SYSTEMS AND SOFTWARE REQUIREMENTS SPECIFICATION (SSRS) FOR

sQuire Collaborative IDE

Picture here.

Version 1.0
February 09, 2016

Prepared for:
CS383-01

Prepared by:
Domn Werner (wern0096) | Robert Carlson (carl7595) | Brian Cartwright (cart1189)
Max Welch (welc2150) | Matthew Daniel (dani2918) | Brandon Ratcliff (ratc8795)
Joel Doumit (doum6708) | Eric Gentile-Quant (gent7104)
Team 4 - It Compiled Yesterday (ICY)
University of Idaho
Moscow, ID 83844-1010

sQuire SSRS

RECORD OF CHANGES

Change number	Date completed	Location of change (e.g., page or figure #)	**A M D**	Brief description of change	Approved by (initials)	Date Approved
1	02/11/16	Page 1	M	Updated Team Name	DW	02/11/16
2	02/11/16	Pages 8-10	M	Updated Functional Reqs	DW	02/11/16
3	02/18/16	Section 3.3	A	Added Use Cases	DW	02/18/16
4	02/18/16	Section 3.4	A	Added Class Diagram Section	DW	02/18/16

*__A__ - ADDED __M__ - MODIFIED __D__ - DELETED

SQUIRE SSRS
TABLE OF CONTENTS

Section Page

1 Introduction

The sQuire Collaborative IDE is a collaborative IDE software project for CS383-01. The intended audience for this project is Java programmers looking for a more social collaborative experience. A large focus of the program is also to help programmers connect with others who may interested in their projects.

1.1 IDENTIFICATION

The software system being considered for development is referred to as sQuire. The customer providing specifications for the system is Dr. Jeffery and the CS383-01 class. The ultimate customer, or end-user, of the system will be Java programmers. This is a new project effort, so the version under development is version 1.0.

1.2 PURPOSE

The purpose of the system under development is to provide Java programmers with a more social collaborative experience. Instead of individual methods of source control, sQuire will provide an environment where programmers can work together in the same environment and instantly see the effect of others' code. While the system will be used by Java programmers, this document is intended to be read and understood by UI CS software designers and coders. The document will also be vetted or approved by Team 4.

1.3 SCOPE

This paragraph shall briefly summarize the history of system development, operation, and maintenance; identify the project sponsor, acquirer, user, developer, and support agencies; identify current and planned operating sites; and list other relevant documents.

[insert your text here]

1.4 DEFINITIONS, ACRONYMS, AND ABBREVIATIONS

This section shall list and define all special terms, acronyms and abbreviations used throughout this document. A tabular form is preferable, but not mandatory.

Term or Acronym	Definition
Alpha test	Limited release(s) to selected, outside testers
Beta test	Limited release(s) to cooperating customers wanting early access to developing systems
Final test	aka, Acceptance test, release of full functionality to customer for approval
DFD	Data Flow Diagram
SDD	Software Design Document, aka SDS, Software Design Specification
SRS	Software Requirements Specification
SSRS	System and Software Requirements Specification
IDE	Integrated Development Environment

1.5 REFERENCES

This section shall list full bibliographic citations of all documents referenced in this report. This section shall also identify the source for all materials not available in printed form (e.g., web-based information) and list the complete URL along with owner, author, posting date, and date last visited.

[insert your citations here]

1.6 OVERVIEW AND RESTRICTIONS

This paragraph shall describe the organization of this document and shall describe any security or privacy considerations associated with its use.

This document is for limited release only to UI CS personnel working on the project and [state others who will receive the document].

Section 2 of this document describes the system under development from a holistic point of view. Functions, characteristics, constraints, assumptions, dependencies, and overall requirements are defined from the system-level perspective.

Section 3 of this document describes the specific requirements of the system being developed. Interfaces, features, and specific requirements are enumerated and described to a degree sufficient for a knowledgeable designer or coder to begin crafting an architectural solution to the proposed system.

Section 4 provides the requirements traceability information for the project. Each feature of the system is indexed by the SSRS requirement number and linked to its SDD and test references.

Sections 5 and up are appendices including original information and communications used to create this document.

2 OVERALL DESCRIPTION

The sQuire project is an answer to the lack of real-time collaborative programming experiences. By bringing programmers together in a more social environment, this program aims to improve collaboration between programmers in a much more fast paced and agile methodology. Furthermore, for programmers who seek others to help with their projects, sQuire aims to provide a simple social platform for engaging with other programmers and start working on a project together.

2.1 PRODUCT PERSPECTIVE

This program will either be a standalone executable with many linked DLLs or a web application that can run on most modern web-browsers with internet access.

2.2 PRODUCT FUNCTIONS

This subsection of the document should provide a summary of the major functions that the software will perform. For the sake of clarity, the functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time. Textual or graphical methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product, but simply shows the logical relationships among variables.

[insert your text here]

2.3 USER CHARACTERISTICS

This subsection of the document should describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. It should not be used to state specific requirements, but rather should provide the reasons why certain specific requirements are later specified in Section 3 of this document.

[insert your text here]

2.4 CONSTRAINTS

This subsection of the document should provide a general description of any other items that will limit the developer's options. These include: a) Regulatory policies; b) Hardware limitations (e.g., signal timing requirements); c) Interfaces to other applications; d) Parallel operation; e) Audit functions; f) Control functions; g) Higher-order language requirements; h) Signal handshake protocols; i) Reliability requirements; j) Criticality of the application; k) Safety and security considerations.

[insert your text here]

2.5 ASSUMPTIONS AND DEPENDENCIES

This subsection of the document should list each of the factors that affect the requirements stated in the document. These factors are not design constraints on the system and/or software but are, rather, any changes to them that can affect the requirements in the document. For example, an assumption may be that a specific operating system will be available on the hardware designated for the software product. If, in fact, the operating system is not available, the document would then have to change accordingly.

[insert your text here]

2.6 SYSTEM LEVEL (NON-FUNCTIONAL) REQUIREMENTS

This subsection of the document should identify system level (whole, not functional) requirements that impact the construction, operation, packaging and delivery of the system and software.

2.6.1 Site dependencies

This paragraph shall specify site-dependent operational parameters and needs (such as parameters indicating operation-dependent targeting constants or data recording). The requirements shall include, as applicable, number of each type of equipment, type, size, capacity, and other required characteristics of processors, memory, input/output devices, auxiliary storage, communications/ network equipment, and other required equipment or software that must be used by, or incorporated into, the system. Examples include operating systems, database management systems, communications/network software, utility software, input and equipment simulators, test software, and manufacturing software. The correct nomenclature, version, and documentation references of each such device or software item shall be provided.

1. Central SQL Server
2. Host-side Project Server
3. Collaborator-side Client

The Central server stores user credentials, project descriptions, and user profile and achievement data.
Requirements for Central SQL Server

1. Host with high uptime percentage
2. SQL capable
3. E-mail capable for password resets
4. Fast enough connection to prevent login timeout, even while handling multiple requests
5. Prefer host with multiple backups

The Host-side server stores the project files, project access list, hosts the editing environment, runs chat channels, and serves files to collaborators for compiling.
Requirements for Host-side Server

1. Java Capable (http://java.com/en/download/help/sysreq.xml)
2. SQL Capable (WAMP/LAMP)
3. 4 GB RAM
4. Hard drive space for server + project files

The client side application connects to the host server, renders GUI elements, stores connection profiles, stores server files, and compiles the project.
Requirements for Collaborator-side Client

1. Java Capable (http://java.com/en/download/help/sysreq.xml)
2. Project specified Java installed
3. 2 GB RAM
4. Hard drive space for project files

2.6.2 Safety, security and privacy requirements

This paragraph shall specify the system requirements, if any, concerned with maintaining safety, security and privacy. These requirements shall include, as applicable, the safety, security and privacy environment in which the system must operate, the type and degree of security or privacy to be provided, and the criteria that must be met for safety/security/privacy certification and/or accreditation.

The collaborative nature of sQuire includes several concerns for security and privacy. The program will include in the license agreement the following stipulations:

1. sQuire is a free development environment, and may be used for commercial purposes
2. No guarantee of code confidentiality is implied by use of sQuire
3. Clients assume the risk of downloading, compiling, and running project files
4. Email addresses are visible as part of a user profile
5. Host assume the risk of allowing peers to connect to their server

However, the program will provide the following minimum features to address security and privacy concerns:

1. All SQL servers will include input sanitization and appropriate anti-injection safeguards
2. Project hosts may turn off guest access to their project
3. Uploads for assets will be limited to folders within the project directory
4. Visibility to host file structure will be limited to project folders only

2.6.3 Performance requirements

This paragraph should specify both the static and the dynamic numerical performance requirements placed on the software or on human interaction as a whole. Static numerical requirements may include the following: a) The number of terminals to be supported; b) The number of simultaneous users to be supported; c) Amount and type of information to be handled. Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions. All of these requirements should be stated in measurable terms. For example, "95% of the transactions shall be processed in less than 1msec."

1. Up to 33 concurrent connections will be supported
2. Edits will be visible to all connected collaborators within 10 seconds
3. Login and server connections will report success or failure within 45 seconds

2.6.4 System and software quality

This paragraph shall specify the requirements, if any, concerned with hardware and software quality factors identified in the contract. Examples include quantitative requirements regarding the system's functionality (the ability to perform all required functions), reliability (the ability to perform with correct, consistent results), maintainability (the ability to be easily corrected), availability (the ability to be accessed and operated when needed), flexibility (the ability to be easily adapted to changing requirements), portability (the ability to be easily modified for a new environment), reusability (the ability to be used in multiple applications), testability (the ability to be easily and thoroughly tested), usability (the ability to be easily learned and used), and other attributes.

