Software Architecture and Design

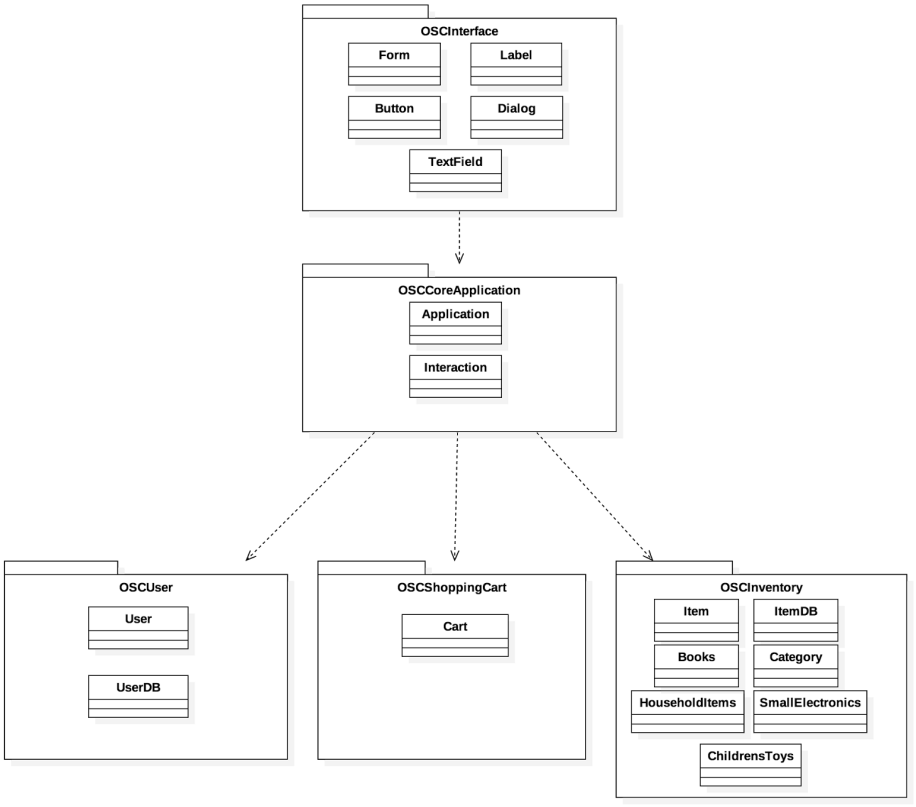
Assignment 3

Dr. Byron Williams

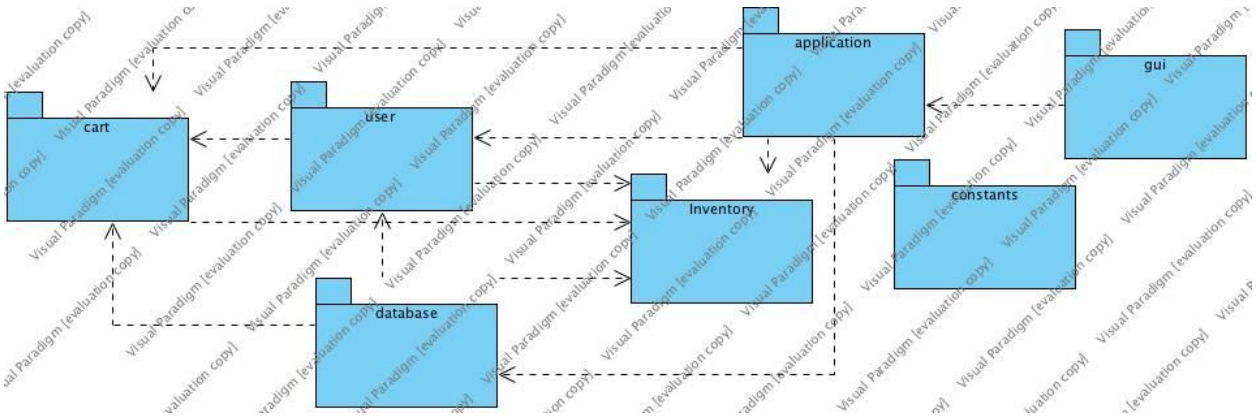
September 14th, 2016

JJ Kemp (jhk114)
Hannah Church (hds109)
Sam Maxcy (sam682)
Dylan Jaye (dtj74)

