Post-Mortem Report TOC

1. Project

A. Description

i. Project Name: Booking Site
ii. Client: Homy
iii. Project Manager: Jasmine Ellis
iv. Solutions Architect: Thomas Higgins
v. Start Date: 03/03/17
vi. Completion Date: 29/05/17

B. Project Overview

The assigned project was a simple appointment booking system that can be used by multiple businesses. The application would allow customers to register, login, making bookings with any business registered in the system, and cancel bookings. Businesses can also make and cancel bookings, as well as other business management functions such as managing employees, appointment types, and opening hours.

i. What was the project success criterion?

   Fulfilment of the specifications given by S.E.P.T Major assignment parts A, B and C within their given timeframes (refer to said documents for details of said specifications).

2. Performance

A. Key Accomplishments

i. What worked well?

   We worked well as a team to divide the workload and collaborate on the project. Some members of the team focused on gui implementation, whilst others focused on testing and building the app, but we all worked together to complete the project. Communication and assignment of tasks was well-organised via Slack, Trello, and regular meetings.

ii. Usage of git to maintain versions, and to prevent conflicting development on the same aspects of the app.

   Our team collaborated via Github. We utilised branches when developing new features and would make frequent commits to avoid merge conflicts. When conflicts did occur, we worked together to resolve them and figure out what's needed and what can be overwritten.

B. Key Problem Areas

i. What went wrong?

   Having to refactor the program from a regular java app, to a maven project, coupled with the lack of knowledge of Scenebuilder, resulted in some difficulties when migrating the project and having to learn new technologies in such a short span of time. We also experienced problems with Git, in which old bugs would be re-introduced into the master branch of the git repo. Another issue we experienced was having to learn various processes on-the-fly, such as unit testing. If we had had more time to focus on unit testing, and software testing in general, we would have been able to iteratively write tests as/before we developed. As much as we tried to focus on test driven development, some times we were lax in writing tests, and they went by the wayside to an extent. Lastly lack of general Java development knowledge hindered our progress to a certain extent.

ii. What project processes didn't work well?

   Our git process could have used improvement. We experienced many problems with the re-introduction of bugs. This may have been avoided via consistent use of branches and smaller, more frequent commits.

iii. What specific processes caused problems?

   To be more specific, our git usage was somewhat haphazard. Smaller, less monolithic, & more frequent commits were definitely an issue we could work to improve on. But this was compounded via the lack of branching, that - at times when we were rushed - resulted in important commits being overwritten, due to the nature of how git determines changes over commits. Of course while accidental, these mistakes would have been avoided via branching, pull requests, and code reviews/QA prior to merging.

iv. What were the effects of key problems areas

   These challenges resulted in a delay in development, as well as spending more than ample time on features which would have been made much easier using different methods. The team managed to overcome these challenges and create the project deliverables and work according to the processes.

v. Technical challenges

   Screen resolution:

      Many of the machines that were used in the development of

the application featured vastly different resolutions, which often resulted in a number of display issues.
In future projects it would be our recommendation to develop for, and on machines with, an agreed upon predetermined screen resolution.

Maven configuration:

The use of Maven for this project lead to a number of issues, particularly its interaction with the project dependencies.
The conversion of the project to a Maven project was fairly lengthy requiring a number of changes to configuration…
Despite working after the time of conversion to a Maven project, the final build was not able to run using Maven commands, as the SQLite functionality was not operational.
Most, if not all of the issues resulting from Maven could have easily been avoided had the project been started as a Maven project. This would have made conversion unnecessary, as well as giving the development team time to familiarise themselves with a tool they had yet to be exposed to.

Scene builder:

Building the JavaFX views from scratch was perhaps the most time consuming process throughout the project.
Having been forbidden from using such a tool on previous projects and without having been told otherwise, the project was developed with scratch built JavaFX objects composing the views.
The team was later informed that not only was the use of Scenebuilder allowed, but highly-encouraged, however it was too late to smoothly transition by that point in the project.
Being informed of the importance and ability to use Scene Builder earlier, would have resulted in the project being developed faster and more smoothly.

C. Risk Management

i. Project risks that have been mitigated:
   - Team members experiencing events outside of the project that affect their level of commitment, e.g. death of family member. Mitigated by consulting with Homy, who modified our schedule to allow for more time to complete certain tasks.

ii. Outstanding project risks that need to be managed:
   - Corruption of source code. Problems related to bugs being reintroduced via Git are ongoing.

D. Overall Project Assessment

8.5/10

3. Key Lessons Learned

A. Lessons Learned

Throughout this project we learned about several new processes including Kanban boarding via Trello, proper Git flow, how to hold various types of Scrum meetings, implementing design patterns, and acceptance testing via Lean Testing. Areas of improvement include how meetings were conducted, e.g. meetings could have been more in-line with Scrum, rather than simply being Scrum-like. As mentioned previously, our Git flow could have been improved upon via consistent use of branches and smaller, more frequent commits. We also could have improved the testing process via more disciplined test driven development. Keeping acceptance tests up to date with the progression of the project and doing regular test runs would have also helped us identify bugs or technical issues in a more timely manner. Another key lesson learned is the need for organised and consistent bug reporting, e.g. keeping our bug reports in one place and assigning bugs to individuals.

B. Post Project Tasks/Future Considerations

i. Ongoing development and maintenance considerations

   Booking generation:
   The program generates bookings on start-up based on how long it has been since the bookings were last generated.
   In the event that the program remains unused, particularly if there are a number of employees and businesses, then the program could take a very long time to start-up.
   In the future and when deployed, it may be wise to have the process occur daily at a preset time, so as to cut down this startup time.

ii. Is there anything still outstanding or that will take time to realize? (i.e. in some instances the full project deliverables will not be realized immediately)

   Given additional time we would have liked to have reconfigured a number of functions so as to require less frequent database queries, as well as cut down on processing time.

   Ideally we would like to implement a post-deployment logging system in the program. so as to keep track of events and errors that occur while the application is live.

   Some unit tests were written before the project was refactored to make use of maven, and these would need to be written as per this new design/file structure.

The abstraction of certain functions into their own classes, such as validating user entries and updating their respective views to reflect this. As has been already mentioned, if we had been instructed on the use of maven and/or scenebuilder much ahead of when we did, we would have built the project in a vastly different way.

Performance against project goals/objectives	1 2 3 4 5 6 7 [✓ 8] 9 10
Performance against planned schedule	1 2 3 4 5 6 7 [✓ 8] 9 10
Performance against quality goals	1 2 3 4 5 6 [✓ 7] 8 9 10
Performance against planned budget	N/A
Adherence to scope	1 2 3 4 5 6 7 8 9 [✓ 10]
Project planning	1 2 3 4 5 6 7 [✓ 8] 9 10
Resource management	1 2 3 4 5 6 7 [✓ 8] 9 10
Project management	1 2 3 4 5 6 [✓ 7] 8 9 10
Development	1 2 3 4 5 6 7 8 [✓ 9] 10
Communication	1 2 3 4 5 6 7 8 9 [✓ 10]
Team cooperation	1 2 3 4 5 6 7 8 9 [✓ 10]
Project deliverable(s)	1 2 3 4 5 6 7 8 [✓ 9] 10