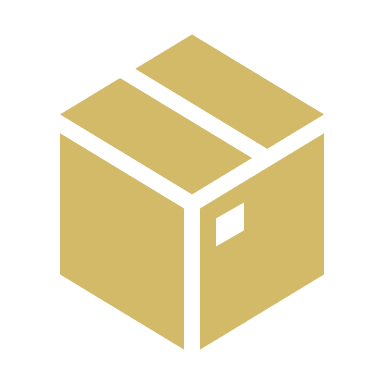
10/7/2019

****

Nathaniel Galbreath nrgalbreath2@mail.fhsu.edu Brandon Nolan btnolan@mail.fhsu.edu Sarah Pauly slpauly@mail.fhsu.edu Logan Wiley lbwiley@mail.fhsu.edu

OrderTracker

Team #5

****Contribution Breakdown****

* Brandon Nolan
* Class Diagram and Interface Specifications
* Design Patterns
* Analysis and Design
* UML Diagram
* Cover Page/ Individual Breakdown/Design of Documentation
* Table of Contents
* Logan Wiley
* System Architecture and System Design
* Architectural Styles
* Mapping Subsystems to Hardware
* Sara Pauly
* System Architecture and System Design
* Persistent Data Storage
* Global Control Flow
* Nathan Galbreath
* System Architecture and System Design
* Identifying Subsystems
* Hardware Requirements
* User Interface Design and Implementation

**Table of Contents**

Contribution Breakdown 1

Table of Contents 2

Glossary of Terms 3

Customer Statements 4-6

*System of Requirements* 6

Functional Requirements Specifications 6-11

*Stakeholders* 6

*Actors and Goals* 6-7

*Use Cases* 7-11

User Interfaces Design and Implementation 11-16

Domain Analysis 16-19

Project Estimation for Use Case Points 19-21

Plan of Work 21

Interaction Diagram 22-26

Class Diagrams and Interface Specifications 27-30

*Design Patterns* 27

*Analysis & Design* 28

*UML Diagrams* 29-30

System Architecture and System Design 31-33

*Architectural Style* 31

*Identifying Subsystems* 31

*Mapping Subsystems To Hardware* 32

*Persistent Data Storage* 27

*Global Control Flow* 28

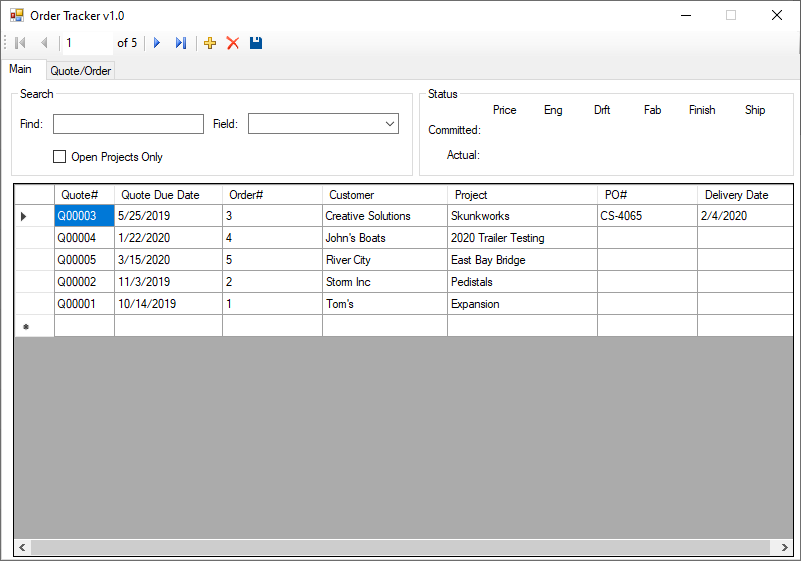
*Hardware Requirements* 28

References 21

**User Interface Design and Implementation**

The user interface is designed is a way to be intuitive and user friendly. Minimal data is needed by the user and one click is all that is needed by the user when “Logging” data for Start/Finish Logs.

