Capytec Task Allocation System

Sandra Czernik W19009505  
Faculty of Engineering and Environment  
*Northumbria University* Newcastle, United Kingdom  
[sandra.czernik@northumbria.ac.uk](mailto:sandra.czernik@northumbria.ac.uk)

Samuel Farnworth W19003324  
Faculty of Engineering and Environment  
*Northumbria University* Newcastle, United Kingdom  
[sam.w.j.farnworth@northumbria.ac.uk](mailto:sam.w.j.farnworth@northumbria.ac.uk)

Kieran Robinson W19002952  
Faculty of Engineering and Environment  
*Northumbria University* Newcastle, United Kingdom  
[kieran.robinson@northumbria.ac.uk](mailto:kieran.robinson@northumbria.ac.uk)

Alex Heaton W19006404  
Faculty of Engineering and Environment  
*Northumbria University* Newcastle, United Kingdom  
[alex.heaton@northumbria.ac.uk](mailto:alex.heaton@northumbria.ac.uk)

Word Count: 14216

Table of Contents

[Introduction 3](#_Toc72795755)

[Mission 1 4](#_Toc72795756)

[Team Code of Conduct 4](#_Toc72795757)

[Project Proposal 9](#_Toc72795758)

[Skills audit of the team 10](#_Toc72795759)

[Skills audit - Alex Heaton 10](#_Toc72795760)

[Skills audit – Sandra Czernik 13](#_Toc72795761)

[Skills audit – Kieran Robinson 16](#_Toc72795762)

[Skills audit – Sam Farnworth 19](#_Toc72795763)

[Quality document 22](#_Toc72795764)

[Risk Engineering 25](#_Toc72795765)

[Risk Analysis 25](#_Toc72795766)

[Risk evaluation 25](#_Toc72795767)

[Risk Manager 26](#_Toc72795768)

[Risk Planning, Risk Control, Risk Monitoring and Risk Staffing 27](#_Toc72795769)

[Work breakdown, task prioritisation and task allocation 28](#_Toc72795770)

[Work breakdown structure 28](#_Toc72795771)

[Programme evaluation and review technique 28](#_Toc72795772)

[MoSCoW 30](#_Toc72795773)

[Scheduling, Version Control and Revisions 33](#_Toc72795774)

[Scheduling 33](#_Toc72795775)

[Version Control 34](#_Toc72795776)

[Revisions and Methodology 35](#_Toc72795777)

[Logging progress, problems, and solutions 35](#_Toc72795778)

[Communication 35](#_Toc72795779)

[Logging of all Progress 35](#_Toc72795780)

[Meetings Documented 35](#_Toc72795781)

[Mission 2 43](#_Toc72795782)

[Requirements Analysis 43](#_Toc72795783)

[A statement of the purpose of the system and how it meets the client’s needs. 43](#_Toc72795784)

[Requirements modelling - UML Use Case Diagram 43](#_Toc72795785)

[Requirements modelling - UML Use Case Descriptions 44](#_Toc72795786)

[Mission 3 47](#_Toc72795787)

[Software Analysis 47](#_Toc72795788)

[Class diagram 47](#_Toc72795789)

[Sequence diagrams 48](#_Toc72795790)

[Activity Diagram 51](#_Toc72795791)

[Mission 4 52](#_Toc72795792)

[Testing 52](#_Toc72795793)

[Bug tracking system 53](#_Toc72795794)

[Process for testing 53](#_Toc72795795)

[Full record of test process in use 54](#_Toc72795796)

[Unit Testing 80](#_Toc72795797)

[Bug Reports 90](#_Toc72795798)

[Mission 6 93](#_Toc72795799)

[*Entity relationship diagram* 93](#_Toc72795800)

# 

# Introduction

The following report includes documentation created during the development process for Capytec plc. A prototype was created with the main four features, maintaining a list of tasks, supporting a team of caretakers by allowing them to allocate daily tasks amongst themselves, maintain a record of when certain tasks have been completed, and provide various reports for managers, on the task completed and allocation.

