CSCI 4448: Virtual Library Final Report

Team

- Matt Oakley
- Daniel Thompson
- David Ingraham
- Cory Cranford

Title: Virtual Library

Implemented Features

ID	Requirement	Topic	Actor	Priority
US-01*	As a user, I want to make an account with my email address so that my data is stored and associated with my account.	Sign-in	Standard User	Medium-High
US-02	As a user, I want to search for an existing book so that I can potentially add it to my virtual library.	Search	Standard User	High
US-03	As a user, I want to add a searched-for book to my virtual library so that I can keep track of my owned books.	User Abilities	Standard User	High
US-04	As a user, I want to select a book in my virtual library so that I can comment about, retrieve information, etc. about a specific book.	User Abilities	Standard User	High
US-05**	As a user, I want to retrieve the page-count of a book that I select so that I can see how large or small it is.	User Abilities	Standard User	Medium
US-06	As a user, I want to retrieve the author of a book that I select so that I can see who	User Abilities	Standard User	Medium

	wrote it.			
US-07	As a user, I want to retrieve the publisher of a book that I select so that I can see who published it.	User Abilities	Standard User	Medium
US-09	As a user, I want to be able to set a book as my “Favorite” so that I can frequently see it, comment about it, etc.	User Abilities	Standard User	Medium
US-10	As a user, I want to be able to to make a comment about a book in my virtual library so that I can record my thoughts about the book.	User Abilities	Standard User	Medium
US-11	As a user, I want to be able to rate a book in my virtual library so that I can record my rating/sentiment about the book.	User Abilities	Standard User	High
US-19	As an admin, I want to make an account with my email address.	Admin Abilities	Admin User	High
US-21**	As a user, I want to retrieve the genre of a book so that I can see what genre the book is.	User Abilities	Standard User	Medium
US-22**	As a user, I want to retrieve the average rating of a book.	User Abilities	Standard User	Low
NFR-01	Upon adding a book to their library, it should appear in the library within 3 seconds.	Performance	Standard User	High