Adaptability

1. The program will allow selection of different compiling programs and command line arguments.

2. The program will allow importing of files of key words to allow other development languages to be used.

2.6.5 Packaging and delivery requirements

This paragraph shall specify the requirements, if any, for packaging, labeling, handling and delivery of the system being developed to the customer.

The executable system and all associated documentation (i.e., SSRS, SDD, code listing, test plan (data and results), and user manual) will be delivered to the customer on CD's and/or via email, as specified by the customer at time of delivery. Although document "drops" will occur throughout the system development process, the final, edited version of the above documents will accompany the final, accepted version of the executable system.

2.6.6 Personnel-related requirements

This paragraph shall specify the system requirements, if any, included to accommodate the number, skill levels, duty cycles, training needs, or other information about the personnel who will use or support the system under development. These requirements shall include, as applicable, considerations for the capabilities and limitations of humans; foreseeable human errors under both normal and extreme conditions; and specific areas where the effects of human error would be particularly serious. Examples include requirements for color and duration of error messages, physical placement of critical indicators or keys, and use of auditory signals.

The system under development has no special personnel-related characteristics.

2.6.7 Training-related requirements

This paragraph shall specify the system requirements, if any, pertaining to training. Examples include training software, tutorials, or help information to be included in the system.

No training materials or expectations are tied to this project other than the limited help screens built into the software and the accompanying user manual.

2.6.8 Logistics-related requirements

This paragraph shall specify the system requirements, if any, concerned with logistics considerations. These considerations may include: system maintenance, software support, system transportation modes, supply-system requirements, impact on existing facilities, and impact on existing equipment.

[Insert a description of the minimum hardware requirements and OS and application software dependencies here]

2.6.9 Other requirements

This paragraph shall specify additional system level requirements, if any, not covered in the previous paragraphs.

[insert your text here]

2.6.10 Precedence and criticality of requirements

This paragraph shall specify, if applicable, the order of precedence, criticality, or assigned weights indicating the relative importance of the requirements in this specification. Examples include identifying those

requirements deemed critical to safety, to security, or to privacy for purposes of singling them out for special treatment. If all requirements have equal weight, this paragraph shall so state.

[insert your text here]

3 SPECIFIC REQUIREMENTS

3.1 Functional Requirements

3.1.1 Project Browsing

These requirements involve the ability for users to find and learn about projects that they may wish to contribute to. Users shall be able to:

1. See a list of open projects.
2. Filter or search projects.
3. View more information about a specific project.
4. Upvote and downvote projects.
5. Comment on projects and interact with its contributors.
6. Request to join a specific project.

3.1.2 Authentication

These requirements involve the ability for users to have individual accounts and the security that comes from that. Users shall be able to:

1. Register for a sQuire user account.
2. Log in to their user account.
3. Log out of their user account.
4. Reset their password via email.

3.1.3 Communication

These requirements involve the ability for users to be able to communicate with other users. Users shall be able to:

1. Open and close project chat.
2. Write to project chat.
3. Read from project chat.
4. Message a user by their name.
5. Leave a comment in a file.

3.1.4 File Management

These requirements involve the ability for users to manage the files that compose a project. Users shall be able able to:

1. Open one or more files.
2. Close one or more files.
3. Delete one or more files.
4. Download one or more files.
5. Add a new file to the project.
6. Add an existing file to the project.
7. Save one or more files.

3.1.5 File Editing

These requirements involve the collaborative editor part of sQuire. Users shall be able to:

1. Enable or disable line numbers.
2. Enable or disable viewing reference counts above each line.
3. Enable or disable viewing date of last edit above each line.
4. Enable or disable view author of each line.
5. Comment/Uncomment a selected section code.
6. Format the document to adhere to code style.
7. Find/Replace specified text.
8. View text highlighted by other users.
9. Type text and have the system apply syntax coloring for Java files and display errors.
10. View other users' carets as they type.

3.1.6 Project Management

These requirements involve the management of entire projects. Users shall be able to:

1. Compile a project.
2. Execute a compiled project.
3. Create a new project.
4. Delete a project.
5. Open a project.
6. Close a project.
7. Invite a user to a project.
8. Join a project.
9. Leave a project.

3.1.7 Project Member Management

These requirements involve project admins managing their members. Admins shall be able to:

1. Invite users to a project.
2. Change user permissions to a project.
3. Remove users from a project.
4. Block a user from a project.

3.1.8 User Preferences

These requirements involve managing user preferences. Users shall be able to:

1. Update their username.
2. Update their password.
3. Update their email address.
4. Update their biography.
5. Update their display name.
6. Enable receiving email updates.
7. Enable receiving messages from any user.
8. Display their email address to selected groups.
9. Change program colors.

3.2 EXTERNAL INTERFACE REQUIREMENTS

This subsection should be a detailed description of all inputs into and outputs from the software system. It should complement the constraints and dependencies defined in earlier sections, but not repeat that information. Hardware, software, user, and other communication interfaces need to be specified. Use the four subsections listed below or the table on the next page, or some combination of both.

3.2.1 Hardware Interfaces

[insert your text here]

3.2.2 Software Interfaces

[insert your text here]

3.2.3 User Interfaces

[insert your text here]

3.2.4 Other Communication Interfaces

[insert your text here]

External Interface Requirements

Hardware Interfaces

Name	Source/Destination	Description	Type/range	Dependencies

Software Interfaces

Name	Source/Destination	Description	Type/range	Dependencies

User Interfaces

Name	Source/Destination	Description	Type/range	Dependencies

Other Communication Interfaces

Name	Source/Destination	Description	Type/range	Dependencies

3.3 SYSTEM FEATURES

Functional requirements should define the fundamental actions (i.e., features) that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These requirements are given in the form of ***Use Cases*** *where possible, denoting a concrete use (discrete user-performable task) of the system. Use case diagrams are followed by use case descriptions, followed by any non-task features. Non-task features are generally listed as "shall" statements starting with "The system shall..." These include: a) Validity checks on the inputs; b) Exact sequence of operations; c) Responses to abnormal situations, including error detection, handling and recovery; d) Parameter specification and usage; e) Relationship of outputs to inputs, including formulas for input to output conversion.*

It may be appropriate to partition the functional requirements into sub functions or subprocesses, but that decomposition (here) does not imply that the software design will also be partitioned that way. You should repeat subsections 3.3.i for every specified feature defined for the system or software.

3.3.1 Use Case Diagram 1: Authentication

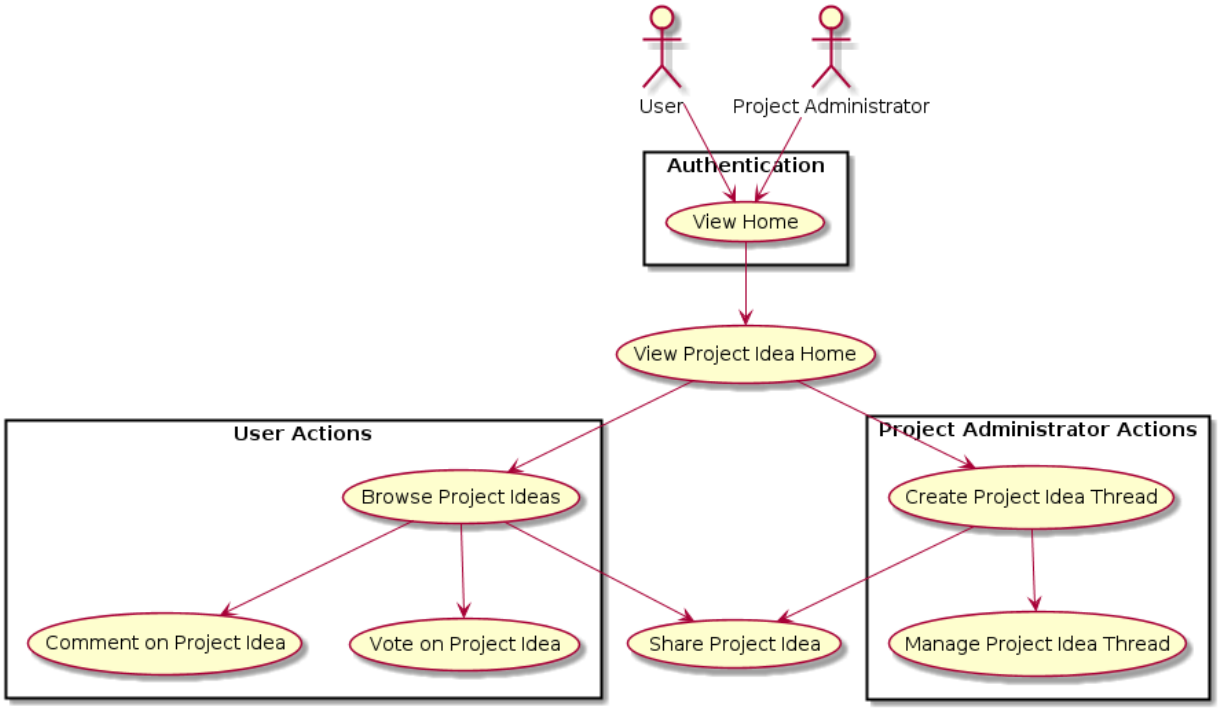
3.3.2 Authentication Feature 1: Use Case 1