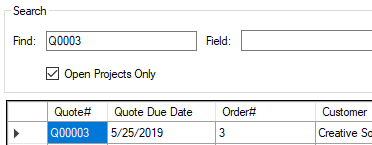
The main display is a grid-based view for simplicity along with a direct Quote/Order visual representation of location when a Quote/Order is selected.



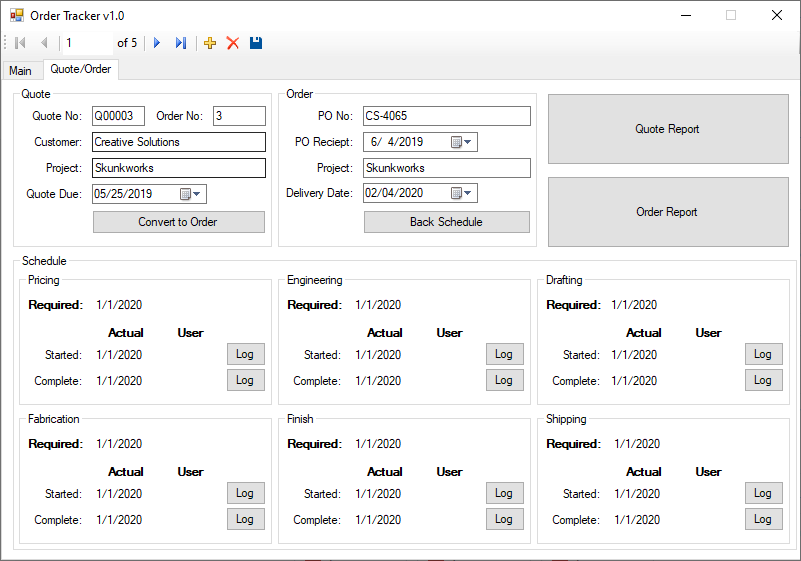
Users will be able to select a Quote/Order and see a Status that shows where the project is located and if it has been tracking on time (Green) or late (Red).



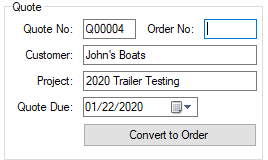
The main screen will also contain a Find or Search allowing the user to quickly narrow down the results to what they are looking for. This will have the ability to limit the Field that is being searched. They can also limit it to only Open Projects.



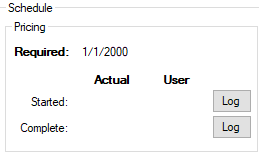
Aside from the main screen there will be a Quote/Order screen used for Quote/Order entry along with a Schedule where users can Log there committed Start and Completion dates into the system.



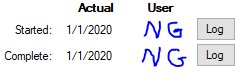
For Quote entry users will be required to enter a Customer Name, Project, and Quote Due date. Quote Number should be auto generated.



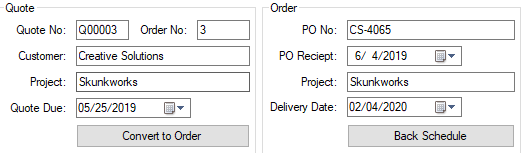
At this point the system will create a quote in the system a set a required date of completion based on the Quote Due date.



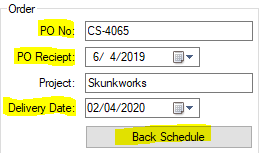
When a User Starts or Completes a quote, they will select the “Log” button to log their date/user for tracking purposes.



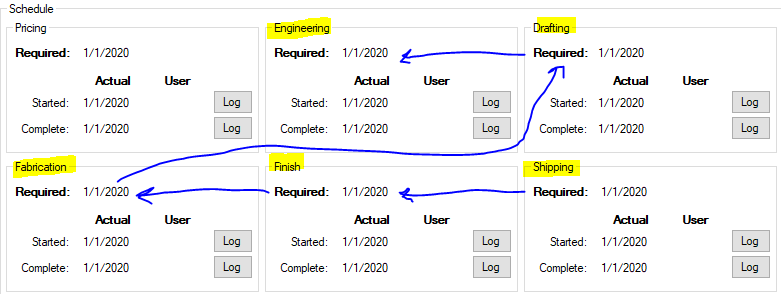
When the user receives a purchase order for the project, they will use the “Convert to Order” button to begin the order entry process. This will assign an Order Number.



