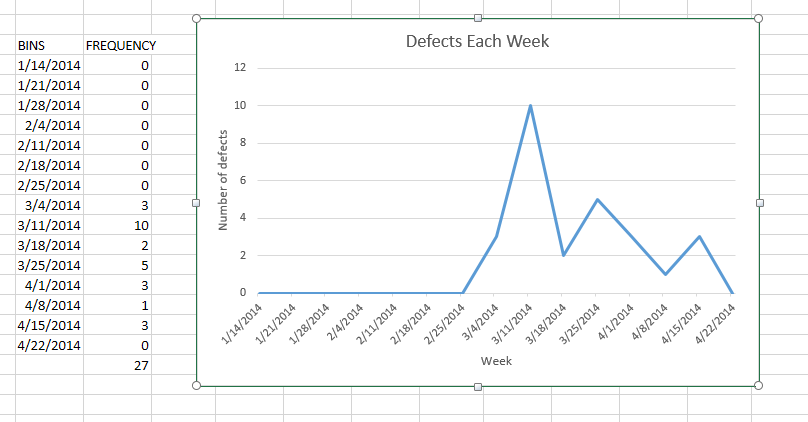
At the time of writing, there were 90 lines of test-stub and driver code, and 983 lines of regular code, with a total of 1,073 lines.

From the code frequency graph available on WikiViz shown below, we can estimate that the maximum code occurred at around February 23rd, at 3,000 lines.

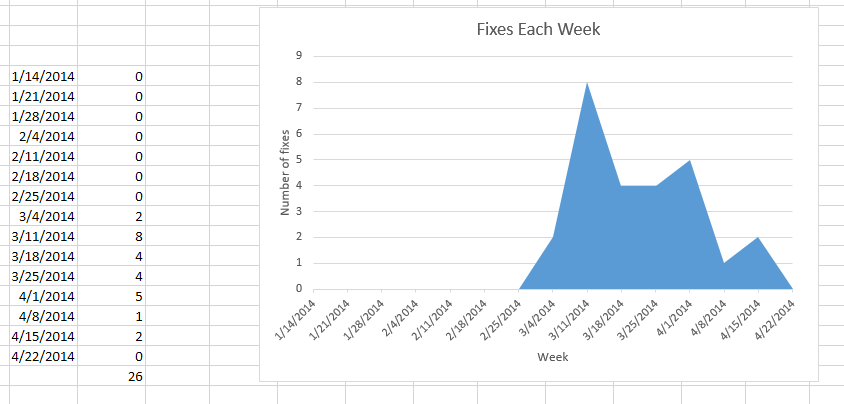


**Task 3)**

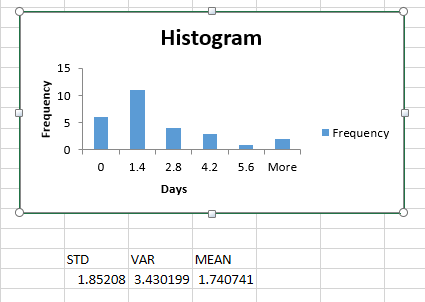
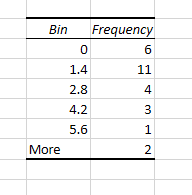
**This chart shows the week in which a bug defect was reported.**



**This chart shows the week in which bug issues were closed.**



**The chart below is a histogram that groups our defect response time into bins and tallies the groups.** We had a very quick response defect resolution time. The reason was because each team member worked in bursts when they had time or on the weekends. So we were able to report a bug and fix the bug, sometimes in the same sitting or on the next day.



There were three instances where bug defects were not fixed within two standard deviations of the mean. Two occurred during the period preceding Test 2, and one instance occurred when Ashley was on spring break.

We had a **very low recall**, 10/30= .33. After we developed our initial idea, we thought that we had a plan that we could execute without much or any change. We had some idea that the UI would be adapted as we become more proficient using kivy, but we had no idea the extent of the change that had to be made. The entire UI had to be redesigned. The classes had to be modified and aggregation had to be rethought. The methods were the only things that stayed the same. The model only had two minor changes that we were unable to predict. We needed a way to associate a parent with its child. While it was simple to add this association, the change spanned across multiple classes.The parser experienced a substantial amount of change as well.

We had a **very high precision**, 1.0. We knew that various parts of the UI and parser would have to be changed when we started the implementing. We predicted 2 classes and 3 methods from the parser would need to be changed, and we predicted 5 classes and the interface of the UI would need change. All of these items were changed from their original designs.

In roughly 3000 lines of code and spanning twelve configuration documents we detected 27 defects. It is very likely that there are substantially more bugs left in the code than we discovered. I estimate that we have not found even half the defects in the software. I estimated this by taking into account the experience of the developers, the scope of the project, the amount of time spent testing, and the amount of time spent on the project as a whole.

