Software Engineering Group Project

End of Project Report

|  |  |
| --- | --- |
| Author: | David |
| Config Ref: | SE\_05\_DEL\_06 |
| Date: | 2016-13-02 |
| Version: | 1.0 |
| Status: | Release |

Department of Computer Science

Aberystwyth University

Aberystwyth

Ceredigion

SY23 3DB

Copyright © Aberystwyth University 2015

CONTENTS

CONTENTS 2

1. Introduction 3

1.1 Purpose of this Document 3

1.2 Scope 3

1.3 Objectives 3

2. Management summary 3

3. Historical Account 3

4. Final State Of The Project 5

5. Location Of Group Repository 6

6. Performance Of Each Member 6

7. Critical Evaluation 6

REFERENCES 6

DOCUMENT HISTORY 7

# Introduction

## Purpose of this Document

This document contains the teams learning outcomes, critical feedback and performance reviews. It additionally specifies improvements for future projects.

## Scope

This report describes the final state of the project and outcomes following the completion of the project.

## Objectives

The reader of this document should understand new mitigation techniques for the difficulties encountered during the completion of the project. It should additionally provide new processes for software developments in large projects discovered by the group.

# Management summary

We set out to create a task assignment system with three components, TaskerCLI, TaskerMAN and TaskerSRV. These components communicate to each other via TaskerSRV and allow for task assignment and synchronisation for users. The client provided a description of each component and their functional requirements to ensure each component meets their expectations.

All members of the team were open and friendly from the beginning. This meant everyone quickly learnt individual strengths and weaknesses which allowed us to split into two separate teams; web and client development. Every member of the team provided strong contributions to the project and was happy to receive constructive criticism and incorporate changes leading to a strong feeling of comradery.

The biggest difficulty continuously encountered throughout this project was lack of experience or exposure to elements used within this project. This included building GUIs, using JDBC in Java, displaying modal windows and handling a variable number of elements in the management component. Whilst prototyping provided some initial experience and subsequent design changes during full implementation bugs arose due to the team not fully understanding the technologies. This led to subsequent refactoring of earlier code as more principles were understood.

Within TaskerCLI race conditions and threading lead to unexpected and difficult to reproduce bugs. This ranged from the database disconnecting randomly to all windows of the program becoming invisible. The client development team developed a stronger understanding of how to use the debugger to reproduce these race conditions and prevent them.

The team developing TaskerMAN was struggling to post data to modal windows. Initially they created several prototypes using AJAX however they later realised that through the use of PHP GET and POST methods they could easily push data to the page. This had the additional benefit of sanitising data when combined with prepared statements.

At the conclusion of this project out of the twelve functional requirements we completed and demonstrated all but two functional requirements. The two functional requirements not completed are FR7, filtering and sorting of tasks in TaskerMAN and FR9, task synchronisation in TaskerCLI.

FR7 was not implemented as the requirement was misread as being optional. FR9 is implemented however two bugs prevent the correct operation of saving elements and updating a task on reconnection.

All documentation is currently in a good state as they contain the latest information on the project and meet QA standards.

# Historical Account

Once the initial functional requirements had been provided for the Tasker system, the group set about deciding upon the platform that the Tasker system would be built on and tested against. They then produced a high level design which included user interface mock ups and UML diagrams to “map out” the overall functionality. This high level design was shown to the client to ensure that the group effort was being applied in the correct way. From that meeting with the client the group learnt that our understanding of Task Elements was incorrect; this delayed development until the group figured out the best way to implement the changes in the user interface design, and the design of TaskerCLI, TaskerMAN, and TaskerSRV.

Once initial designs had been agreed upon, the group was separated into two sub-groups to separately work on development of TaskerCLI and TaskerMAN. This was done to allow the group member’s skills to be applied more effectively in their strongest area.

## TaskerCLI

The first task for the team handling TaskerCLI was to begin work on the Design Specification whilst the Test Specification was being developed.

The Design Specification contains some contents from the high level design document, which had already been completed. However, due to changes that were recognised during the production of the high level design documentation, these diagrams could not be copied over without modification. We decided that the team working on TaskerCLI would not contribute heavily to the Test Specification and only work on the Design Specification for TaskerCLI. This was done to ensure that all changes could be made before the Design Specification deadline arrived.

This change to how Task Elements were handled, as pairs of descriptions and comments as opposed to a single element, required changes to the user interface design as well as a lot of the logic of loading and storing data. Interfaces and Class Diagrams also had be developed in accordance with this new way of handling Task Elements.

After the Design Specification had been submitted, work began on building the prototype of TaskerCLI for demonstration to the client. The major difficulty in developing this prototype was that TaskerSRV was not functional during development, meaning the desired functionality (downloading Task data) could not be implemented in time for the prototype demonstration. Fortunately, however, TaskerCLI had the ability to work in an “offline mode” so the basic functionality could be displayed.

Development of the final software was a generally straightforward task, however there were a series of bugs found during testing that extended the development time. The most difficult of the errors to find and fix were race conditions that arose when working with TaskerSRV. These errors were arduous and took many hours to fix, meaning that some team members were required to work extra hours in order to complete the development of the software.

