Recipe Finder Application

Project Legacy

COP 4331, Fall, 2013

Modification history:

Version	Date	Who	Comment
v0.0	11/12/13	Team	Initial SetUp and entry
v1.0	11/15/13	R. Kinner	Entry
v2.0	11/18/13	M. Bald	Updated analysis
v3.0	11/23/13	J. Barnett	Addition of roles and role percentages
v4.0	11/25/13	M. Bald, R. Kinner	Updated analysis

Team Name: Recipe Bytes

Team Members:

- Rachel Kinner rachel.kinner@knights.ucf.edu
- Vincenzo Marconi vincenzo.marconi117@knights.ucf.edu
- Ronald Hyatt enishi@knights.ucf.edu
- Matt Bald mbald@knights.ucf.edu
- Brian McCormick mccor140@knights.ucf.edu
- Josh Barnett jbarnett@knights.ucf.edu

Roles

Member	% Done	% Done	% Done	% Done	% Done	% Done

	ConOps	Project Plan	SRS	PMP	Design	Code
Josh Barnett	16.66%	4%	4%	4%	16.66%	0%
Matt Bald	16.66%	4%	4%	4%	16.66%	10%
Rachel Kinner	16.66%	40%	40%	40%	16.66%	0%
Vincenzo Marconi	16.66%	44%	44%	44%	16.66%	25%
Ronald Hyatt	16.66%	4%	4%	4%	16.66%	25%
Brian McCormick	16.66%	4%	4%	4%	16.66%	40%

Member	% Done Code Documentation	% Done CM	% Done QA	% Done Final Deliverables
Josh Barnett	0%	16%	65%	25%
Matt Bald	0%	16%	20%	20%
Rachel Kinner	0%	16%	0%	20%
Vincenzo Marconi	0%	16%	0%	20%
Ronald Hyatt	10%	16%	5%	0%
Brian McCormick	90%	20%	10%	5%

Team Member Roles:

Member	Role Description
Vincenzo Marconi	● Role: Project Manager, Website maintainer, Documenter ● Skills: Fluent in HTML/CSS/JavaScript and C#
Josh Barnett	● Role: Tester, legal consultant ● Skills: Troubleshooting, Data Inputing, software testing
Ronald Hyatt	● Role: Programmer, trainer, and involved in designing ● Skills: Programmer, acceptance testing
Matt Bald	● Role: Tester, Database updater, documentation writer and food taster

	• Skills: Patience, debugging, unit & acceptance testing
Rachel Kinner	• Role: Programmer, food consultant, database designer • Skills: Communication skills, C# knowledge, organization
Brian McCormick	• Role: Programmer, Librarian, and Source Control Repositor manager • Skills: Knowledge of Git, C#, and programmer

---

Analysis

- Recipe Finder saves users time when searching for nourishing recipes. The software is available for and reliable when run on a modern Windows operating system on a computer having at least a 2.33 GHz dual-core processor and 1 GB of RAM. Since we used a Phased Model of development, Recipe Finder can be updated and maintained as long as developers desire. Recipe Finder executes a search in less than 1 second so that users have minimal wait time. In the event of a user having a specific food allergy, the allergen filter eliminates all recipes that contain the selected allergen. A food safety module can also be launched from the ingredient selection page. The module contains helpful cooking information to ensure that users do not become ill after eating improperly prepared food.

- The recipe finder application was created with college students and young adults in mind as primary users, but any person able to use a computer can use the application. The recipe finder application should be used when a user needs a recipe that uses items they already have through the ingredient search. The application can also be used to find good, healthy recipes through the browse function.

- Known Problems: Certain ingredients not bringing back results: milk, ground chicken.

- Our group did not adhere well to our initial project plan. Our project estimates were fairly accurate, but we did not account for the extreme demands on our time considering external factors. The application is a good reflection of the initial design, however, necessary changes were made. Due to the logistics of programming the project, we abandoned the idea of a SQL database pretty quickly, instead opting to have the program read in all necessary data from a text file. We also, to keep the project relatively simple, chose to forgo the idea of an interactive website, and instead made a desktop application. Although we were unfortunately unable to have almost all of our “would-like-to-have” features, the team shifted design to focus on the features we found of core importance: a simple, solid, and comprehensive search function, along

with a focus on food/user safety.

- Defect Analysis: As with any software development, we ran into our fair share of bugs, problems, and defects. These issues ranged from very minor and easily correctable (i.e. some typos on the ingredient list), to complete headaches. Early on in the development cycle, there were a few notable bugs. The first issue we had was that GUI did not scale properly when maximizing the window. Initially we had planned to make this scale both horizontally and vertically, but after encountering this glitch where it did not scale either way (and trying to correct it), we found that the application looked much more professional when it would only scale vertically. Another issue that we had early on in the development cycle, and our first fatal error at that, was that the program crashed when unchecking an allergen box. This was before allergen functionality was even implemented (just the GUI display), which made it infinitely easier to hunt and kill that bug. As we got further along in the development cycle, we encountered several issues that appeared to be bugs, but ended up being typographical errors/oversights in the creation of our input files. (We initially thought that the allergen filter was working inconsistently, until it was discovered that several ingredients were simply not flagged with their corresponding allergen). As we neared the end of our development cycle, however, we began to uncover a very large problem in our program, and needed to resort to our "good-enough-to-ship" policy that had been laid out earlier in the semester. Although we wanted to include at least 100 recipes, we had an issue where only 31 would show up in our results when searched. After spending a long time looking through the search function, and an even longer time looking through the input files, we were unable to completely fix this error. Brian did, however, find a way to increase our recipe input-to-output ratio from 100-32 to 92-76 by removing some problem cases, and altering the recipe input function. We believe that there is a current issue in the search function preventing us from accessing the rest of the recipes. Due to time-constraints, and the invisibility of this issue to the user, we determined the program was good enough to ship, even with this known defect.

- The application was very simple, allowing us to rigorously test it. It goes without saying that with more time, there could have been more testing, but considering the time constraints our Quality Assurance was very comprehensive.

- Configuration Management was difficult for the group because of many challenges with BitBucket. We attempted to integrate our repository into Visual Studio, with little to no success, so we switched to running SourceTree. However, several team members had difficulty installing and running SourceTree software. The BitBucket tool was sufficient once all team members were able to access. In the future, we would recommend having a team session devoted to familiarizing the team with the configuration tool to allow for proper configuration management.

- For a project of similar size and scope, the processes would have been more efficient if a more

stringent timeline had been applied and followed. Better communication processes implemented at the beginning of the project would improve efficiency. We would suggest to future teams to settle on a single configuration management repository and code early. We would also suggest to future teams to allow more time than anticipated for coding. For a project roughly 10 times the scope, we would keep the idea of project update meetings. We would implement a standard communication method and more stringent deadlines. For a project 100 times the size and scope, we would implement managers of each category of work to manage the work being done in each phase.