*****Denotes partly-implemented features

- US-01: User objects are created but are not persisted in a database

** Denotes added/changed functionality from original use case document

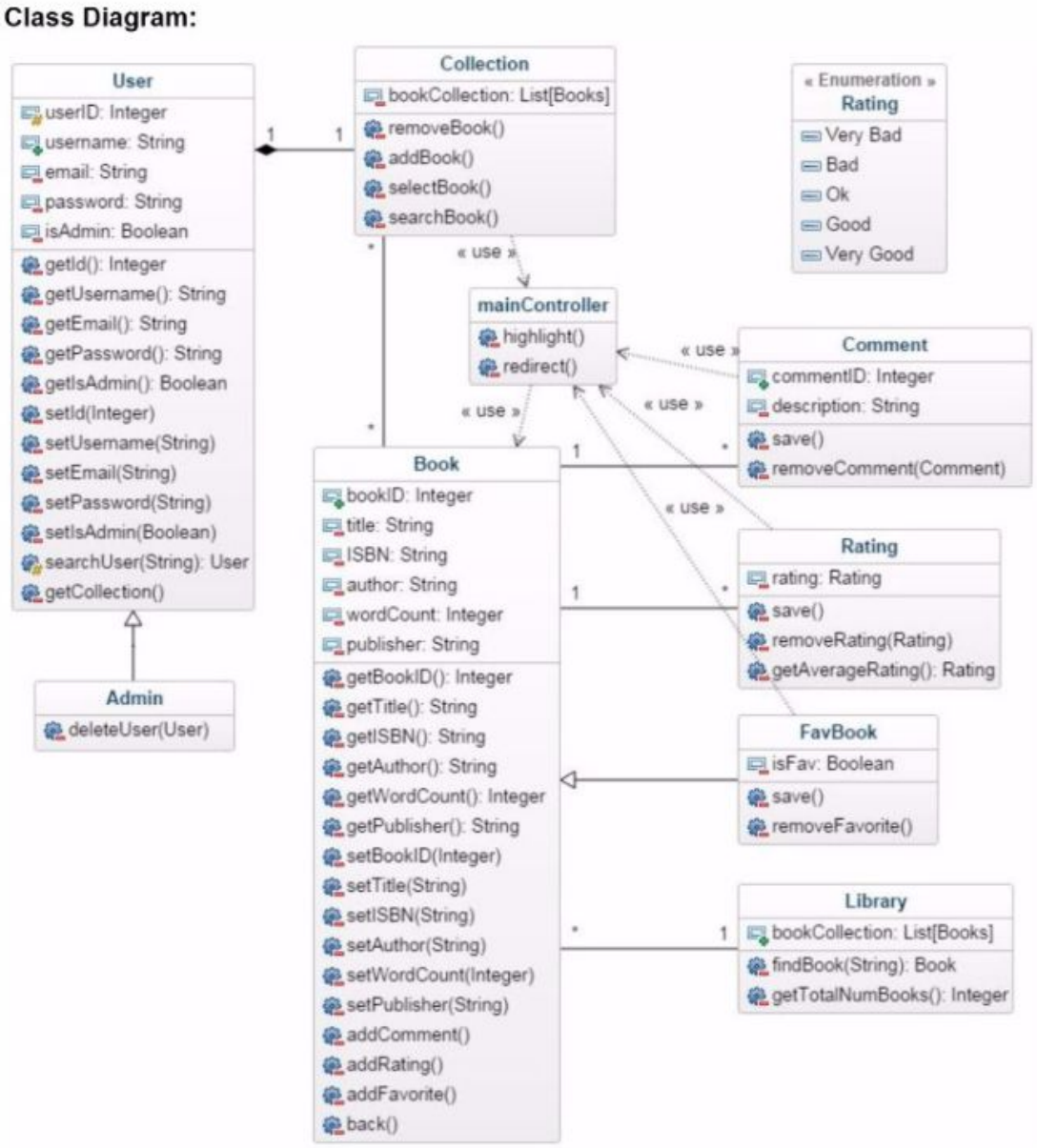
- US-05: Changed “word-count” to “page-count”
- US-21: Added genre attribute instead of ISBN attributes
- US-22: Added functionality to retrieve average rating