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.3 Authentication Feature 2: [Use Case Description 2]

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.4 Use Case Diagram 2: Project Browsing



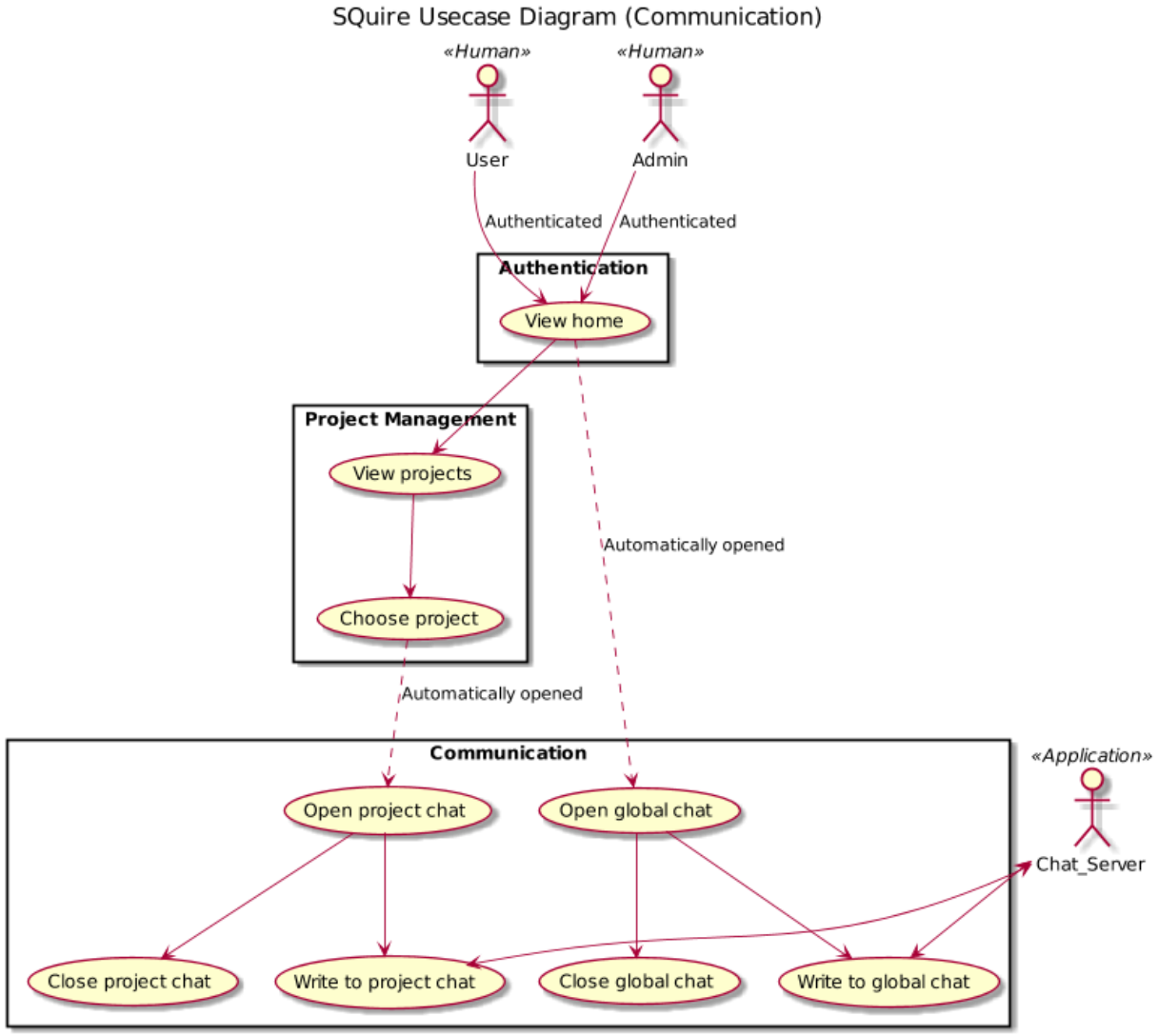
3.3.5 Project Browsing Feature 1: Use Case Description 1

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.6 Project Browsing Feature 2: Use Case Description 2

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.7 Use Case Diagram 3: Communication



3.3.8 Communication Feature 1: Use Case Description 1

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

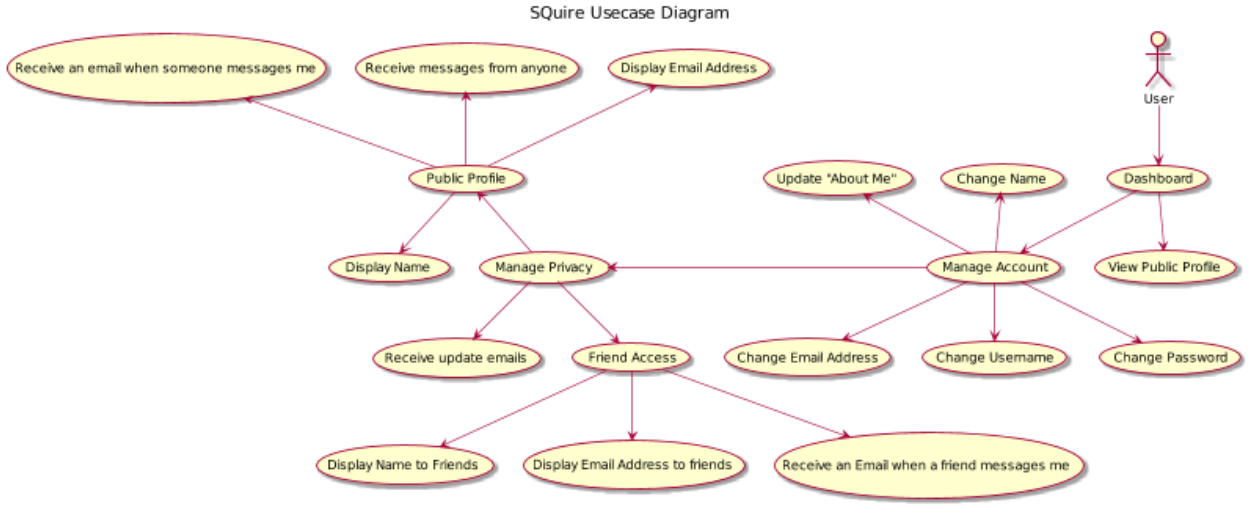
3.3.9 Communication Feature 2: Use Case Description 2

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.10 Project Browsing Feature 2: Use Case Description 2

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.11 Use Case Diagram 4: User Preferences



3.3.12 User Preferences Feature 1: Use Case Description 1

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.13 User Preferences Feature 2: Use Case Description 2

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.14 Use Case Diagram 5: Project Management

Need picture here.

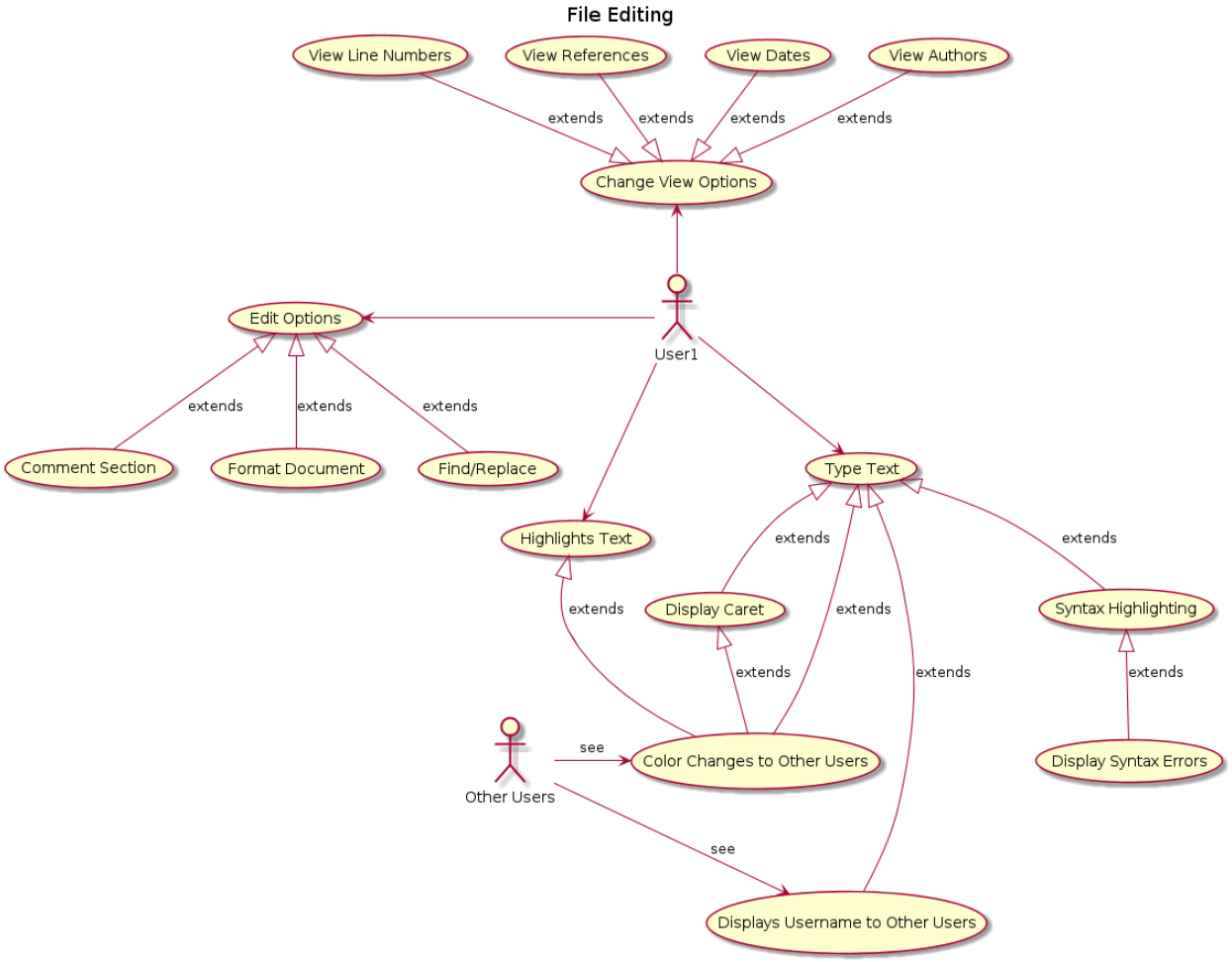
3.3.15 Project Management Feature 1: Use Case Description 1