## TaskerMAN

The first task for the php team was the design specification in joint progress with the TaskerCLI team whom were also working on the design specification.  
The creation of the design specification initially when on without much of an issue, however there were timetabling conflicts which resulted in TaskerMan team meetings without  all members present .

After the submission of the test specification, the feedback introduced a curveball which affected the design stage, as we were unaware of the complexity behind task elements. We believed that Task elements was nothing more than a long description of the task.

During the prototyping stage of the project The entire team ran into issues due to TaskerSRV not having been functional which held back all other development. Thankfully TaskerMan was somewhat functional as we had modal windows and login working correctly.  
When it came to actually implementing the PHP the hardest issue to deal with was finding a creative solution for Task elements which was eventually tackled but not without its struggles.

Aside from this implementation was relatively calm without any major hiccups or setbacks. Bugs were identified and patched as coding week progressed until a fully operational website was up. All functional requirements were met to a decent level, aside from the filtering of tasks via End date and Status. This was a functional requirement that was unfortunately completely forgotten during implementation

## TaskerSRV

TaskerSRV was planned with the functional requirements in mind to contain three tables: Users, Tasks and Elements. Whilst the users and tasks table design did not change throughout the project a misunderstanding about foreign keys led to the elements table being redesigned. Previously the elements table contained two foreign keys and a string to allow elements to map to a user and task. After discussing with the client the team realised we needed to store a comment and the current design would not work.

This lead to us redesigning the elements table to hold a primary index, string comment and element description and a foreign key to the relevant task.

The next step was creating this schema from the design, for this task we used MYSQL workbench which allowed us to graphically prototype the database. After tweaking some minor mistakes such as the length of a column being too short we verified TaskerSRV’s operation.

Using the queries generated from MYSQL we created a BASH script to generate a fresh database. This script can be run interactively and from a command prompt and will detect an existing installation, or partial installation and generate clean tables with the user's permission. The final step was taking test data produced and using the bash script to automatically insert this data when instructed to create test data by the user.

# Final State Of The Project

The project is viewed as meeting all but two functional requirements at its completion.

TaskerSRV correctly implements all functional requirements; this includes storing the following information within its schema:

* Full name and Email Address for Users
* Task Title, Start Date, End Date, Status and Allocation
* Task Element with free text comments

TaskerMAN meets all but one functional requirements; these are:

* Adding, Updating and Deleting team members
* Adding, Updating, Viewing and Deleting task data
* Reallocation of tasks
* Identification of managers

This includes input validation and additional checking such as enforcing a member cannot have any tasks before being deleted. Filtering and sorting by column whilst viewing tasks specified in FR7 has not been implemented so FR7 is not completely met.

TaskerCLI implements all but two functional requirements:

* Connecting to database and storage of connection details
* Allowing users to login and local storage of this list
* Local copies of tasks periodically updated from TaskerSRV (every minute currently)
* Intuitive GUI (IR1)

Due to a bug in saving task elements an update comment will not synchronise back to the database. This same code path also is responsible for uploading changes made offline as soon as a connection is available thus offline changes are discarded. Despite this changing a Task’s status still synchronises correctly whilst online.

Secondly whilst offline the database will automatically attempt to reconnect on a timer, this leads to multiple dialog boxes being displayed every time a connection is attempted if the application was previously online. This can make the program difficult to use due to dialog boxes appearing on every change or periodically.

# Location Of Group Repository

The group repository is located at:

<https://github.com/antibones/CS221_Group_05>

The final branch for delivery is the “master” branch.

# Performance Of Each Member

# Critical Evaluation

The team performed very well during this project even during setbacks faced throughout the project. All members clearly knew their areas of expertise and weakness allowing the team to best utilise individual skills. An area for improvement is during quieter weeks additional training could have been performed ensuring skill sets are more equally spread across team members. During this project members were present throughout however if somebody with a unique essential skillset went ill it would have caused heavy delays.

The project could have been improved in two major ways. Firstly the final product did not have any unit tests built for it. Initially we were planning on using test driven development; however our lack of experience with the technology meant whilst tests would pass the program would not function correctly.

Following this the team agreed additional prototyping would have paid dividends. Our initial prototypes proved concepts such as using window builder to create GUIs but complete additional steps such as refreshing the displayed data. This lead to large deviations from the original plan which assumed certain functionalities that could have been avoided with more rigorous prototyping.

From this extra prototyping new unit tests should be built, a common issue during implementation was regressions which went largely undetected from the lack of unit tests. Whilst unit tests would have taken additional time to produce, the time saved would have paid dividends.

The team developed an understanding of the need for a well-developed project plan. By completing research and identifying risks in new technology additional testing and training can take place. This prevents large design changes, unnecessary refactoring and bug introduction. In larger projects with multiple teams these changes would significantly impact on other teams who may depend on the old incorrect design.

Finally whilst within more complex methods it is worth breaking them into sequence diagrams. Whilst the implementation may seem trivial and obvious to all people everyone will still approach that algorithm differently. This leads to people assuming the algorithms behaves in a certain manner as it is not documented and leads to subsequent bugs stemming from those assumptions.

REFERENCES

[1]

DOCUMENT HISTORY

| *Version* | *CCF No.* | *Date* | *Changes made to document* | *Changed by* |
| --- | --- | --- | --- | --- |
| 1.0 | N/A | 13/02/2016 | Original Version | DAF5 |