Darren Huynh
CSCI 233

Final Report 5/25/2016

Over the course of the semester, I have been working on an Inventory management program. The program was required to use SQLite (database), TkInter (GUI), and was to be written for Python 2.7. This program was designed to allow users to create, edit, view data entries via an user friendly interface. My initial goals of the project were to ensure data normalization by catching exceptions/errors before communicating with SQLite database, proper communication and setup with the SQLite database, and complete development of the GUI menus and widgets.

With Python, I was able to fully implement the data manipulation functions that I wanted. Users are able to create, edit, and view entries with the error handling mostly complete. Instances that would cause errors are caught, and communicated to the user. For example, creating an entry for a computer with the same serial number notifies the user that the computer already exists, and does not modify the database. I also included a changelog, which records activity, activity type, and time of activity into a text file that is automatically created if the directory doesn't exist. As an additional feature, I added in an Admin login/password for deleting entries, so that we could protect data integrity by disallowing arbitrary deletions.

Within TkInter, the button widgets are fully functional, and majority of the frame/label/entry widgets are positioned appropriately for easy user navigation and usage. Many of the dialog boxes are created via TkInter, to communicate errors and confirmations to the user. Utilization of the pack and grid methods made the GUI more user friendly, and easy to navigate.

SQLite3 was probably the least used out of the three required technical components. Creation of the table was easily handled by Python, and all the SQL commands were bundled in Python functions. The table was set up with good data normalization practices such as multiple columns, a primary key etc.

Some difficulties that I expected to face were (initially) data normalization/exception handling, and that creating some of the functions would be somewhat difficult. However, some of the actual difficulties I faced were learning grid, getting column data from SQLite table, and catching extraordinary exceptions. Initially, pack was the only method that I knew of, and learning grid proved to be a bit of a task at first, due to a long time of not practicing Python. Additionally, getting column data from SQLite tables was incredibly frustrating, due to the way that the method of fetchall() stores its data into a list, forcing me to basically parse its content every time I wanted the results of an SQL command. Finally catching weird exceptions was difficult in the sense that I did not account for such exceptions, such as my summary page loading statically (changes not represented until reload of program), and the view entry page accepting multiple correct parameters, but only displaying one page. Despite these difficulties, I was able to overcome them and produce a more functional, and stable product. I am happy to say that I was able to meet my goals.

Lessons I learned while working on this project was that Grid was (mostly) superior to Pack for more interactive programs. Pack is incredibly useful when you do not need to micromanage your widgets, but Grid allows you to place every widget exactly where you need it, and allows you to manipulate size. Grid was vital in my project for this reason. Another lesson was that the preloading framework was insufficient for the summary page that I wanted to create. Originally, the summary page did not reflect changes until you restarted the program, because the “summary page” is preloaded upon start. To work around this, I implemented the

TopLevel method, and dynamically created the summary page. The summary page now reflects changes as soon as you access the page. Finally, I learned that error handling is easier to write if you plan initially. I didn't have to backtrack very often since I wrote most of my error catching as I wrote functions. This was very valuable in saving time, and making sure things worked before I moved on.

An entire different version/expansion of my project would scrap the multiple pages setup, and create an "all in one" tabbed version of the program. By reducing the amount of pages needed, we can have a cleaner appearance and implement more intuitive functions, such as clicking an entry from a list box, and choosing to edit that entry alone. This allows the user to save time.

Some possible modifications to my project is the usage of objects. Object manipulation and functions would be incredibly useful in generating more detailed information about computers, and not just simple identification details. By using objects, we would probably also be able to reduce some of the repetitive code. This also allows for management of a ticketing function (if pc has an error or malfunction). Similarly , the GUI management may be worked on, as some of the widgets are not 100% placed properly, and the resizing of windows is not implemented. Scrollbars would also be super useful. Lastly, including more options to sort the summary page would increase the utility of the summary page for the user dramatically. One of the only regrets that I have about my project is that some of the code is incredibly repetitive, and could probably be condensed into a function. This is probably doable.

In conclusion, I learned a lot about GUI programming, and have come to appreciate database management's strengths and weaknesses. I think that I worked hard on something, with a great result. I enjoyed working on this project as it holds personal relevance; I would like to be able to develop such end-user programs for professional and personal use. Learning to

catch errors and work with SQL databases increases my interest in such elements of Computer Science, and I hope that this will prove useful in the future.