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.16 Project Management Feature 2: Use Case Description 2

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.17 Use Case Diagram 6: File Editing



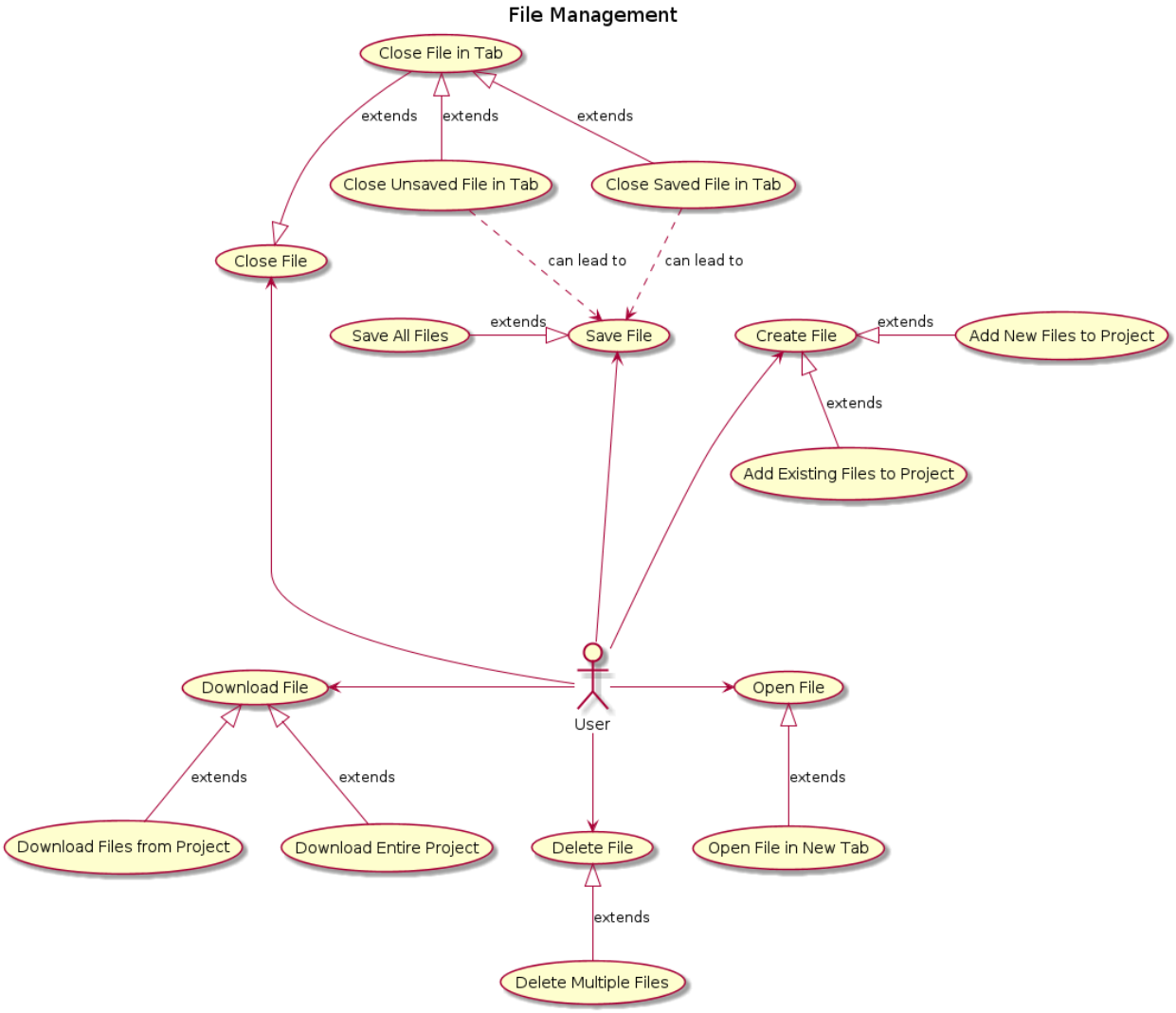
3.3.18 File Editing Feature 1: Use Case Description 1

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.19 File Editing Feature 2: Use Case Description 2

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.20 Use Case Diagram 7: File Management



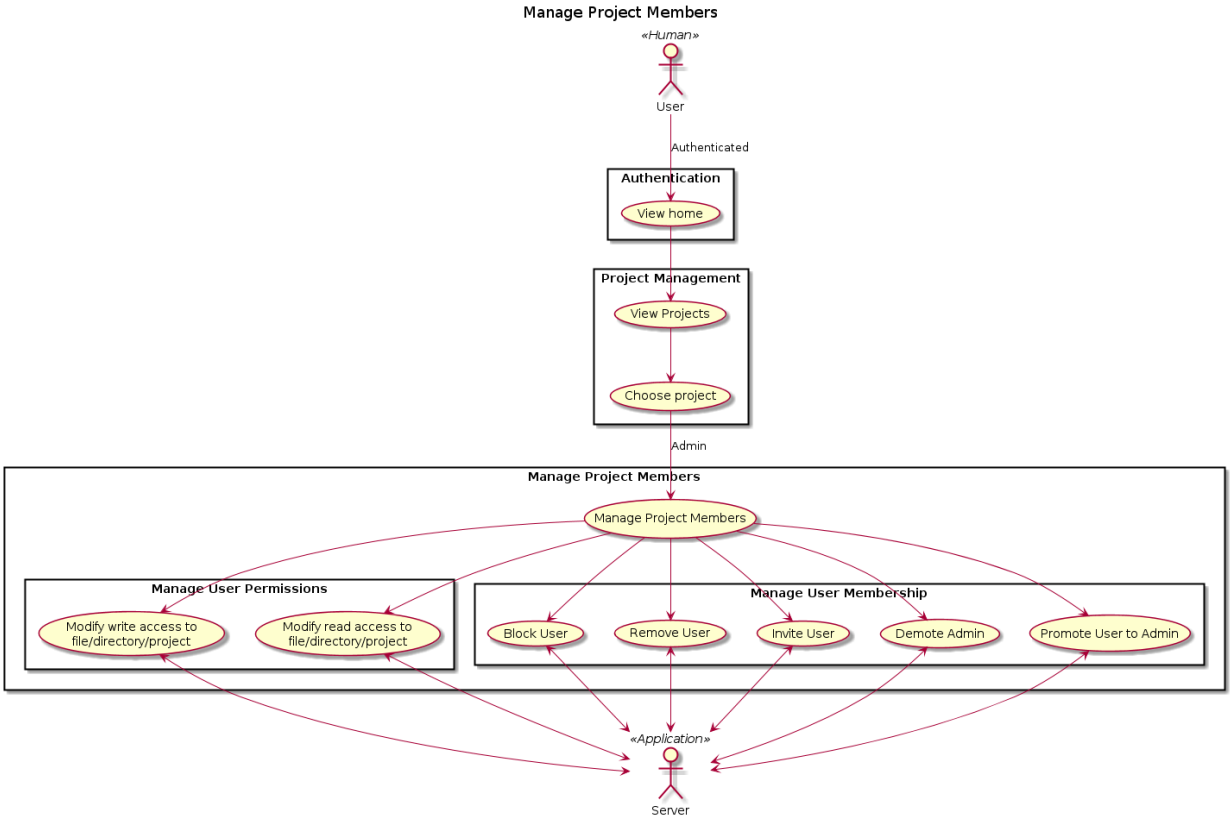
3.3.21 File Management Feature 1: Use Case Description 1

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.22 File Management Feature 2: Use Case Description 2

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.23 Use Case Diagram 8: Project User Management



