DUCK
DEBUG

The Initial Technical Report
Golf Team November 2016

Developers

- **Maximilian Thomson**
- **Ben Solis**
- **Itay Grudev**
- **Alexander Hellings**
- **Tendekai Marawa**

Table of Content

Developers 1

Table of Content 2

THE PROJECT 3

THE CRITICAL USE CASE MODEL 3

Core Requirements: 3

Non-functional Requirements: 3

Functional Requirements: 4
- Sign up: 4
- Basic Functionality: 4

User Case Scenarios 5

THE SYSTEM 6

The Architectural Diagram 6

The Architecture Justification 7
- Why Ruby on Rails? 7
- Why Elastic Search? 7
- Why Use Load Balancers? 7
- Why PostgreSQL? 8

Three-Tier Architecture 8

The System Sequence Diagram 9

Duck Debug Entity Relationship Diagram 9

THE RISK ASSESSMENT 10

THE PROJECT PLAN 11
- The Plan 11
- Milestones 11
- The Initial Schedule 12

THE PROOF OF CONCEPT 13
- What has been completed so far? 13
- Problems we faced 13
- What's next? 13
- Conclusion 13

The Project

DuckDebug is an online service for getting your code debugged. The way it works is that you can post your project or snippet online and offer a bounty on getting it fixed. Developers can then fix people's code and claim those bounties.

Such a service can help people from being stuck on the same problem for hours or even days, or can help companies save money when their developers are stuck on a problem by having an expert look into their code.

Our service is different than other websites like Stack Overflow as they discourage people from posting code instead of asking useful questions. Our service is exactly the opposite and is oriented in fixing code.

The Critical Use Case Model

Core Requirements:

Users must be able to:

- Login and Sign up. Create a profile and set up billing information
- Upload Problems / Projects / Code
- Offer bounties on resolving their problems
- Upload solutions to problems
- View and filter available bounties based on project languages and technologies specific to the developer’s sill-set
- Input payment and cash out methods

Non-functional Requirements:

- HTML5 and CSS3 compatibility
- Page load time within 400ms (excluding connection limitations)
- Sign up on the website within 3 minutes
- Secure PCI DSS compliant payment methods
- Code upload of any size (free within reasonable limits - 100MB or more for a fee)
- Email notification when bounty was accepted or completed
- 95+% Service availability

Functional Requirements:

Sign up:

- User should be able to register and create an account
    - They should be able to enter their names
    - They should be able to enter their email addresses
    - They should be able to choose a username
    - They should be able to create a password within certain specified bounds
    - They should be able to input payment information for creating bounties for their own submitted code
    - They should be able to input payment information for receiving bounties for solving peoples issues
    - They should be able to choose whether to submit all payment info at a later date
    - Payment data should be stored securely as per the Payment Card Industry Data Security Standards (PCI DSS)
- User should be able to edit their own information
    - Edit passwords
    - Edit username
    - Edit payment information
    - Edit current email address

Basic Functionality:

- User should be able to view another users profile
    - They should be able to search for a specific user
    - They should be able to view a specific users profile page
    - They should be able to see important information pertaining to that user:
        - Avatar
        - Username
        - Reputation
- User should be able to submit code to be debugged
    - They should be able to create a title
    - They should be able to input a code fragment
    - They should be able to set a bounty for the code to be solved
    - They should be able to create a note to go along with the code
    - They should be able to edit these details at a later date (except for bounty)
- User should be able to cancel their own problem
- User should be able to see other people's code bounties
- User should be able to select a specific problem to view their code
- User should be able to submit a solve for a problem
    - They should be able to submit the completed code
    - They should be able to leave a note which may help explain what they have done
- User should be able to receive payment for completing another user's problem

User Case Scenarios

Courtney - 21 year old student with coding homework due by the end of the week with a bug which she doesn't understand.

She needs to be able to sign onto the website and upload or paste a snippet of code. She would like to receive Email or SMS Notifications as soon bounty is accepted or completed and would like a Secure Payment Method when her code has been fixed and the bounty had been claimed.