## Mission 1

### Team Code of Conduct

A code of conduct has been created to set guidelines for team members to follow during the project. Team members were expected to exercise their own judgement to meet the requirements of the code and seek help if in doubt. The code of conduct displays professional standards required by all four members within the group. It outlines areas such as communication, roles and responsibilities, meetings, attendance, absence, and conflict resolution. Any time the Code of Conduct is edited in any way, it will need to be approved by all members and all version changes will be documented as well. As seen in Figure 5, there is a table for version control and any changes mentioned above so all team members are aware of all changes made by anyone in the team.

Text, letter, email

Description automatically generated

Figure 1 Code of Conduct page 1.

Table

Description automatically generated with medium confidence

Figure 2 Code of Conduct page 2.

Text

Description automatically generated

Figure 3 Code of Conduct page 3.

Text

Description automatically generated

Figure 4 Code of Conduct page 4.

Table

Description automatically generated with medium confidence

Figure 5 Code of Conduct page 5.

### Project Proposal

The project proposal for the following project has already been provided and can be seen below:

Capytec task allocation system

Capytec plc is a company that provides a range of business and light industrial services. They are based at a large site just north of Nottingham. The site has a number of buildings of various sizes. The site is maintained by a team of six caretakers. During a working day, they carry out various maintenance and cleaning tasks. Some of these tasks are routine (e.g. emptying wastebaskets). Some of these are one-off tasks (e.g. replacing a broken window).

The current system is paper-based. Each day, administrators in the Property Services department provide a list of one-off tasks that need completing. At the start of each day, the caretakers read the list of one-off tasks that was produced the previous day. The caretakers also maintain their own list of routine tasks. The caretakers distribute tasks among themselves. They cross off any one-off tasks that have been allocated. If there are too many tasks to complete in one day, unallocated tasks are left for the next day. Usually, there are not many one-off tasks, so the caretakers also select some routine tasks to do that day.

The caretakers are experienced enough to be able to select appropriate tasks. For example, they can see for themselves whether the windows need cleaning and have a good idea of when the waste bins will need emptying.

The company want to move to an electronic system for maintaining the list of tasks and distributing them amongst the caretakers.

Your team has been asked to produce a working prototype that offers four main features:

1. Maintain a list of tasks.
2. Support a team of caretakers in allocating daily tasks amongst themselves.
3. Maintain a record of when certain tasks have been completed and who completed them.
4. Provide various reports on task completion and allocation to interested parties.

A table was created with missions that each team member has been allocated to, according to the skills audit conducted, which can be seen in the next section below, as seen in Figure 6.

Table

Description automatically generated

Figure 6 Project proposal - assigning missions.

### Skills audit of the team

A skills audit has been conducted for each team member to self-asses non-technical and technical skills, which aided in deciding which missions each team member will complete and will also be a valuable tool in the reflection assignment, where reasons will be given as to why certain team members were chosen to do specific missions. The skills audit also helped all team members with personal developments, to see which skills were developed at the end of the project.

#### Skills audit - Alex Heaton

Table

Description automatically generated

Figure 7 Skills audit - A.H. Page 1.

Table

Description automatically generated

Figure 8 Skills audit - A.H. Page 2.

Table

Description automatically generated

Figure 9 Skills audit - A.H. Page 3.

#### Skills audit – Sandra Czernik

Table

Description automatically generated

Figure 10 Skills audit - S.C. Page 1.

Table

Description automatically generated

Figure 11 Skills audit - S.C. Page 2.

Table

Description automatically generated

Figure 12 Skills audit - S.C. Page 3.

#### Skills audit – Kieran Robinson

Table

Description automatically generated

Figure 13 Skills audit - K.R. Page 1.