3.3.24 Project User Management Feature 1: Use Case Description 1

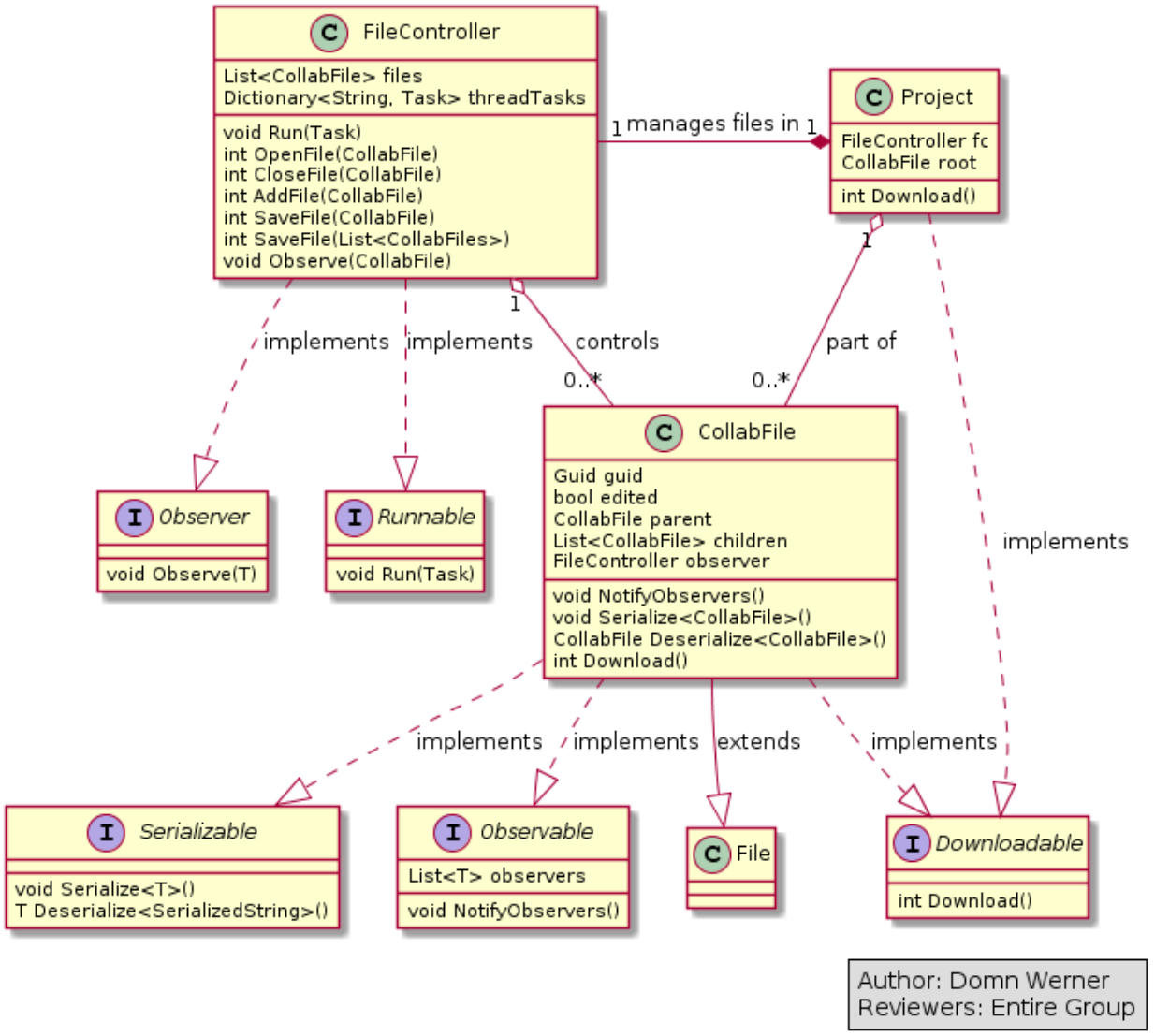
For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.3.25 Project User Management Feature 2: Use Case Description 2

For each feature, you should either provide a Use Case Description ***or*** *a Non-task feature description, whichever is more appropriate.*

3.4 CLASS DIAGRAMS

3.4.1 Class Diagram 1: File Management



3.4.2 Class Diagram Description 1: File Management Description

Interfaces:

- The *Observer* interface requires its implementers to implement a method with the following signature: **void Observe(T)**. The purpose of this method is for classes to implement ways in which to observe other classes. We foresee the **FileController** class implementing this interface in order to observe changes in **CollabFile** objects.

- The *Runnable* interface requires its implementers to implement a method with the following signature: **void Run(Task)** where **Task** is a method that can be run in a separate thread. The **FileController** class will implement this interface in order to execute its IO operations in a separate thread. This will keep the UI thread free and our program responsive.

- The *Serializable* interface requires its implementers to implement the **void Serialize<T>()** method and serializes objects of type **T**. It also requires the implementation of the **T Deserialize<T>()** method which will operate on a serialized string object and return an instantiated object of type **T**.

- The *Observable* interface requires its implementers to have a list of observers and a method to notify its observers of changes to itself. The purpose of this implementation is to communicate with the **FileController** object and notify it when a CollabFile changes.

- The *Downloadable* interface requires its implementers to implement a method with the following signature: **int Download()**. The purpose of this method is for classes to implement ways in which their objects can be downloaded. The **int** return type will be used as a status code. We foresee this interface being used with the **Project** and **CollabFile** classes, as the diagram shows, allowing users to download files or projects with the **Download()** method.

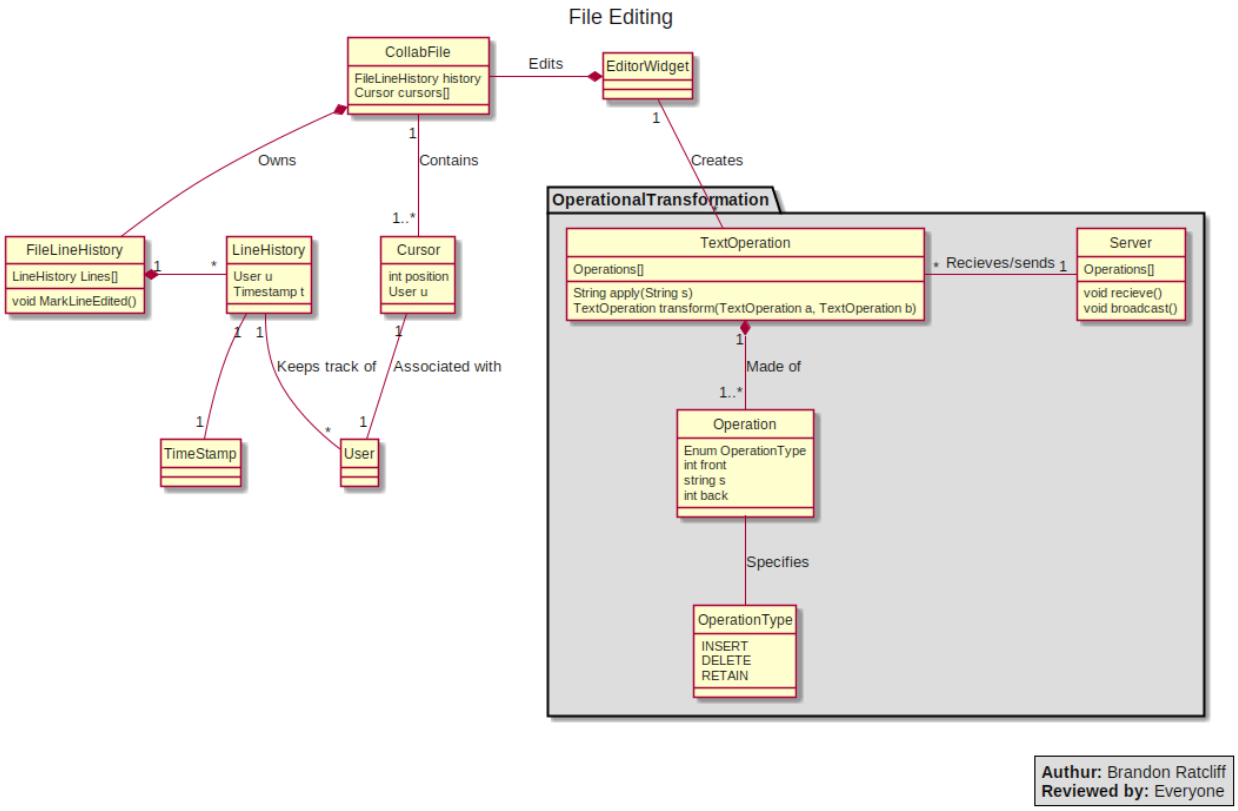
Classes:

- The **FileController** class will manage the **CollabFile** objects in the **Project** class. It will do so by storing a list of files and a dictionary of its methods that can be run in a separate thread. Its methods all deal with managing files. It will implement the *Observer* and *Runnable* interfaces. Refer to the interfaces list above to see the details of such implementations.

- The **Project** class represents the entire project that users work on. This includes users, files, permissions, etc. For the sake of simplicity, this diagram only lists properties and methods relating to file editing. Objects of type **Project** will have a **FileController** and a root **CollabFile** as per a file-tree structure. The **Project** class must also implement the *Downloadable* interface in order to specify how projects are downloaded.Refer to the interfaces list above to see the details of this implementation.

- the **CollabFile** class represents a file in a **Project**. It extends the **File** class for the purposes of allowing collaborative editing, among other project functions. It implements the *Serializable* interface to allow its information to be transported over the internet in the best possible format. This requires the implementation of the **void Serialize<T>()** and **T Deserialize<SerializedString>()** methods which will handle serialization and deserialization. This class also implements the *Observable* interface which will specify how it communicates with the **FileController** class in order to notify of relevant changes to **CollabFile** objects. This requires the implementation of a list of observers and a method to notify observers. Lastly, it implements the *Downlodable* interface which will specify how **CollabFile** objects are to be downloaded. Refer to the interfaces list above for more details of such implementations.