John - 36 year old experienced developer working in his free time for additional income.

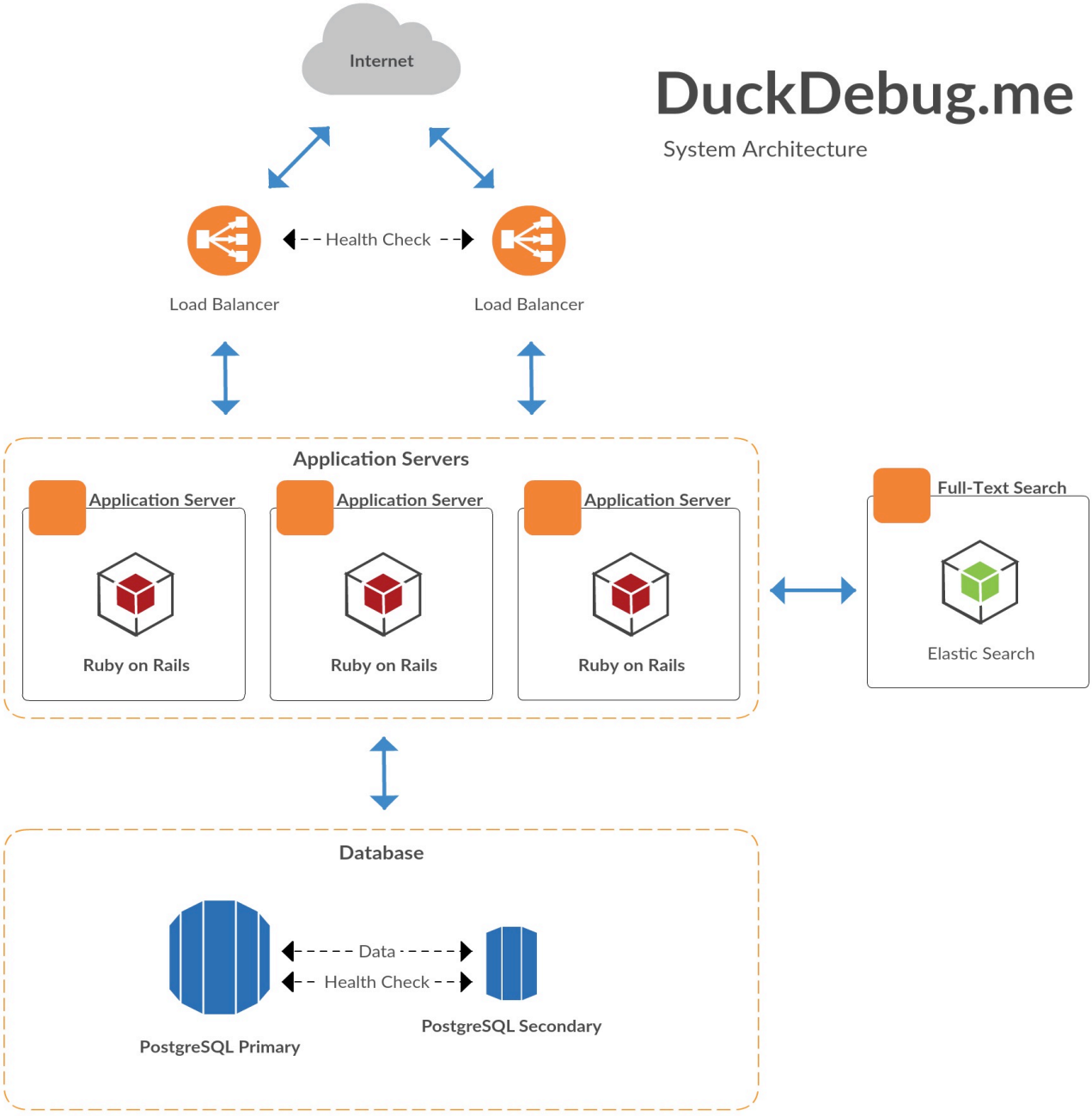
He needs to be able to sign onto the website and be able to view available bounties. Must be able to accept a bounty and download or clone it's code and upload changes when ready. He should receive payment for claiming the bounty within 14 days.