Non-Implemented Features

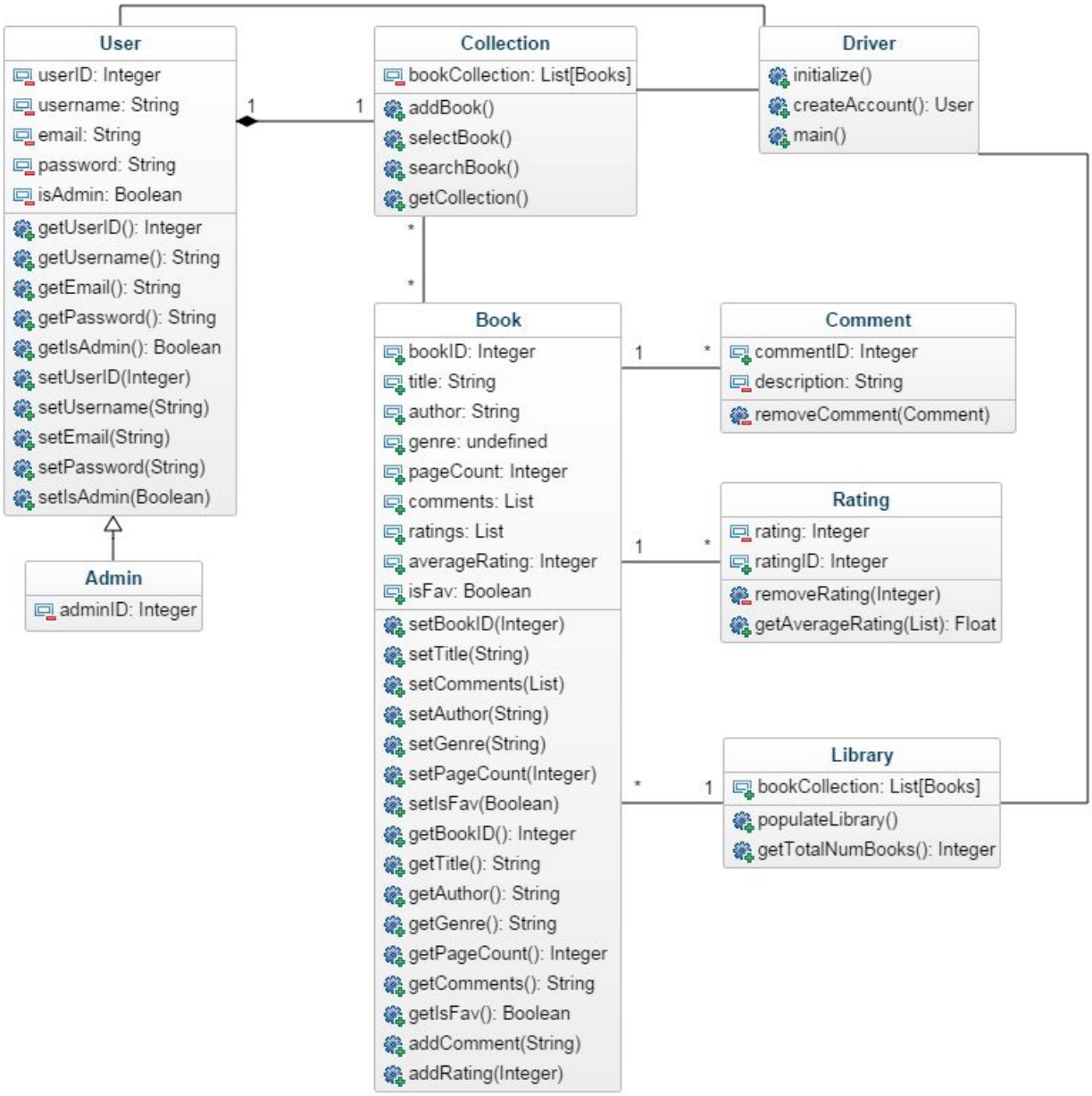
ID	Requirement	Topic	Actor	Priority
BR-01	The database which stores books will be updated to include new books on a monthly basis.	Data Update	None	Medium
US-08	As a user, I want to retrieve the ISBN number of a book that I select so that I can keep track of it.	User Abilities	Standard User	Low
US-12	As a user, I want to remove a book from my library so that I can delete books that I accidentally add.	User Abilities	Standard User	Medium-High
US-13	As an admin, I want to search for a user based on their username so that I can view information about their account.	Admin Abilities	Admin User	Medium-High
US-14	As an admin, I want to update a user’s comment about a book so that I can manipulate poor comments.	Admin Abilities	Admin User	Medium
US-15	As an admin, I want to delete a user’s comment about a book so that I can monitor and/or get rid of bad comments.	Admin Abilities	Admin User	Medium
US-16	As an admin, I want to manually remove a book from a user’s library so that I can keep their virtual library accurate.	Admin Abilities	Admin User	Medium

US-17	As an admin, I want to delete a user so that I can get rid of malicious users.	Admin Abilities	Admin User	Medium
US-18	As a user, I want to search for another user based upon their username so that I can view their virtual library.	User Abilities	Standard User	Low
US-20	Upon creating an account, the user is registered with the system.	Register	Standard and Admin User	High
NFR-02	The user's email, password, username should be securely stored in a database.	Security	Standard User	High

Original Class Diagram



Final Class Diagram



What changed:

- Removed deleteUser() method from Admin class
- Added adminID attribute to Admin class
- Replaced mainController class with Driver class
- Removed FavBook class and added this functionality to Book class
- Removed save() methods from Rating and Comment classes
- Removed Rating enumeration and handled ratings via integers