3.4.3 Class Diagram 2: File Editing



3.4.4 Class Diagram Description 2: File Editing Description

Editor:

- The *CollabFile* class is a class used in many of the other class diagrams in this project. It is the general class containing all the methods and variables for managing a file. It contains a **FileLineHistory** object. CollabFile has a list of **Cursor** objects, one for every user editing the file..
- The *User* class is a class used in many of the class diagrams. It represents a single user of the sQuire program.
- The *TimeStamp* class is used in several other places. It represents a date and time.
- The *FileLineHistory* class is a class that keeps track of who last edited every line in the file. This will be used to display the changes inside the editor. It does this by having an array (one element per line in the file) of **LineHistory** objects.
- The *LineHistory* class is contains the information used by the **FileLineHistory** class. It contains a **User** the last one to edit a particular line and a **Timestamp** (when the line was last edited). More fields can easily be added to this if it turns out there is more information we'd like to keep track of on a line-by-line basis.
- The *EditorWidget* class is something that we will (hopefully) not write ourselves. It will be the editor widget we use for providing the code editor. Preliminary research found RSyntaxtTextArea (https://github.com/bobbylight/RSyntaxTextArea). This looks like a good fit because it has syntax highlighting, auto completion, code analysis, and of course, support from java. It also has a simple plugin architecture, so it looks like it would be easy to extend to our needs. More research needs to be done to figure out the exact class structure for this.
- The *Cursor* class is made up of a **User** and a position within a file- everything that is needed to display a users cursor inside the editor.

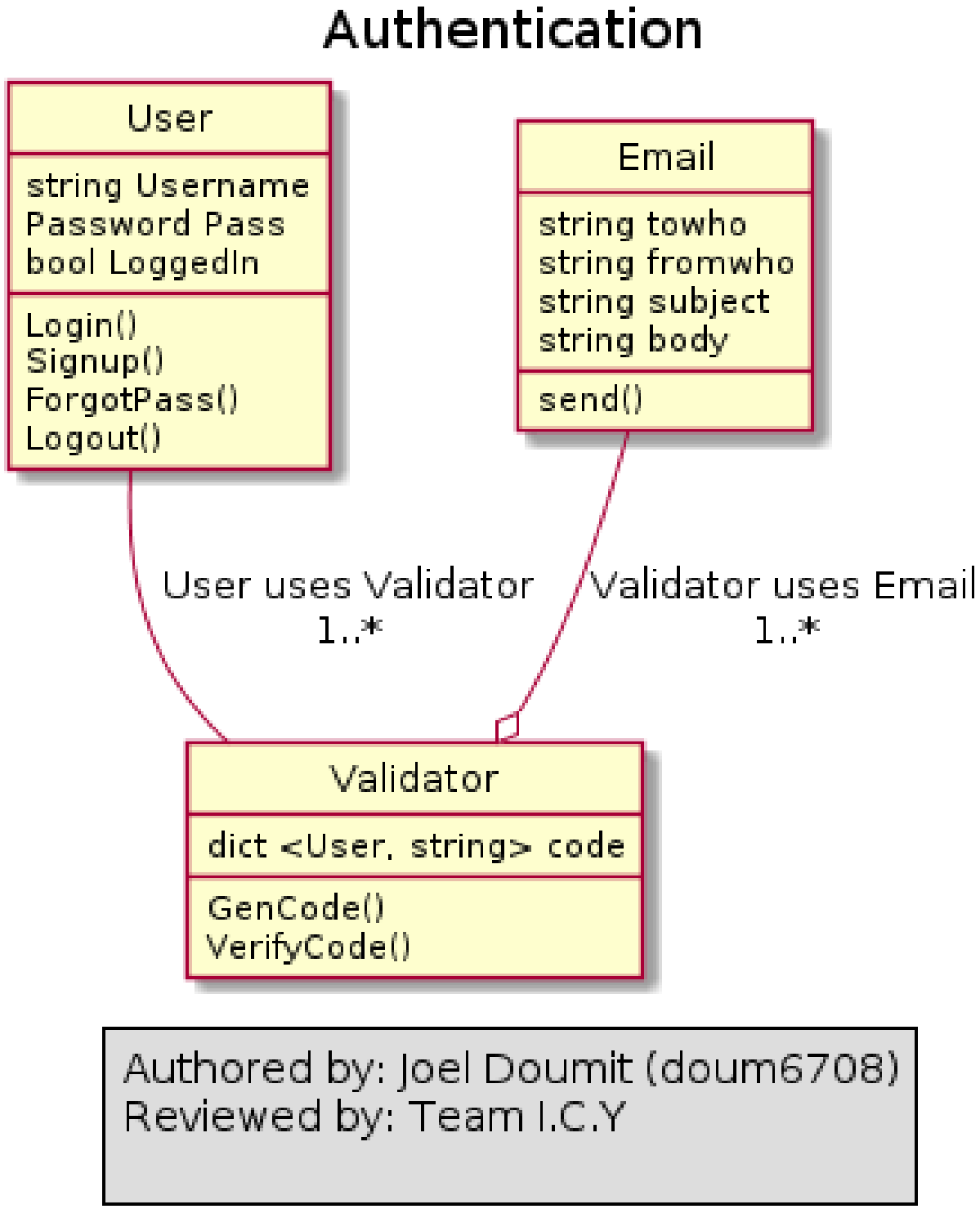
OperationalTransform:

I organized this one into a separate package because that's how I'm pretty sure we'll write it. The OperationalTransform algorithm is a collaborative editing algorithm used to allow multiple people to edit the same document at the same time, and keep the documents in sync. Ideally, we would use a pre-built library for this, as the algorithm is quite complex and there are lots of special cases, but I was unable to find one written in Java. Our best best will probably be to port an existing library in another language. The clearest, best documented implementation I found was OT.js (https://github.com/Operational-Transformation/ot.js/). The following classes are the main data structures implemented by this version of Operational Transformation.

- The *TextOperation* class represents a sequence of **Operation** objects, or changes. The TextOperation can then be applied to a string, or transformed with anther TextOperation (from another client) in order account for changes that occurred simultaneously. See the Operational Transformation algorithm for more details.
- The *Operation* class represents a specific operation. This contains an **OperationType**, a integer front, which specifies the number of characters before the change, a string s, which contains the actual character changed (ex, inserted or deleted). And then an integer back, which contains the number of characters until the end of the document.
- The *OperationType* Enum is used to specify what type of operation a **Operation** is. Type can be either an INSERT, when character(s) are inserted to a document, DELETE, when character(s) are deleted from a document, or RETAIN, used to shift other operations.

- the *Server* class is a class that will be running on the server to sync changes between clients. It's job it to listen for **TextOperations** from all connected clients, and when it receives one, broadcast that change to all connected clients.

3.4.5 Class Diagram 3: Authentication

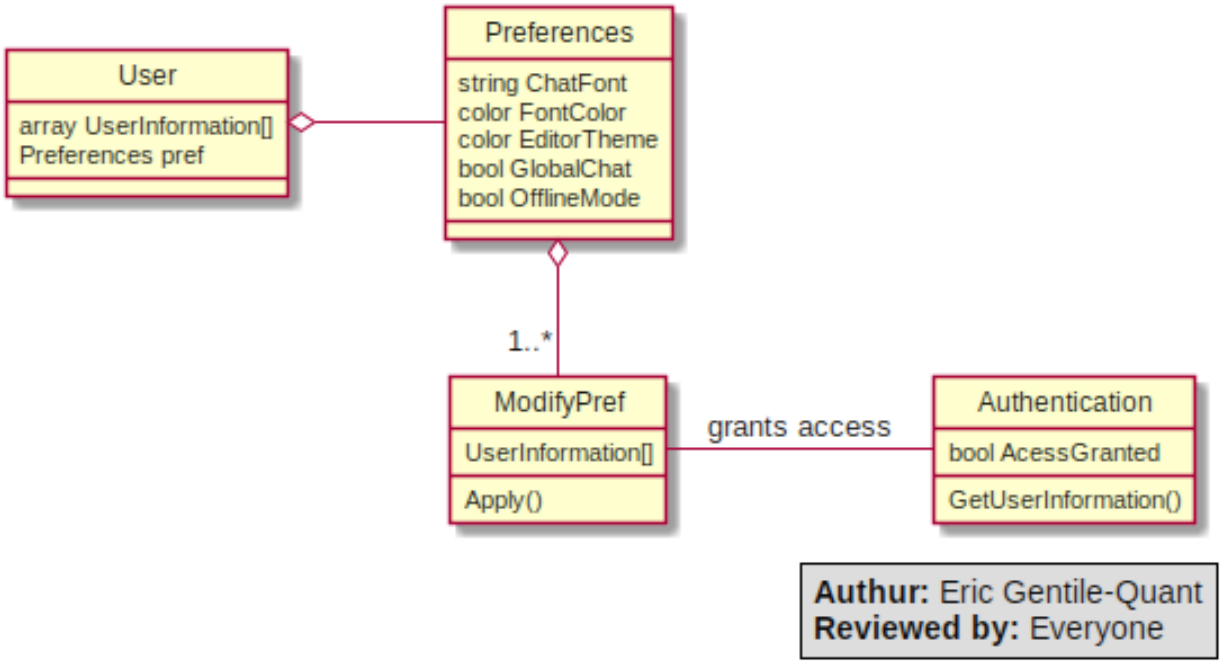


3.4.6 Class Diagram Description 3: Authentication Description

Classes:

- The **User** class represents the main user of the entire program. It details the basic information of each individual user, and allows each user the ability to create an account, to log into an existing account, and once logged in, to log out of the user account. It also allows a user to change his/her password, which involves the other classes.
- The **Email** class allows the program to send emails to Users who have signed up, or are signing up. It stores User information as a series of strings to be used by the **send()** function, which sends validation codes to Users' email addresses.
- The **Validator** class will run validation functions when called to do so by the User. Upon a User indicating they would like to change/have forgotten their password, it generates a validation code for that particular User, which it then stores in a dictionary. This validation code is sent to Users by means of the **send()** function denoted earlier.