Allison – 27 year old Project Manager, creating a new application but struggling with a bug her team is struggling with.

She must be able to upload large chunks of code. Code must only be visible to programmers accepting the bounty to reduce number of people with access to this companies source code. Must be able to submit payment as a corporation.

The System

The Architectural Diagram



The Architecture Justification

Why Ruby on Rails?

Well-known advantages of Ruby:

- Available tools make it possible to create more features in less time
- Vast number of existing Libraries
  - Thousands of Gems publically available
- Learnability
  - Many developers agree that Ruby on Rails is easier to learn than some of its competitors
- Community
  - Ruby on Rails has a very large community
- It is still an up to date/Next Generation Programming language
  - Many well known code learning platforms like CodeAcedemy or Steer teach Ruby, this suggests there will be a lot more ruby developers in the next generation to come

Why Elastic Search?

While the output of an elastic search is severely limited, for both official as well as community client libraries a far greater number is supported in Elastic Search compared to its main competitor (Solr) which for our project is far more important than for example 3rd Party Product integration.

Why Use Load Balancers?

Load Balancers has several advantages:

- Scaling Usage:
  - With load Balancers in place even sudden spikes in traffic can be handled by spreading the load across several servers
  - Allow for a simple way for the administrators to add more servers to handle the load as demand increases
- Data Redundancy
  - As multiple servers are being used through this process, even if a server where to fail due to, for example, hardware failure there is assurance any processes can be continued on a separate server
- Server Uptime
  - All servers eventually need some sort of maintenance performed on them, with this system in place it would be easy for system administrators to take a server offline for maintenance while maintaining the system as a whole

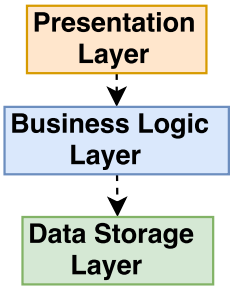
Why PostgreSQL?

Multiple known advantages:

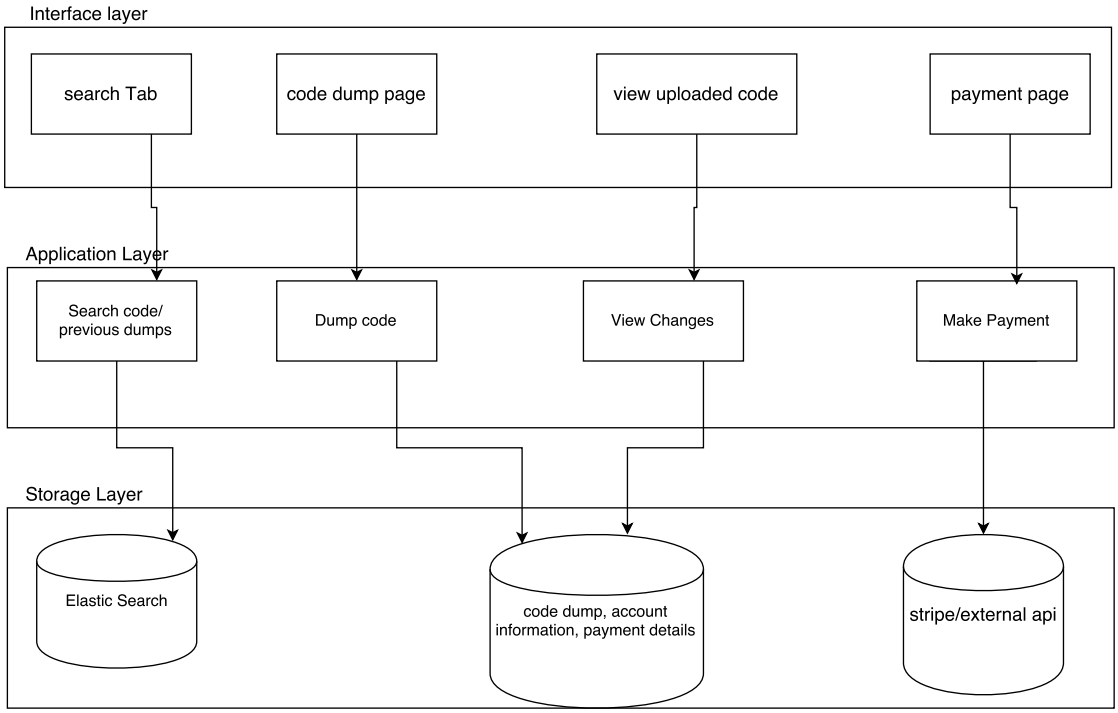
- Larger communities meaning better support options
- Well known reliability and stability
- Source code available at no charge
  - Changes can be made with minimal effort and no additional cost
- Immunity to so called “Over-deployment”
  - Generally opens up possibility of more profitable business models
- Incredible cross platform support
  - Almost every brand of Unix (more than 34 platforms)
  - Native Windows compatibility
- Many GUI tools already available

