\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Team Dijkstra  
Crist – Howell – Ghodratnama

t14 – Defect Database  
Email Server/Client System

Software Engineering II  
Spring 2013  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

*“Elegance is not a dispensable luxury but a quality that decides between success and failure”   
-* Edsger Dijkstra

Table of Contents

[Defect Overview 3](#_Toc353817653)

[Defect Prevention 3](#_Toc353817654)

[System Wide Defect Database 4](#_Toc353817655)

[Local Defect Database 8](#_Toc353817656)

# Defect Overview

The following is the defect database for our project. In our project we designated two types of defects. One is a local defect which affects only one developer. The other is a system wide defect. A system defect is one that affects multiple developers and needs to be solved before development continued. In our project we were fortunate enough have very few of these defects.

Having two types of defects, we needed to have a system of tracking defects when they arise. For a system wide defect. We used the Google Code Issue tracker included with our SVN repository. This allows a central location to where each developer can see each defect. The second type of defect is a local defect. These are normal defects that arise when a developer is coding and testing their piece of software. Since these defects are normally caught before they are committed to the SVN system, it is not necessary to alert the group to these issues. These issues should instead be logged in the developers personal PSP log.

# Defect Prevention

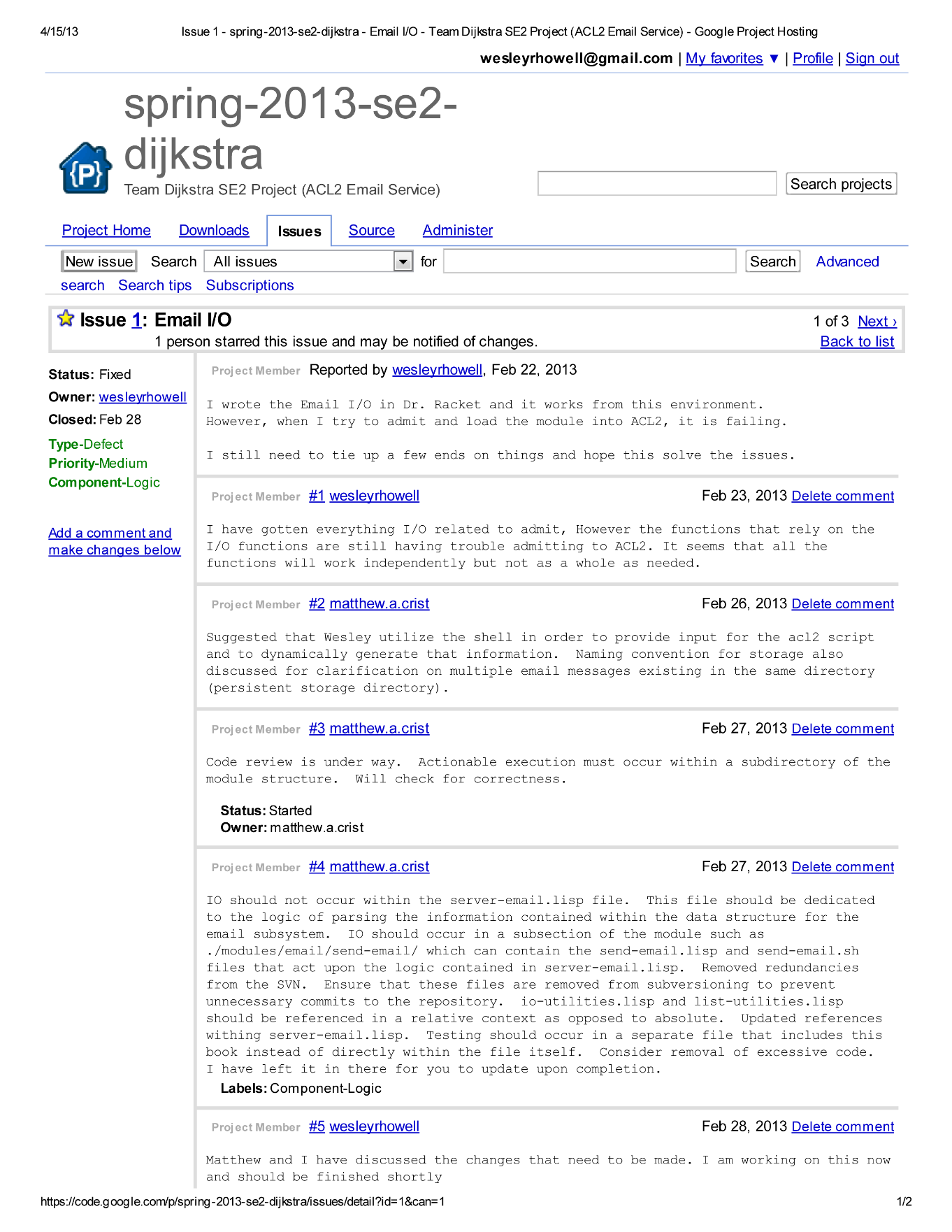
In order to have a successful implementation, we needed to take several step to eliminate defects. The first step of defect prevention was the use of the modular structure outlined in this document. This allowed each module to be developed independently from each other. This allowed only one program to be written to execute each action as needed. This also allowed us to keep IO and logic files separate. By doing this, generating theorems and test for the logic was less complicated since we enforced the requirements of having no dependencies in the ACL2 logic portions of the modules.