3.4.7 Class Diagram 4: User Preferences

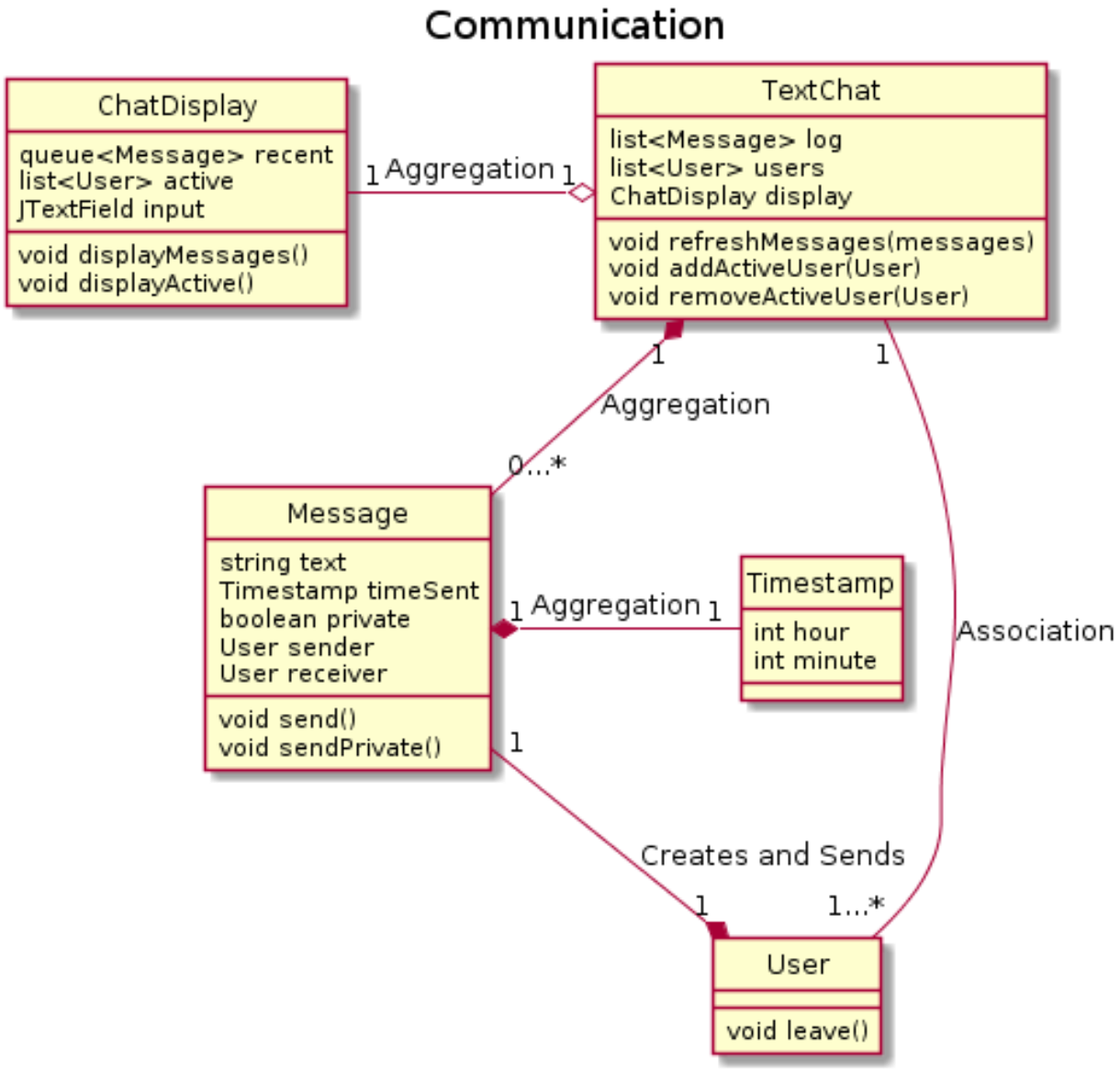


3.4.8 Class Diagram Description 4: User Preferences Description

Classes:

- **User:** Represents the human user of the program. It will hold the user's profile information so that it can be validated later.
- **Preferences:** Holds all the user's account preferences. This includes profile picture, chat font, chat color, ect.
- **ModifyPref:** Allows the user to modify his or her's preferences.
- **Authentication:** Authenticates the user's username and password to allow access to make changes on their account.

3.4.9 Class Diagram 5: Communication

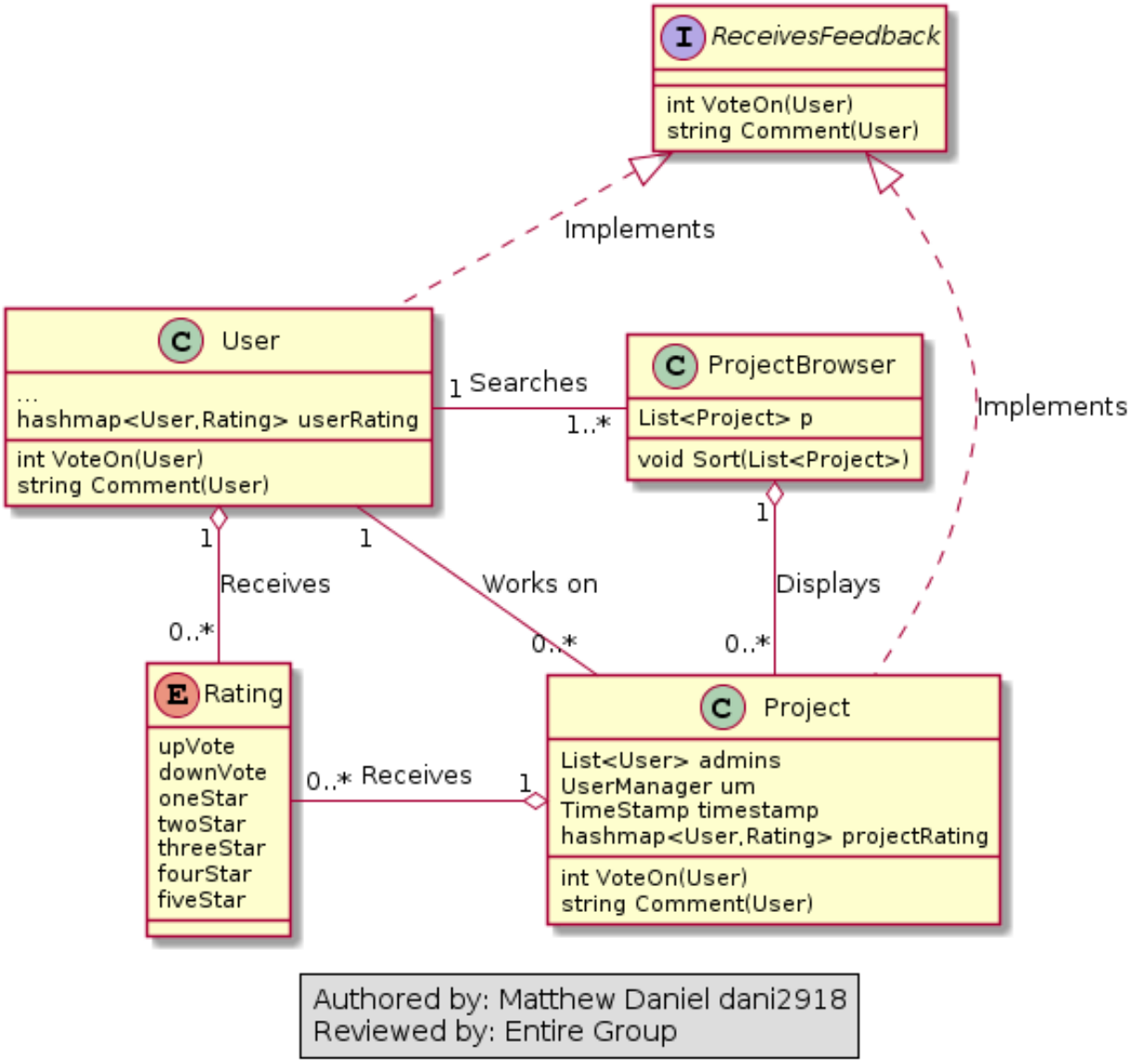


3.4.10 Class Diagram Description 5: Communication Description

Classes:

- **TextChat:** The master class to manage the messages, users, and display of the system.
- **ChatDisplay:** Displays relevant information including most recent 20 messages and active users
- **User:** Users interacting with the chat system.
- **Message:** Messages sent by users to TextChat, contain a string, timestamp, and sender/receiver data.
- **Timestamp:** Recorded time of when message was sent.