Three-Tier Architecture

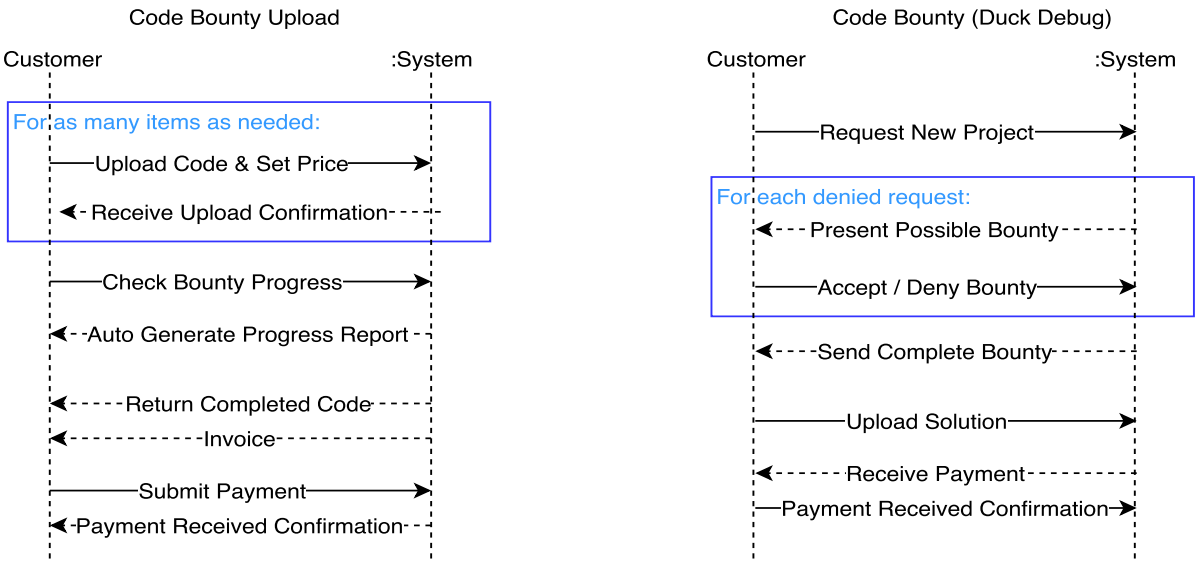
For our project we are using the principle three tier architecture (Which can be seen in the diagram to the right).



Below is an example of our three-tier architecture for a code bounty-submitting user.