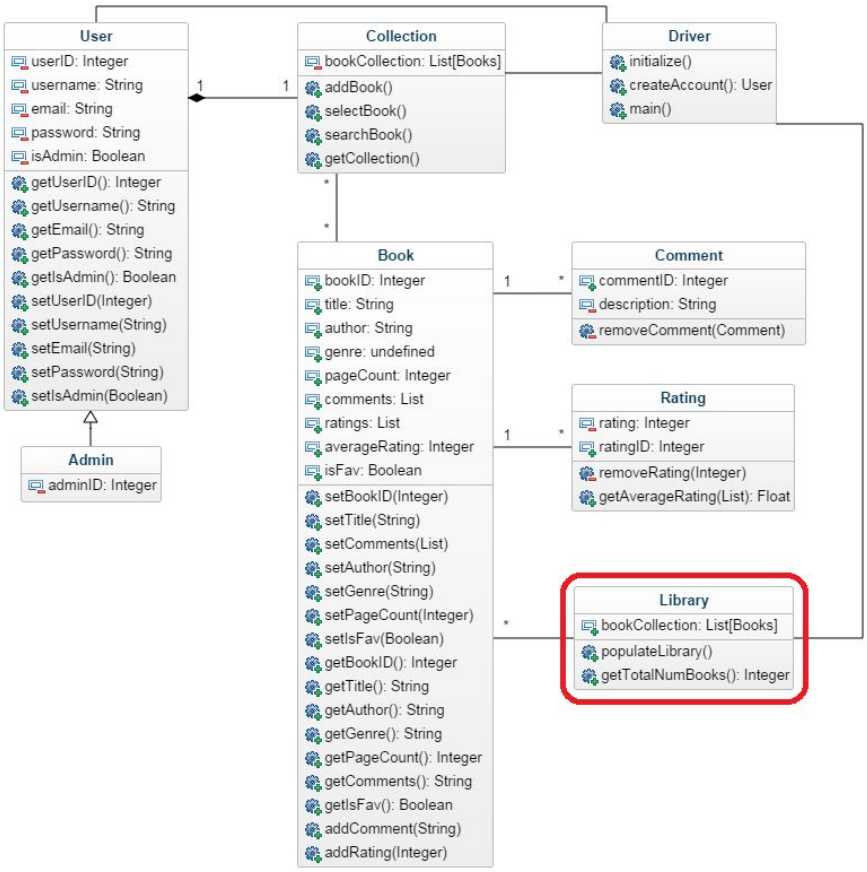
- Changed several attributes/methods from private to public
- Replaced findBook() method in Library with populateLibrary() method
- Changed rating attribute in Rating class from a Rating object to an integer
- Added ratingID attribute to Rating class
- Added getCollection() method to Collection class
- Changed attributes of Book class (ISBN, wordCount) to attributes which reflected our data (genre, pageCount, etc.)
- Removed back() method from Book class

Why:
The first large change we made was with respect to the attributes within the Book class. We wanted attributes such as ISBN number and number of words but could not find the proper data. Therefore, we changed the attributes within the Book class to reflect the data that we gained access to. Another large change we made was getting rid of the mainController class and replacing it with the Driver class. We initially intended to use MVC for our project which is what led us to including a Controller class in our original class diagram. However, we eventually decided on having our program run purely from the command line and created the Driver class to act as the mainController we originally had. Additionally, we removed the deleteUser() method from the Admin class due to the fact that this was not the proper way to specify permissions. Other small changes were made to our class diagram during development which are outlined in the section above.

Even though we didn't have to change a lot about our design, the design was crucial in guiding our development. Having a strong and cohesive class diagram allowed us to quickly build the shells for our classes. This was useful because our focus could remain on implementing the class functionality instead of worrying about the design of them. Doing a design up front also ensured that the entire team was on the same page and understood the purpose and interaction of each class. This ended up preventing duplicate code and bugs that may have occurred if there was any confusion about the functionality of each class. Lastly, it was very beneficial to be able to refer to our class diagram when we were deciding what methods/attributes to put where. Instead of having to make this decision during development, we were able to quickly refer to our class diagram and get an answer. Overall, doing this design upfront greatly improved our efficiency in terms of development.