3.4.11 Class Diagram 6: Project Browsing



3.4.12 Class Diagram Description 6: Project Browsing

Enums:

- The *Rating* enum produces a value based upon the user's desired rating of another user or a project.

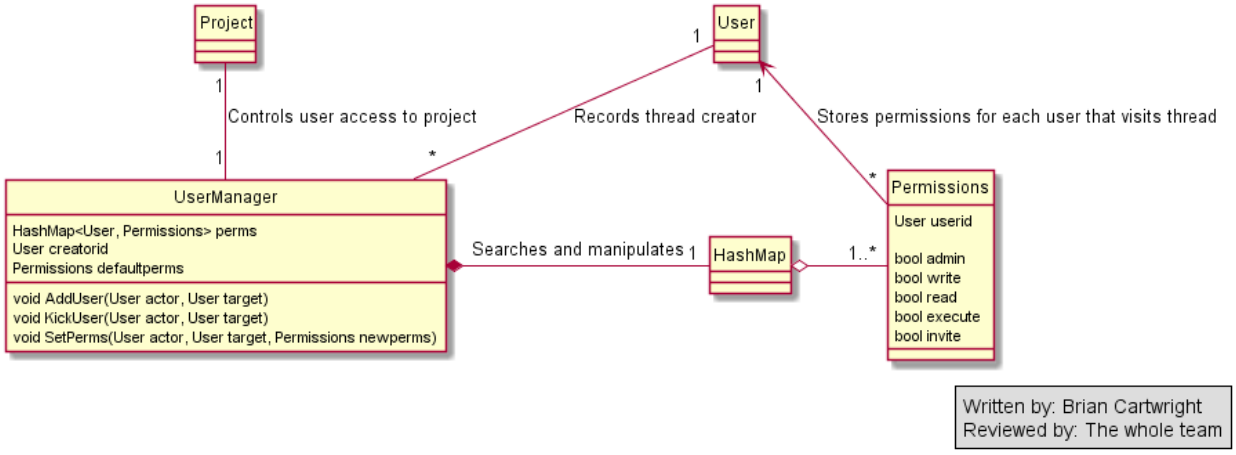
Interfaces:

- The *Receives Feedback* interface requires its implementers to implement two methods: **int VoteOnUser(User)**, which returns a value based on a user's rating, and **string Comment(User)**, which leaves feedback in the form of a comment.

Classes:

- The **User** class will be the class, shared across many of the class diagrams, that stores information about a user. The User class will have not only the methods and fields shown in this diagram, but a concatenation of the ones shown here and all other methods and fields from the other diagrams. Here, the User class implements the ReceivesFeedback interface so that other users may leave comments/reviews of a User object, and so that they may receive an accompanying rating from one to five stars. In relation to browsing projects, a User is the agent who searches a ProjectBrower object and works on a Project object.
- The **Project** class will also have the methods and fields of other class diagrams, similar to the User class above. Here, the Project class implements the ReceivesFeedback interface so that users may leave comments/reviews on projects, as well as vote up or down on projects that they come across. Projects are displayed in the ProjectBroswer class and worked on by users.
- The **ProjectBrowser** class contains a search-able list of zero or more projects tailored to a user's search. Users browse the list in order to find projects of interest.

3.4.13 Class Diagram 7: Project Member Management

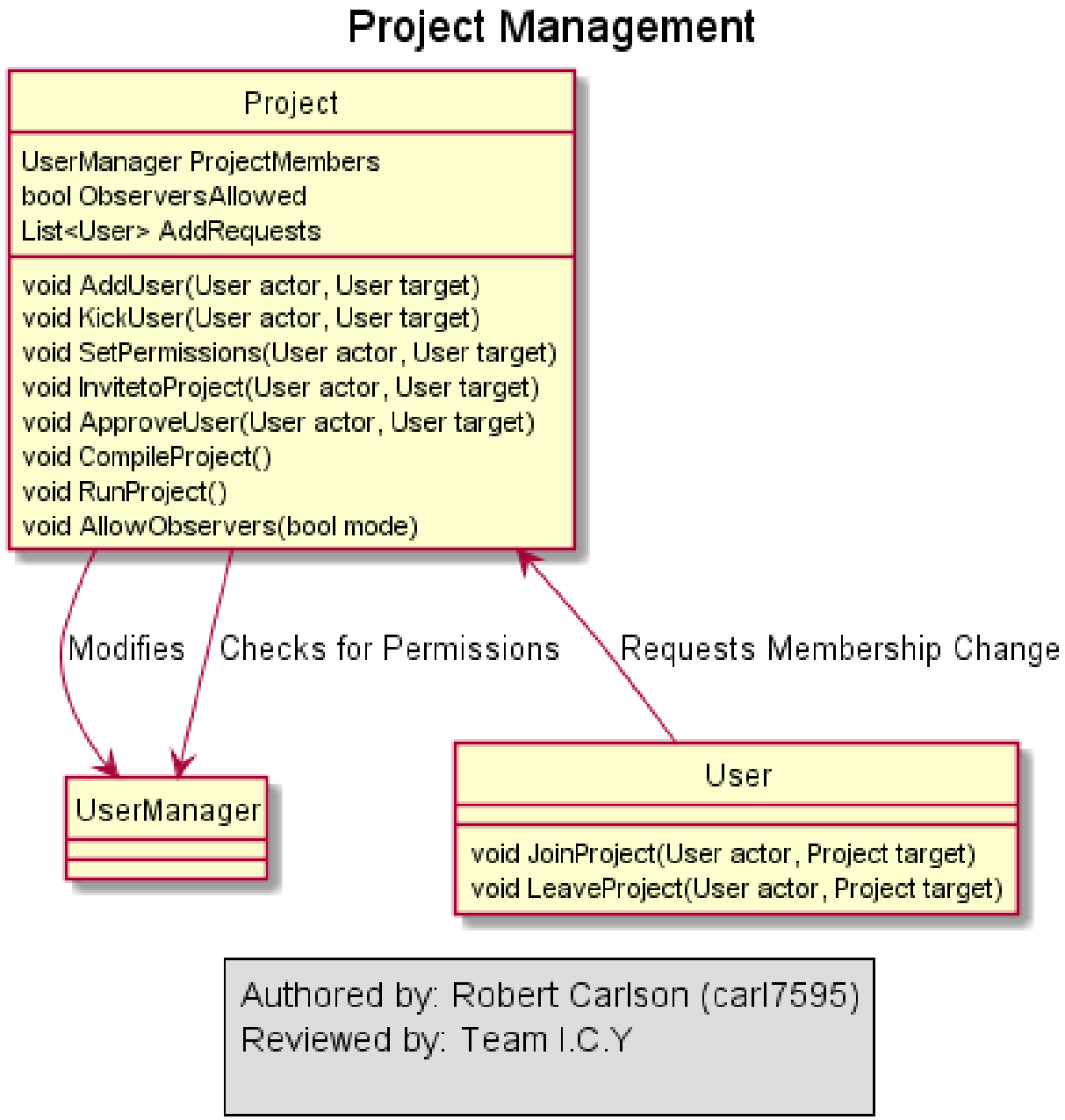


3.4.14 Class Diagram Description 7: Project Member Management Description

Classes:

- The **User** class contains the profiles of everyone who uses sQuire, and identifies anyone who tries to access a board.
- The **Project** class contains all of the information pertinent to an individual project, including one UserManager, which the client uses to control what kind of access each user has to that specific project.
- The **UserManager** class is referenced to check if a user has permission to read, modify, or run a project, or invite, ban, or change the permissions of another user. It does this by updating and checking against a HashMap of Permissions idexed by User. It records the User profile of the project creator to prevent the creator being demoted by another admin. It also contains a set of Permissions to use by default, before users are manually added to the project. The functions AddUser, KickUser, and SetPerms all modify the permissions HashMap after checking against it to make sure the active user has the authority to change the permissions of the target user.
- The **HashMap** class, in this case, functions as a permissions lookup table. It's indexed by User, and for each User in it there's one set of Permissions that it returns.
- The **Permissions** class is a set of bools that store whether each User has permission (within the instance's parent project) to read, write, execute the project, invite users, and/or modify the permissions of other users regarding the project.

3.4.15 Class Diagram 8: Project Management



4 REQUIREMENTS TRACEABILITY

This section shall contain traceability information from each system requirement in this specification to the system (or subsystem, if applicable) requirements it addresses. A tabular form is preferred, but not mandatory.

Feature Name	Req No.	Requirement Description	Priority	SDD	Alpha Release		Beta Release	
					Test Case(s)	Test Res.	Test Case(s)	Test Res.
	1.1							
	1.2							
	...							
	1.[n]							
	2.1							
	2.2							
	...							
	2.[n]							
	3.1							
	3.2							
	...							
	3.[n]							
...	...							
	[m].1							
	[m].2							
	...							
	[m.n]							

Priorities are: **M**andatory, **L**ow, **H**igh

SDD link is version and page number or function name.

Test cases and results are file names and **P**ass/**F**ail or % passing.

5 APPENDIX A. [insert name here]

Include copies of specifications, mockups, prototypes, etc. supplied or derived from the customer. Appendices are labeled A, B, . . . n. Reference each appendix as appropriate in the text of the document.

[insert appendix A here]

6 APPENDIX B. [insert name here]

[insert appendix B here]