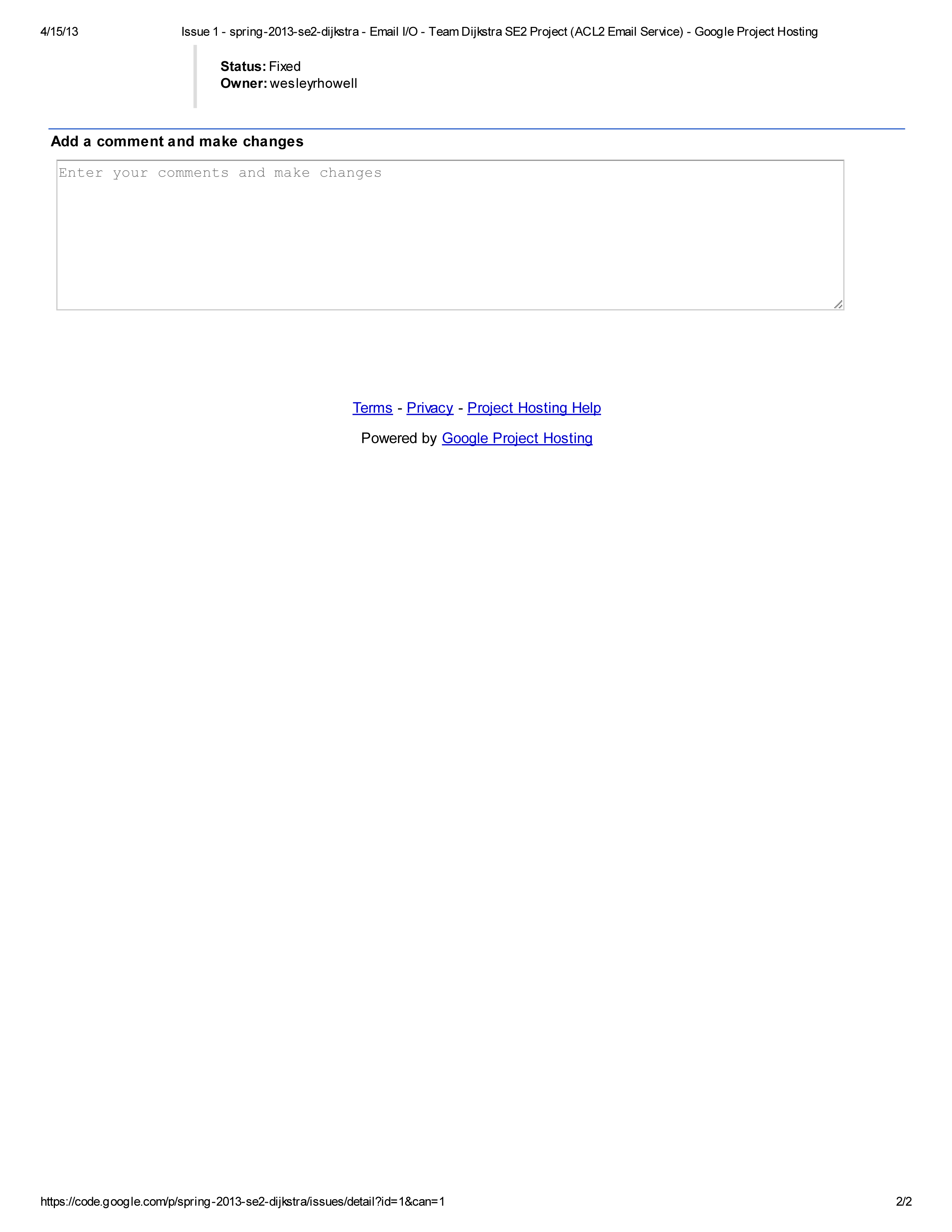
The next step in defect prevention was the use of ACL2 theorems to verify that the output of our logic functions, in fact, were correct. The theorem structure outlined in the *Testing* section allowed us to provide a logically sound proof that the functions we defined generated the guaranteed output. This guaranteed output along with strict XML DTD’s provided assurance that no stray data is going to be sent to or from the server.

The final step we took was careful planning and design of the modules. Since we designed this program modularly, we knew that once a module is completed, it would continue to work regardless of the progress made on other modules. This careful design and constant planning for of the program allowed all developers to be on the same page as the development process began. Constant updates to the design were inevitable but rarely deviated from the initial modular concept of the application.

# System Wide Defect Database

Below is the log of issues recorded into the system wide defect database.









# Local Defect Database

The following is the individual defect database from each developers logs.