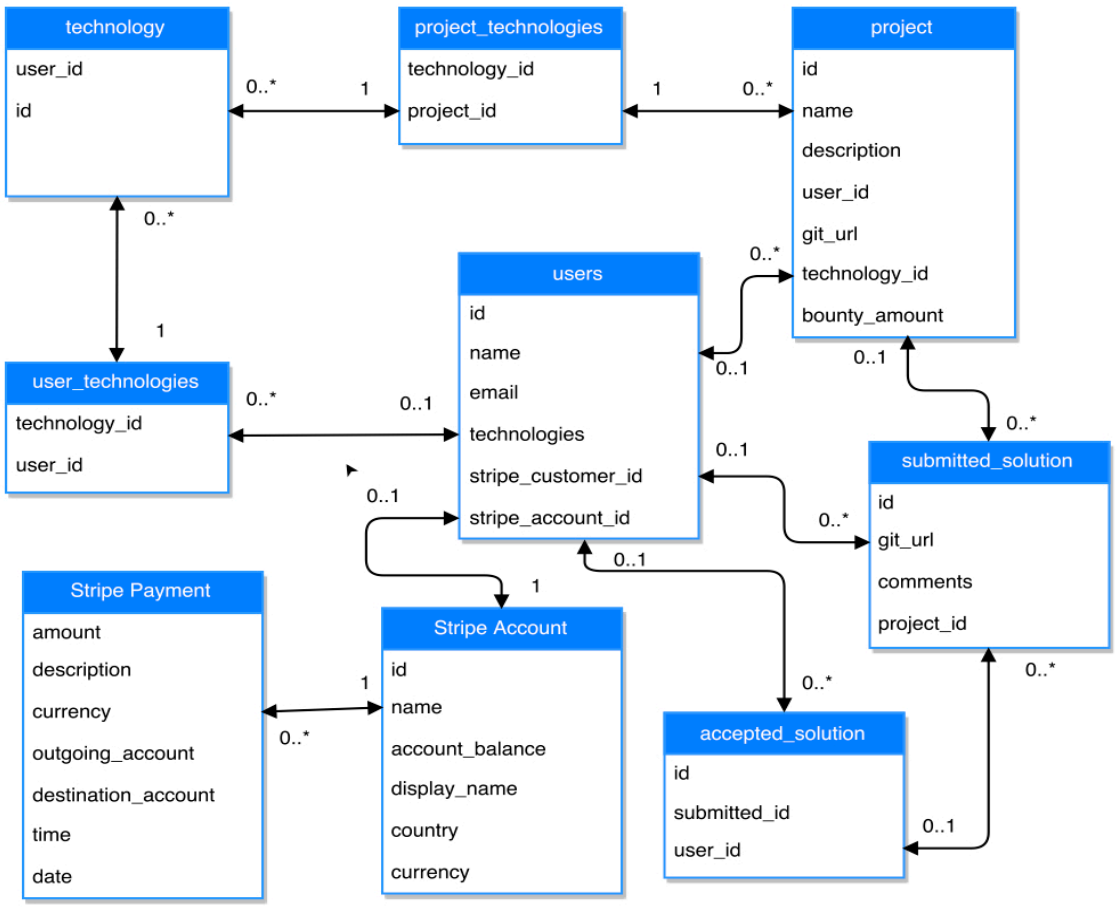


The System Sequence Diagram



These are the interactions we expect to be happening between the system and the code bounty uploaded (left side diagram) as well as the programmer completing and the bounty (right side diagram).