The user will be required to enter the customer PO#, PO Receipt date, and Delivery Date for the project. Once logged the user will select the “Back Schedule” button to back schedule the requested dates for the rest of the business users to complete.



Back Scheduling will take the delivery date and schedule Engineering, Drafting, Fabrication, Finish, and shipping to meet the delivery address. Users of each department will Log their time the same way as Pricing for a quote.



The back Schedule and User Log’s is what will drive the visual display for tracking.



The user will also have the option to view a Quote/Order report. This will likely be added as an additional tab and not driven off buttons like in the current display. This will be displayed with a grid view based on open Quotes/Orders prioritized by Quote Due/Delivery Dates. This should display sim\ilar to the below with one reporting grid for open Quotes and another reporting grid for Orders

**Interaction Diagram**

***UC1 – User Login:***

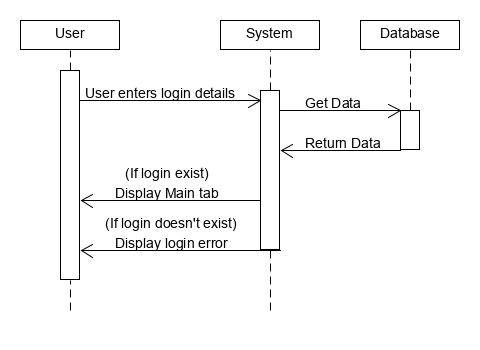


Figure 7-1 Sequence Diagram for Use Case 1

Figure 7-1 shows the sequence diagram for Use Case 1 which is logging in. This is used to monitor who is logging into the system to limit who can and can’t access sensitive information to the company. The user will first enter log in details provided to each employee by the admin (Initials/Password). The system to verify the information against the database and then display either a login error or bring the user to the Main tab of the program and load the data into the grid.

***UC2 – Quote/Order Entry:***

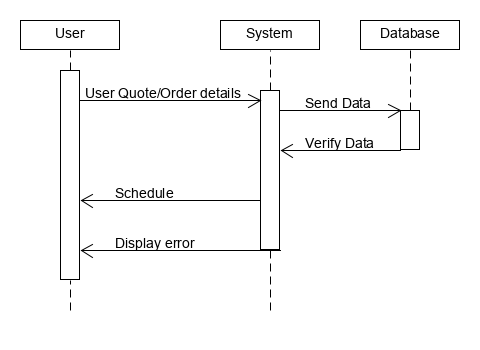


Figure 7-2 Sequence Diagram for Use Case 2

Figure 7-2 shows the sequence diagram for Use Case 2. It is used for getting the Quote/Order information from the user. First the user will enter the Quote/Order information needed and then select “Save”. This will send the data to the database and return a schedule. If there is a scheduling conflict the system will notify the user with an error message.

***UC3 – Reporting:***

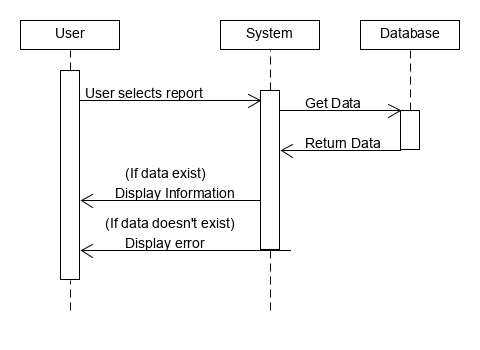


Figure 7-3 Sequence Diagram for Use Case 3

Figure 7-3 shows the sequence diagram for Use Case 3. It is used for getting the Quote/Order reports from the system and display them to the user. First the user will select the Reporting Tab and then select Quote Report or Order Report. This will request and “open” or active Quote/Order data from the database and return organized information to the user. If there are no Quotes/Orders “open” or active, then the system will notify the user with an error message.