Table

Description automatically generated

Figure 14 Skills audit - K.R. Page 2.

Table

Description automatically generated

Figure 15 Skills audit - K.R. Page 3.

#### Skills audit – Sam Farnworth

Table

Description automatically generated

Figure 16 Skills audit - S.F. Page 1.

Table

Description automatically generated

Figure 17 Skills audit - S.F. Page 2.

Table

Description automatically generated

Figure 18 Skills audit - S.F. Page 3.

### Quality document

A quality document was produced to outline standards for formatting and layout for any documents created by team members, for example, the Code of Conduct. This makes every document consistent and makes the whole project look more professional.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 19 Quality Doc. Page 1.

Text

Description automatically generated with medium confidence

Figure 20 Quality Doc. Page 2.

Text, letter

Description automatically generated

Figure 21 Quality Doc. Page 3.

### Risk Engineering

#### Risk Analysis

Diagram

Description automatically generatedA risk analysis has been conducted to identify risks that can be caused by unforeseen or unplanned events occurring in the team, or assumptions made during the planning process which may cause the project to be delayed. The goal of the risk analysis was to first identify any concerns and to identify their risk and any risk owners. Then, the risk was evaluated based on the likelihood of it occurring (improbable, possible, probable), and the consequences (acceptable, tolerable, undesirable, and intolerable), which resulted in a risk level from low to extreme, and a risk value from 1-12, 12 being the most extreme. The post-mitigation column showed a description of how a specific risk can be managed or decreased, the following columns show the risk severity, likelihood, and level if the mitigations and warnings are being followed by the risk owners, as seen in Figure 23. The following 15 risks provided in the spreadsheet have been anticipated by all team members from common sense, past experiences, and creative thinking.

Figure 22 Risk analysis section.

Table

Description automatically generated

Figure 23 Risk Matrix, High-resolution image available at: <http://unn-w19009505.newnumyspace.co.uk/SEP/RISK.PNG>.

The following table in Figure 24, has been used in the risk analysis to calculate risk levels and values. For example, if a risk is improbable, but is intolerable, the risk level would be high, with a value of 10. This risk ‘level’ is also known as risk exposure.

#### Risk evaluation

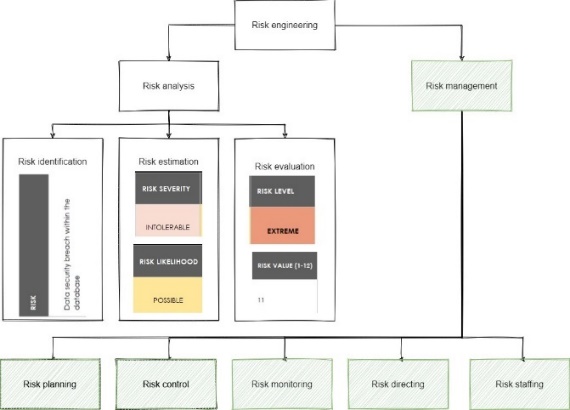
Some risks have been treated together as a single risk, as they are dependent on each other. For example, a risk with the ID 2, includes PC failure while working on code, or other important documents which can result in data loss. This task could have been split into two tasks but data loss depends on a PC failure or corruption, so it has been added as one risk.

#### Risk Manager

The project manager for the group project was also assigned as the risk manager. A risk manager is responsible for defining all risks, structuring the risk spreadsheet provided in Figure 20, and implementing an approach that will aid all team members with minimizing any risks, which includes risk mitigation and remedies for if a risk occurs. The risk manager is also responsible for presenting a summary of all risks to all team members and any further reviews so they can all be agreed upon and all team members are aware of any potential risks within the project.

Figure 24 Risk analysis table

According to Levenson, 1991, ‘risk is a function of three things’

* the likelihood of a hazard occurring (h)
* the likelihood that the hazard will lead to an accident (a)
* the worst possible potential loss associated with that accident (l)

This creates the following formula: r = P(h) \* P(a) \* l

The goal of risk management is to reduce at least one of those three elements mentioned above. The risk manager has also ranked the risk values/exposure into four categories, which can be seen below:

Figure 25 Risk management section.