Duck Debug Entity Relationship Diagram



The Risk Assessment

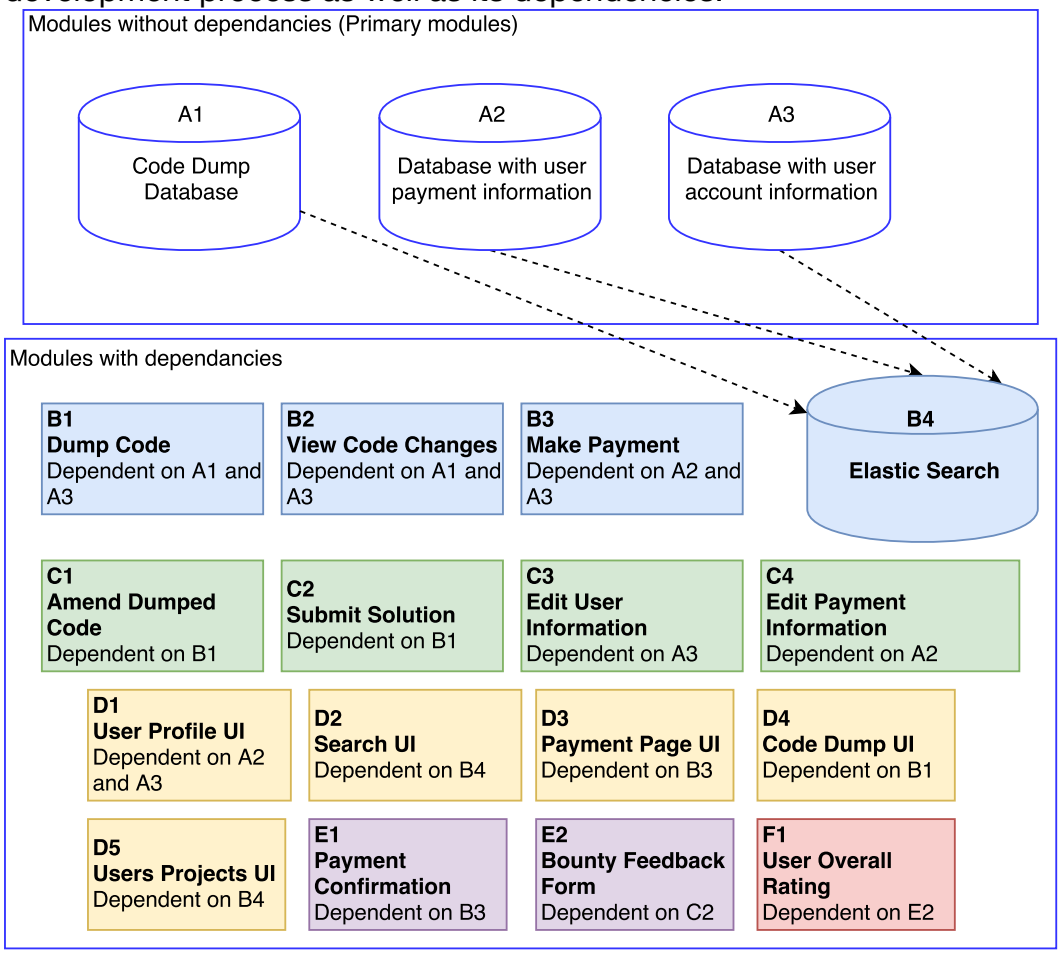
Below is a list of Risks that may cause problems throughout the development of Duck Debug with an explanation on how we plan to deal with them.

- High Risks
    - Security of Data
        - As the code uploaded to our servers is private property and the information about each user is sensitive, all data must be encrypted and as secure as possible. Failure in this department would not only open our businesses up to lawsuits it would ruin our reputation most likely leading to a loss of customers.
    - Loss of Data
        - As the bounties are uploaded to the servers in order for them to be solved, they need to be safe from corruption or loss of data. As aforementioned our choice of using load balancers will also greatly benefit us in terms of data being backed up to several different servers.
    - Privacy Issues
        - As there is a lot of private data involved with our project, including payment information we must be certain the data is only accessible by parties with the legal right to view them, to ensure this our contract will have to be well constructed to ensure we can provide the best experience possible to our users. Failure to protect our users privacy could also end in extensive lawsuits that might ruin our business.
- Low-Medium Risks
    - Large increase in users / user activity
        - Should our servers not be able to support a rapid increase of users or user activity, we could not only lose clients but also risk our uptime being affected drastically. At the moment we plan to address this issue through the use of load balancers, enabling us to quickly add more servers to our network and spreading the load among them as efficiently as possible.
    - Over-Deployment
        - Over-Deployment is a serious issue. It is regarded by many as one of the top licensing compliance problems and can cost a business such as ours a significant amount of money in lawsuits. The best way to negate the risk of over-deployment is to use software with no associated licensing cost. This is one of the reasons we have chosen to use PostgreSQL.

