Analysis of the jHotel software.

1. Met requirements

1.1. Completely met requirements

1.1.2. The user is able to mark a reservation as checked in (Req_04)

1.1.3. The user is able to give a special price to a booking (Req_06)

1.2. Mostly but not completely met requirements

2.1.1. The software can store customers, and information on them. However, it does not keep track of a customers power/loyalty level (Req_03)

2.1.2. The software can delete a booking, but it can’t mark it as cancelled and at the same time retain information on what the customer is to pay/ has paid (Req_07)

2. Unmet requirements

2.1. There is no way to add more than one room to a booking (Req_02)
2.2. The system cannot differentiate between two different hotels (Req_01)
2.3. The system does not keep track of any room data, specifically:

2.3.1. The rooms do not have a quality attribute (Req_01)
2.3.2. The rooms do not have information on neighbouring rooms (Req_01)

2.4. The software does not meet the speed requirements:

2.4.1. Check in management takes 10-20 seconds to start up (NonFunc_02)
2.4.2. Room selection takes 10-20 seconds to start up (NonFunc_01)

2.5. The system does not handle checking out (Req_05)

3. Problems with the software

3.1. There is no separation of the classes into packages/modules
3.2. There is an unclear MVC separation or similar division
3.3. Dependencies hare handled very strangely, and are often reversed (e.g. a *ReservationManager* (which is a window) is passed into the constructor of a *Reservation* to add that *Reservation* to the *ReservationManager*)
3.4. No clear OOP - customers and rooms are represented by String-arrays inside ArrayLists

3.4.1. The customer in a booking is not a reference to a customer in the database, but only a copy of *some* of the fields of a customer

3.4.1.1. Clients created through the booking window is not added to the clients list

3.5. Confusing naming convention - “guest.java” is not a guest object but a DAO for guests
3.6. Old and outdated graphics library (Swing)

3.6.1. Uses deprecated methods within Swing
3.6.2. Some components does not work at all (e.g. date picker)

3.7. Bad code quality

3.7.1. Sometimes there are 400 rows that could have been condensed into a loop of maybe 10 rows

3.7.2. Some if-statements are superfluous because they depend on a single variable which then is the only difference between the branches

3.7.3. Variable names are hilariously bad (e.g. “jButton47”, or “g”)

3.7.4. Very few inline comments, very few things are explained.

3.7.5. Javadoc comments give no worthwhile information about the methods

3.7.6. Some javadoc comments are situated over the wrong methods

3.8. No error messages if a form is not filled in completely

3.8.1. Sometimes generates null pointer exceptions (but does not crash) if a form is not filled in correctly

4. Reflection/rationale

The jHotel software is a hotel front desk software written in Java. It keeps records of customers, the rooms of the hotel and which floor they’re on, and any reservations made. On a first glance, this is a good candidate for the Linneaus Hotel, as it fits the problem domain and provides a pretty extensive base to build on. Some modifications would be necessary but without looking at the code, and only judging as an end user, the software is nearly there.

We considered refactoring, but it quickly became clear that it would be a severe re-architecture. Lots of things would have to be changed, since everything was a tangled mess to begin with.

The database would need to be modified extensively, since the properties of rooms depended on the array they were stored in, and as such they could only have one (their size). It’s functionality did in no way warrant it’s complexity, and in adding information on what view the rooms had, what hotel they belonged to, and what rooms were connected, would no doubt only increase it further. It was also incredibly slow to load and save, since every time it read all data from disk, even if it only needed one entry, but didn’t keep things in memory for very long. We quickly opted to replace the database.

The GUI would also need to be remade in some parts, because it relied on an outdated library, and within that made use of archaic and obsolete (to the point of not compiling) functionality. All design was generated programmatically, with an impenetrable naming scheme, and an unclear separation from the controller and model functionality.

At the very least, we would have to learn an old and otherwise irrelevant GUI framework. If we wanted to switch to a more modern one, we would also have to lift the various controller functionality from the old view classes, and rewrite them to work with the newer framework. Since an application like this is driven by the controller serving as a backbone, this would be a deep change.

To summarize, getting the software up to speed in a reasonable way would mean rewriting the view, rewriting the controller, and rewriting the model. This would still be done within the old jHotel codebase, meaning any old idiosyncrasies be kept if they weren't in the way, worked into the new system.

At this point, it felt more merciful to do a complete rewrite, especially looking at the overall quality of the old system. This would allow us to separate classes into different packages from the beginning, use a modern GUI framework (thereby removing the window layout from the code), use a relational database for data storage (speeding things up, making it more human readable, and easier to add attributes to in the future), and divide the application into two, a client and a server (to better manage the program being run in two different hotels at once).

A rewrite would also mean that we could leave out functionality in the jHotel software that we felt were irrelevant for us, e.g. the handling of different floors, and translations of the program.