* ≥ 11 **Extreme risk** which requires urgent action
* ≥ 7 < 11 **High risk** which requires action as soon as it is possible
* ≥ 3 < 7 **Medium risk** which may require action in the future
* < 3 **Low risk** does not need extra action

Timeline, treemap chart

Description automatically generatedBased on the values above, an assumption table has been created, where tasks in the top right corner, with the highest severity and likelihood, are located and were tackled first, as seen in Figure 26. In the figure, many of the risks can be seen in the extreme and high-risk category, which were lowered by accessing each risk and creating a post-mitigation risk exposure table.

Figure 26 Risk values/exposure pre-mitigation.

Chart, treemap chart

Description automatically generatedA post mitigation table was also created, to display where the risks would be on the risk value table after the risk owners ensure that the mitigations have been followed, as well as the warnings, which drastically drop the risk value for most risks to low, which can be seen in Figure 27. Desirable results were gathered from evaluating and analysing the risks pre-mitigation and post-mitigation. The most important result from both of the tables was that mitigation and warnings were extremely important as they reduced the risk drastically when they were followed during the project. As seen in Figure 27, most risks after mitigation and warnings can be seen in the bottom left-hand corner on the table, where the severity and likelihood are at their lowest, with no risks seen in the ‘extreme’ category, meaning most risks can be avoided if certain steps are taken to avoid them, provided by the risk manager.

Figure 27 Risk values/exposure post-mitigation.

#### Risk Planning, Risk Control, Risk Monitoring and Risk Staffing

Graphical user interface, text, application

Description automatically generatedA contingency plan was made to see what can be done if a risk occurs, and if recovery can occur when it comes to a natural disaster or data loss, how can it be recovered, and how much can be recovered. A table has been created for the contingency plan, along with the risk ID and a column to describe how the team will know a risk has occurred, and what can be recovered. In some cases, such as team members not being committed to the project and inexperienced team members which may result in delays do not need any sort of data recovery but will need motivation and meetings to discuss things further if those risks ever occur. Risks such as a natural disaster or data breaches do have a recovery plan on the other hand, where backups will need to be restored, for example from GitHub or a local/personal PC.

The project manager, as well as the other team members, were all responsible for risk control to minimize problems and quickly react should any risks occurred. Each risk owner was responsible for their risks and minimizing the chances of it occurring at all times. Risk monitoring was done by the risk manager, and any risks which may have increased or decreased in the risk level were adjusted in the risk matrix spreadsheet. Risks were reviewed daily and team members were asked to review the risk matrix regularly as well in case a risk level needs to be adjusted or a new risk added.

There are many ways to deal with risks, such as;

* Crisis management

Figure 28 Contingency plan. High-resolution image can be found in the Risk Matrix figure at: <http://unn-w19009505.newnumyspace.co.uk/SEP/RISK.PNG>.

* Fix on failure
* Risk mitigation
* Prevention
* Elimination of root causes.

Risk management and prevention have been vouched for as the most advantageous options, by all members, as it allowed for risks to be either completely prevented or mitigated as much as possible before any risk occurred. Crisis management and fix on failure might still have been necessary if mitigation and prevention failed but was not a regular occurrence during the project.

### Work breakdown, task prioritisation and task allocation

#### Work breakdown structure

For project planning to be effective, certain steps had to be followed. A work breakdown structure has been included in Figure 29 to show how all the tasks within the missions have been allocated into 4 different categories (orange) and three different levels (black). Level 1 shows the project title and the final deliverable. Level 2 displays the main project phases, including the requirements stage, design stage, construction stage and testing stage. The lowest level shows which tasks need to be finished to complete each of the project phases above.