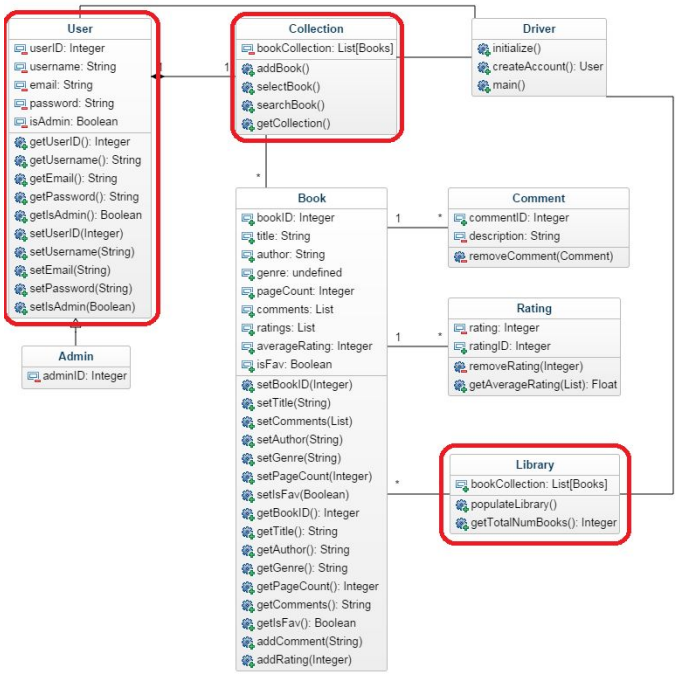
Design Patterns

While our team had limited experience with Java and OOD which led us to mainly focus on development, we did make a concerted effort to incorporate design patterns into our project. The content below outlines which design patterns we did implement as well as the ones we thought about implementing.

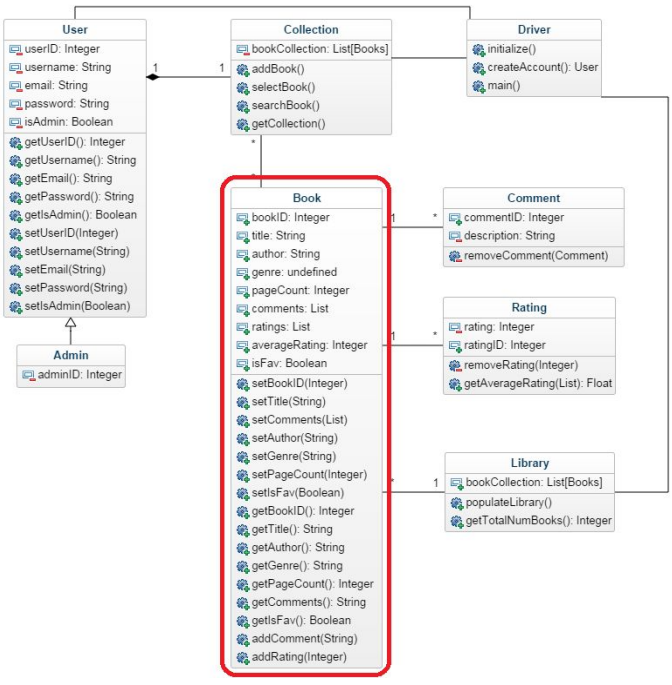


Singleton: Our project reads from a database which contains a large number of books. While each user has their own, unique “Collection” of books they own, we wanted to also create a generic “Library” class which would represent the current state of our database. Since there would only be one instance of this class, we thought this lended itself to the Singleton design pattern. We accomplished this by having one private attribute in the Library class (bookCollection) which would be instantiated once (and only once) the program began. We also included a couple methods which interact with this class such as retrieving the total number of books available and listing out all of the books which exist in our database.