***UC4 – Searching:***

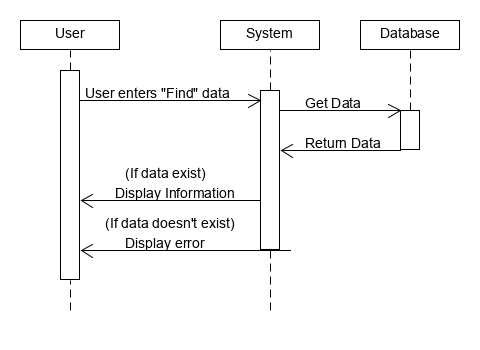


Figure 7-4 Sequence Diagram for Use Case 4

Figure 7-4 shows the sequence diagram for Use Case 4. It is used for searching the Quote/Order data stored in the database and the system will display the results to the user. First the user will enter data into the “Find” text box on the Main tab. A “live” search will begin to narrow the results in the grid for the user to see. If there are no results from the “Find” text box, then the system will notify the user with an error message.

**Class Diagram and Interface Specifications**

***Design Patterns:***

In reviewing the different design patterns, the consensus was that there were two design patterns that most commonly represented our project. The main idea of state design is when an objects behavior depends on its state; it allows the object to change the behavior without changing the class. With this in mind, it should allow for the code to remain cleaner from a visual standpoint for future ease of use. On the other hand, the command pattern was also a pattern that was implemented in our system. This simple means the pattern is intended to encapsulate an object for all the different data require for that said given action or command.

In accordance to our functional features for what we are wanting to accomplish with this system, state design allows the business to view prioritized quote report based on the consumers need of date. This will not only satisfy customer needs leading to more retention for the business, but also will improve workflow for employees and the task to be set forth. This will aid employees to allocate their work in accordance to consumers’ needs with shipment of products. The state design will help our system work hand in hand with employees to make the overall experience of the business more efficient for all parties involved. There will end up being three states here: the order entry, the order details, and the prioritize of time based on customer date needs. The order entry will allow the customer to interact with the system to enter the customer need. The order detail state will be separate but will be the bridge point between the order entry and the allocation of time worked. Order detail will list every piece of information that was listed from the customer in the order entry area. In the prioritizing of time state, employees will be able to allocate time frames based on the information given from the customers. The system will work with the employees to give an understanding of how to allocate workloads based on the dates needed for delivery from the customers order entry and order details. Overall, the state design seems to be the most effective design pattern to implement with our system and aiding the customers and employees with knowing what to do in the different states.

The second design pattern that meshed well with our system was the command design. This design allows the customer to interact with the system for the needed information that they are requesting. This will allow us to implement items/features that will allow the consumers to view updates on their order. It will us to implement and give customers the opportunity to view the orders location, which team is carrying out the current workload at said given time, start and completion times, and if an order is behind schedule. By implementing this into the system, it will hit on our main priority of making it have an ease of use feel for any intended users.

***Analysis and Design:***

|  |
| --- |
| Design |
| Order: |
| -placementDate: Date  -orderNumber: int  -deliveryDate: Date |
| -preciseLocation: Location  -carryingTeam: Team  -startCompletion: Time  -orderBehind: Time |

|  |
| --- |
| Analysis |
| Order: |
| Order Entry  Order Details |
| Prioritize Needs |

***UML Diagrams:***

**State Design:**

|  |
| --- |
| State = PackageState |
| next() |
| prev()  printStatus() |
|  |

|  |
| --- |
| Context = Package Class |
| previousState() |
| nextState() |
| printStatus() |

|  |
| --- |
| DeliveredState() |
| next() |
| prev()  printStatus() |
|  |

|  |
| --- |
| RecievedState() |
| next() |
| prev()  printStatus() |

|  |
| --- |
| OrderedState() |
| next() |
| prev()  printStatus() |
|  |

**Command Design:**

|  |  |  |  |
| --- | --- | --- | --- |
| Customer | Business | Tracking | Customer |
| (client) | (invoker) | (command) | (receiver) |