Original Package Diagram



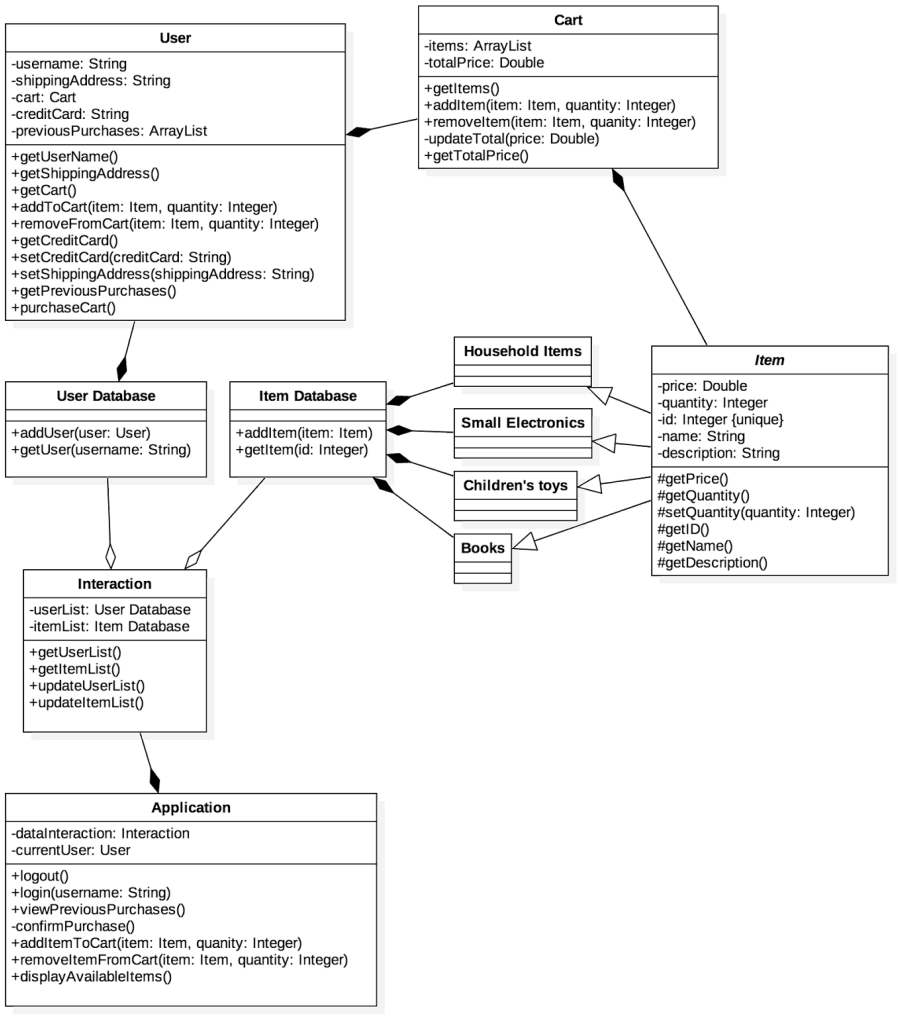
Reverse Engineered Package Diagram



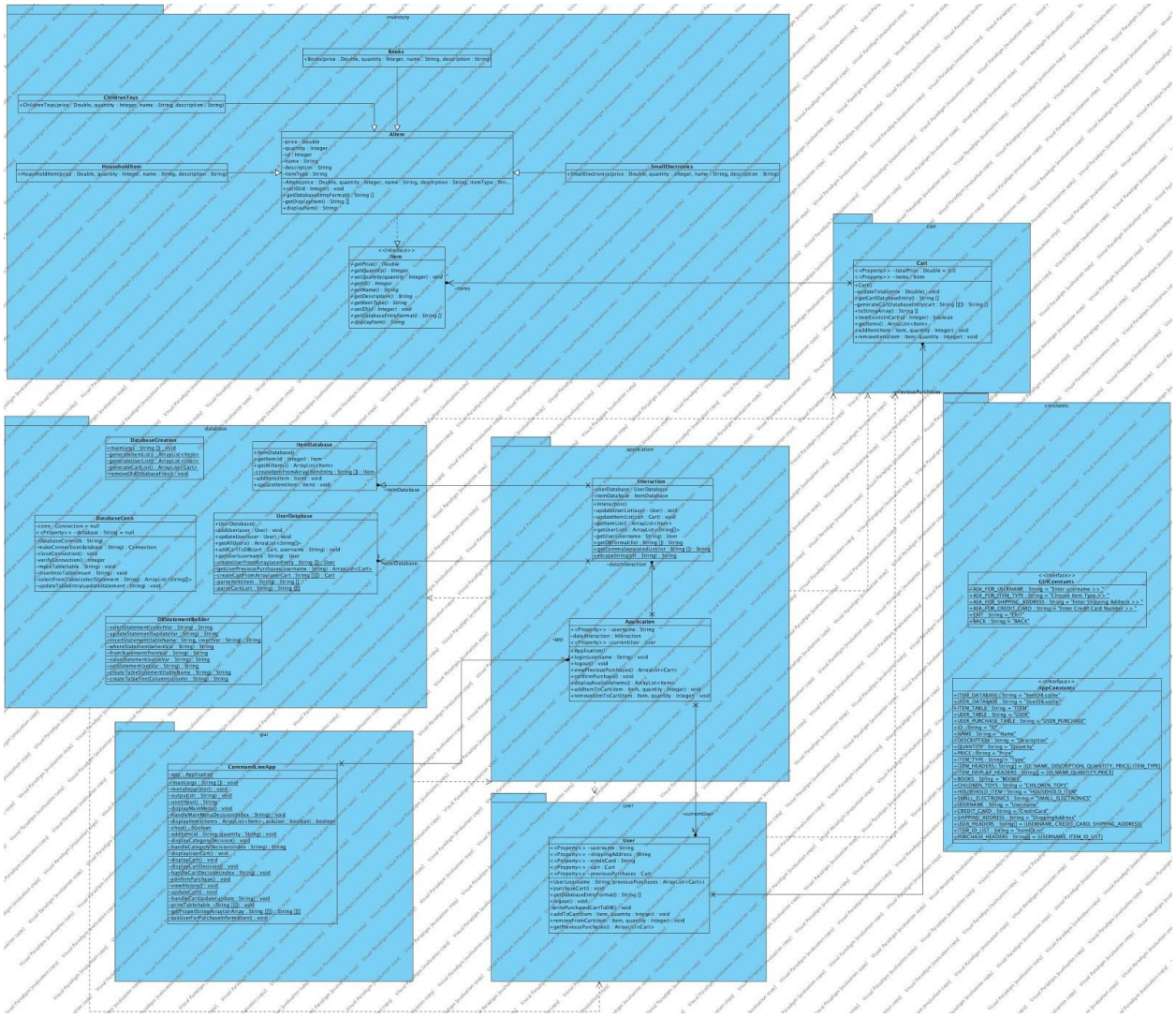
The RE package diagram closely resembles the original package diagram. The original diagram shows Application, User, Cart, Inventory, and Interface packages. The implementation resulted with all of the original packages as well as two extra packages: Constants and Database. The prescriptive architecture specifies individual classes for database connections, however the descriptive architecture ended up containing an entirely separate database package for modularity. The constants package contributes to the system by keeping all commonly used constants in a central, easily accessible location. There are many more connections than

originally perceived as well. In the RE diagram, nearly every package interacts with the other packages. Originally, there were only interactions between Interface and Application, and also between Application and User, Cart, and Inventory. Overall, however, the architecture remained highly similar to the original design.

Original Class Diagram



Reverse Engineered Class Diagram



The RE diagram shows multiple dependencies that were not shown in the original diagram. Cart is dependent upon User, Application, and Database. User is also dependent on Database, and Application is dependent on GUI. Some of the original associations are still there, such as Interface to each type of database. The CommandLineApp class of the new GUI package shows an association with Interface of the Application package. The constants file doesn't extend or instantiate any others, so it can be viewed without association/dependencies. The RE diagram maintains the subclass relationships between the different types of household items as well. The original diagram, however, does have the arrows going in the wrong direction to represent the subclasses. The RE diagram revealed multiple dependencies and associations that weren't originally planned for.

Lessons Learned

Our biggest takeaway from this project was the realization of the cruciality of a solid design phase. The design phase forces the software engineer to fully conceptualize and reason about an architectural system. A well executed design phase will lead to smooth implementation. Another lesson we learned is that descriptive and prescriptive architecture will not always match. During the design phase, it is often not possible to think of and plan for every possible situation or edge case in a software system. Because of this, changes will have to be made or new features implemented which skews the consistency of the prescriptive and descriptive architectures. The design portion was slightly more difficult due to the amount of planning. The design process starts from a more abstract mindset, so it can be difficult to pinpoint where to start or how to design the system. When writing code, it is easy to get modules finished due to the ability to immediately see everything work together. Writing the code was also made simpler by having the design specifications already laid out.

The RE plugin can be used to compare the final deliverable to the original specifications. This can help ensure that the customer is getting exactly what he wanted. The plugin can also be beneficial to improve the documentation and ensure that everything is properly documented for future maintainers and users. The RE diagrams showed that the implementation was closely based on the design diagrams. A few classes were added for clean functionality (the constants and database classes), but overall the descriptive architecture was closely based on the prescriptive architecture.