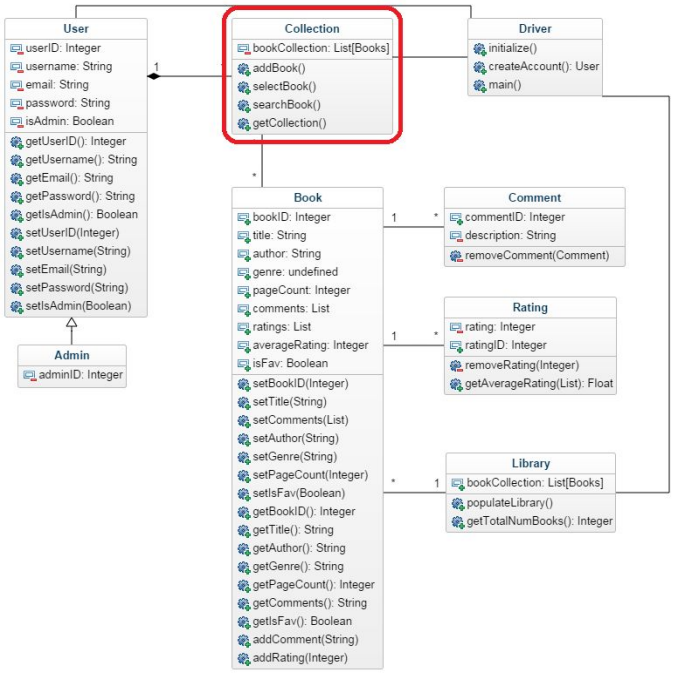
Proxy: Proxy, specifically Remote Proxy, is something that we originally wanted to implement but did not get around to doing. The reason we wanted to implement this was because we thought signing in, re-instantiating the Library class each time the program runs, etc. could all be cached with the user and would save on expensive and potentially unnecessary operations. If we were to implement it, the classes associated with this design pattern would be the Library and User classes.



Flyweight: This is a design pattern we thought would work well for our project but ultimately found out that it would not. Our reasoning was that we were dealing with a very large number of Book objects which seemed like it would lend itself to the Flyweight design pattern. However, we realized that this wouldn't really work because each Book object has very different extrinsic attributes and very few intrinsic ones. Therefore, we ultimately decided against using this design pattern for our project.



Strategy: This is another pattern that we thought would work provided we had more time/resources/experience. Strategy would be a part of our Collection class which contains a sort of algorithm which would be slightly manipulated provided the types of media in the Collection class. If the user's Collection was able to store other types of media such as MP3s, videogames, and movies instead of just books, the step of the algorithm associated with interacting with the media would change. The viewing and selecting steps of the algorithm would be the same, but the interaction step would be different depending on the media. MP3s would use playMusic(), videogames would use startGame(), movies would use playMovie(), etc.



Reflection

While completing this project our team learned that the most important consideration for this type of project was to have a clear and concise idea about the design going into it. The amount of time that we saved during development was remarkable. When creating our class diagram we were unsure if it would truly help when we transitioned into development. Fortunately, we found it to be extremely helpful. It was very convenient to quickly add methods and attributes to our classes just by simply looking at our class diagram. In addition to the class diagram, creating a use-case document at the beginning of our project was very helpful. Having all of these use-cases upfront made development a lot quicker and more efficient. It also seemed like we were getting things done at a faster pace in comparison to other group project we had been on. Being able to treat our use-case document as a "checklist" and splitting up large functionality into separate, individual use-cases was very helpful. It lessened how abstract our project was and cleanly laid out what needed to be implemented.

Design patterns were also an interesting thing to both learn about and attempt to implement. Taking an abstract idea like a design pattern and converting it into code proved to be more difficult than we were anticipating. Having our class diagram handy

made choosing design patterns a bit easier but, as previously stated, implementing them correctly was a bit more difficult. If we had more time to work on the project we would definitely like to implement some accurate design patterns into our project such as Remote Proxy and Strategy.

In addition to the analysis and design that we experienced over the course of this project, learning about Java and OOP was something we all appreciated. Both Java and the concept of OOP seemed daunting at first because our group had very limited experience in this context. However, we soon learned that Java was nowhere near as scary as it initially seemed and that it lended itself to OOP very easily. Creating and referring to our class diagram also greatly helped get us started developing.

What we learned most through doing this project is the importance of design. All of us admitted to getting frustrated with previous group projects due to the lack of guidance and direction of the projection itself. In hindsight, we can fortunately say that we did not experience this during this project. Designing our project upfront proved to be extremely beneficial and is something that we will be sure to do on future projects and in industry.