ViewOrder()

OrderNumber()

OrderPull()

Example:

TeamCarry()

OrderLocation()

Order Date

#22361

- Description of

Items

StartCompletion()

OrderBehind()

**System Architecture**

***Architectural Style:***

Our group used event-driven architecture in the design of our system. An event can be defined as "a significant change in state". For example, when a consumer purchases a car, the car's state changes from "for sale" to "sold". In our case when you change from shipped to delivered by our order tracker.

***Identifying Subsystems:***

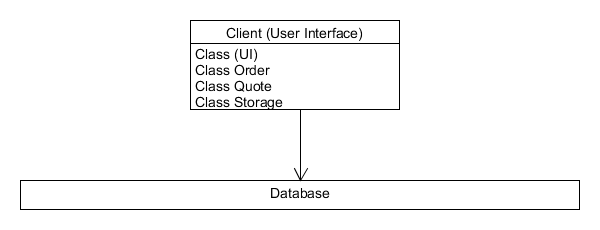


Figure b-1 UML Subsystem Diagram

In this system there are two subsystems. There is the Client subsystem and a Database subsystem. The client subsystem contains the structure for the user interface. In the Database subsystem the Client side sends a request, the request is handled on the Database side, then the result is sent back to the Client side.

***Mapping Subsystems to Hardware:***

The system will have to be to run on multiple devices. In particular, the system will have to be able to communicate with the order tracker. The order tracker will be run on an internal server with multiple computers being able to connect to the order tracker.

***Persistent Data Storage:***

The Order Tracker system must save data so that the data will outlive a single execution of the system and be prepared for the next execution. (i.e. date, time, employee quotes, reports, tracking). The database is used to determine the status of an order that an employee has placed. As we want our system to be error tolerant, we need to make sure order information can be stored and updated in a proper pattern.

The system must store all the information concerning current and past orders or employees in a persistent storage. All operations connecting the information of an order must operate using transactions: at the lower level for updating only one value from the database, relational SQL databases have trans-actionality, for the upper level.

We only want to update the information about an order if the designated order information to be altered has been completely entered by the employee. The simplest method to use is a relational database, MySQL and store the information as tables.

|  |  |
| --- | --- |
|  | Order Tracker |
|  | Pricing |
|  | Engineering |
|  | Drafting |
|  | Fabrication |
|  | Finish |
|  | Shipping |

***Global Control Flow:***

Our system is an event-driven system that waits in loops for events and the user can generate the actions in different order with each use.

The system is an event-response type where there is no real concern for time constraint on the user, besides the scheduling of a time for an order by another employee. The system does not require a time dependency.

When there are multiple users, the integration of their usage will cause concurrency to be an issue. When multiple employees are utilizing the system to schedule orders, the systems database will need to update the storage often to ensure proper scheduling of orders for certain departments to allot.

***Hardware Requirements:***

* Windows based PC
* Minimum 1gb of storage (Expand as needed for database)
* Network connection

The system is built to run on a windows-based PC. For initial installation it is recommended to have at least 1gb of space available for database storage. Storage will need to be expanded as the database reaches its limits. A network connection is required for future updates/syncing.

**References**

Software Engineering Book

* <https://ww.ece.rutgers.edu/~marsic/books/SE/book-SE_marsic.pdf>

Parking Lot Automation (Rutgers)

* <https://www.ece.rutgers.edu/~marsic/books/SE/projects/ParkingLot/2013-g5-report3.pdf>

Traffic Monitoring Report (Rutgers)

* <http://www.ece.rutgers.edu/~marsic/books/SE/projects/Traffic/2011-g7-report3.pdf>

Gantt’s Chart: “How to Make a Gantt Chart in Excel.” Office Timeline

* <https://www.officetimeline.com/make-gantt-chart/excel#tutorial-auto>.