**Task 4) Everett**

II). See the timeline in document 4.5. Our effort estimation was very close to the actual amount of effort that the project took.

|  |  |  |
| --- | --- | --- |
|  | Estimated (in hours) | Actual (in hours) |
| Sprint 1 (1/27-2/7)  (research) | 30 | 20 |
| Sprint 2 (2/7-2/21)  (research and design) | 30 | 35 |
| Sprint 3 (2/24-3/7)  ( top five user stories) | 53 | 62 |
| Sprint 4 (3/7-3/21)  (remaining user stories) | 22 | 40 |
| Sprint 5 (3/21-4/4)  (testing) | 37 | 35 |

We did not spend enough time upfront in sprint 1 doing research. This was a very poor decision on our part because the design process injected two major defects. These defects were fixed early on in the implementation phase. But the cost of the effort required to fix these defects was very high. Because the bug was present in the project for so long, we had many hours of work that went to waste. Many things we had done after the bug injection but before the fix could not be integrated into the new design. Consequently we had to increase the number of hours spent during milestone 3 and 4 to compensate for lost time. We are anticipating another burst of activity during this final sprint so that we can have a functional project to demo.

Our configuration items are the SRS, the homework, the diagrams, our docs folder, the Readme, the functional tests, the unit tests, the model code, controller code, network code, the main loop code, and the images. Overall we had 3 baseline versions. Each time we refactored after adding additional functionality we created a new version. Each configuration item had numerous patches in between each new version.

Our testing was performed on each of the three major components of the project: the model, controller, and view. The model and controller had both functional tests, unit tests, and integration tests. The controller has a two important modules, the parser and the network. Unit testing was done on all the classes in both modules. The UI was tested using user verification.

II) Tabulate software quality assurance metrics as of April 16, 2014.

|  |  |
| --- | --- |
|  | Evaluation |
| sufficiency | 8.6 |
| robustness | 5 |
| flexibility | 5 |
| reusability | 10 |
| efficiency | 5 |
| reliability | 5 |
| scalability | 5 |
| security | 6 |

**NOTES:**

**Sufficiency =** 13/15 user stories

**Reusability:** Both the parser and ui can be easily reused in other contexts. The parser extracts important links. That could be useful in many web projects. The UI graphs parent-child relations. It could help visualize any hierarchical data.

Efficiency: The model uses a linear search through the node list. Node creation follows a very puzzling route before a node is added to the ui.

Robustness: Many methods do not check for None parameters or wide ranges of data.

**Task 5)**

The most critical stage in the process was planning and design. By choosing Kivy for a UI framework, we received both benefits and disadvantages that affected the entire project. As this was the first time any of our team had used Kivy, those disadvantages were not well understood until they were directly encountered during implementation. Kivy uses its own Python interpreter, which caused problems when all team members tried to set up their development environments. This could be solved in the future by using a tool designed for synchronizing development environments, like a shared virtual machine or Vagrant.

Github and email proved to be a satisfactory way for us to manage the project, after some team members overcame an initial lack of familiarity with git. Github’s issue tracking and commenting system were helpful for group communication and documentation. There was occasional confusion over meeting schedules, which could be easily remedied by a shared calendar in the future.

We do have unfinished work that could be taken over by another team. Our documentation is supportive of this task, but it needs to be updated to reflect the architectural changes we’ve made during development. Our discussions and issues list on Github will provide supporting information that is more up-to-date than the SDS. The SRS, however, would still be very beneficial as it is still current.

We are treating this project as open-source software, rather than packaged, proprietary software. The source code is available on Github for anyone interested in reviewing, modifying, or maintaining the codebase. Because realistically our team members are not going to be able to continue development on this project after this semester, users will have to patch any bugs and address any maintenance issues themselves. If anyone does so, we will merge their patch into the source so that new users may benefit. To upgrade, users will have to visit the Github page and download the latest package.

Our application doesn’t collect any personal or sensitive data, so our security concerns were limited. We do interact with Wikipedia, so we took care to meter our API requests so we don’t exceed their usage limits. We also sanitized user input so no malicious API requests could be sent using our application.

Potential maintenance of the project should be minimal, assuming the Wikipedia API remains stable. To support new touchscreen devices, a new version of Kivy may eventually need to be merged into the source, and certain code updated. How much effort this would require depends on how much the Wikipedia API and/or Kivy changes. It could be significant or trivial.

In its current build, our application is not reliable. It is still under heavy development and new bugs are discovered whenever new features are added. In the next couple weeks, we expect that development cycle to slow down and the application to become much more reliable as we test more extensively and eliminate bugs.

**Homework 4.5**