The Project Plan

The Plan

In order to proceed with the development of this project we must establish which sections are dependent on each other. Once that has been completed it should then be possible for all five developers to work on different sections simultaneously. Below is a Diagram, which through the use of colours and A1-3, B1-4, C1-4, D1-5, E1-2 and F1 roughly depicts the planned development process as well as its dependencies.



Milestones

The following list of milestones depicts what we believe the most crucial steps in our development cycle to be completed.

- **Database Implementation**
  - Once our PostgreSQL database is implemented all other functions will be able to draw from one centralized location rather than different files holding data. This will also benefit security.
- **Implementation of Code Dump**
  - While our Proof of concept software already has the ability to dump code, edit it and view it, this is accessible to all users of Duck Debug and not yet separated. Once the code dump has

been successfully implemented to the extent that only those whom accept the bounty can see it this will be the majority of our business layer completed.

- **Implementation of Elastic Search**
  - To ensure each user can get a good selection of bounties to choose from (those relevant to his skills in the user profile) elastic search must be completely implemented.
- **Implementation of User Ranking System**
  - It is important for programmers to receive feedback in order to improve their work. This feedback system for the customer will allow the programmers to receive a skill rating / ranking, which will become more relevant and accurate as they complete more projects.
- **Implementation of User Interface**
  - Once all Business Layer and Storage Layer functions are complete we must create a user interface that is user friendly and incorporates all of our functions as well as possible.

The Initial Schedule

Weeks 1,2 and 3: Brainstorming:
The first three weeks where spent brainstorming possible projects and completing market research as well as basic cost projections to check the viability of the projects we where coming up with. In order to improve communication and teamwork we implemented a custom management system allowing us to keep track of what developer did which work in the future.

Weeks 4 and 5:
Now that we had selected our project “DuckDebug”, and completed the income and expenses models it was time to design our database in order to assure all the features we wanted could be included in a well-constructed system.

Week 6:
During week 6 we began the development of our system, we created a basic framework, which would allow us to ease development later down the line.

Week 7:
We completed the Registration, Login and Basic Code Dumping functions.

Week 8 and 9:
In the final two weeks we tested and fixed some bugs in our proof of concept software and completed the documentation.

The Proof of Concept

What has been completed so far?

Our proof of concept software has been developed rather efficiently thus far and we are happy to report that the following core requirements have been completed in terms of backend programming:

- User is able to Register
- User is able to Login
- User is able to submit code with a title and explanation of the issue
- User is able to edit his code submission
- User is able to submit more than one section of code (multiple bounties)
- User is able to submit replies (solutions) to uploaded bits of code.

While these backend functions are completed, our UI (as it was not a priority thus far) has been kept rather simple, with only the required fields present with little or no design thought put into it. However now that a lot of these basic functions are complete it will be relatively easy to create a user-friendly interface going forward.

Problems we faced

Other than some small bugs that came up every now and again through the development cycle, we have had little to no problems with the software so far. The only issue that we foresee will be the successful implementation of a filtering system allowing only certain users even the opportunity to view a bounty, however we strongly believe that if we move forward as we have planned in our designs even this challenge should be manageable without too much difficulty.

What's next?

Now that our software has a decent framework with our core functionality already implemented it should be simply for us to move forward as a unit in future development. Once we flesh out the "Code Dump" system as well as the "Elastic Search" and are able to filter projects depending on users abilities (listed in their profile), we will be able to start fleshing out our user interface design. As aforementioned (in the milestones section) we plan to also implement a form of feedback and ranking system, which we hope to be able to complete in parallel with the development of our user friendly user interface.

Conclusion

While we struggled at first to come together as a unit, thanks to the implementation of our own custom project management system we where able to work together quite well and have been able to pass some valuable skills onto each other. While we were not always able to complete our tasks fully on time, every member of our group has learnt from it and improved throughout the first semester of this project. Once the complete development starts we are confident that with the help of our design and the management system we will be able to complete tasks efficiently and quickly discover any problems that we can address before they affect our final product.