Figure 29 Work breakdown structure.

#### Programme evaluation and review technique

The PERT diagram provided in Figure 30 includes similar categories as seen in Figure 29, but with an extra phase at the end called ‘release software’. The following diagram allowed the identification of task sequences, as there is a number in each node, which showed which tasks needed to be completed in which order. The length of the arrows between each task shows the duration of each task, for example as seen in Figure 29, the use case diagram is planned to take less time than all the use case descriptions. The same can also be seen for the data component, as additional data will need to be inserted as the construction phase progresses. Tasks on the critical (middle) path cannot be overrun, which is why it's important to order each phase correctly. These critical paths have also been colour-coded, to provide clarity. The nodes are also known as points in time, whereas activities that need to be completed are on each arrow.

Time management has been taken into consideration, which is why certain ‘phases’ have been split into two, for example in the requirements phase, the code of conduct is to be completed first, but the skills audit can also be started at the same time, but it will most likely be completed later as it needs to be completed by all four team members. Another example is the design phase, where the UML diagram needs to be completed before the class diagram, but then use case descriptions and class diagrams can be started at the same time.

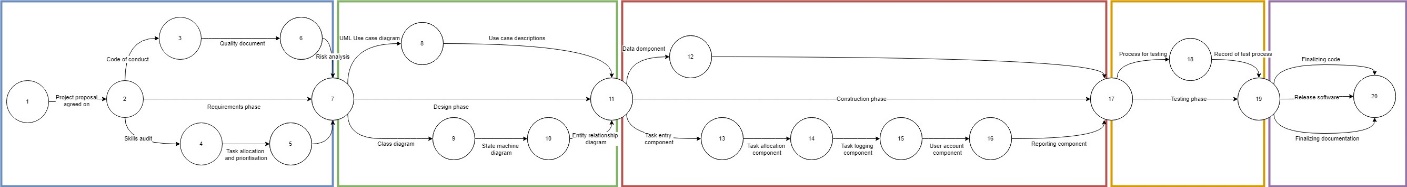


Figure 30 PERT Diagram, high-res image available at: <http://unn-w19009505.newnumyspace.co.uk/SEP/PERT.jpg>.

Diagram

Description automatically generatedThe first phase required the project proposal to be agreed upon, which has already been provided for the group. The code of conduct and the skills audit tasks are on separate paths but can be worked on at the same time, the code of conduct is expected to be completed first and the skills audit second, then the quality document and task allocation can begin at the same time. The risk analysis can be completed at the last stage of the requirements phase.

Diagram

Description automatically generated The second phase has been split into two as well to allow for use case diagrams and class diagrams to be worked on at the same time, as they are two different missions, where one of those was managed by one team member, and the other was a group mission. The use case descriptions were predicted to take quite a considerable amount of time as several were produced. If this PERT diagram was to be redesigned, the only change applicable would be adding the entity-relationship diagram onto a completely different activity arrow, as it could have been created independently without the need for a class diagram.

Figure 31 Phase 1.

Chart, diagram

Description automatically generatedPhase 3 was the construction phase where the most time was allocated to. The data component including the database development was predicted to continue throughout the whole phase until all the other components on the other activity arrow were complete. This is because additional functions were needed to be added and uploaded to GitHub regularly to allow other team members to finish their tasks on schedule.

Figure 32 Phase 2.

Figure 33 Phase 3.

Diagram, schematic

Description automatically generatedPhase 4 was the second-to-last stage which included a process for testing which needed to be completed before the record of test processes was to begin, otherwise, the testing would have not been able to be recorded anywhere for future reviews and evaluations.

Diagram, schematic

Description automatically generatedThe final phase consisted of releasing the software. This is where any testing that produced errors was worked upon, as well as any documentation that hasn’t been completed yet. This phase was predicted to not take a considerable amount of time, as the software was practically almost completed at that point, and only small changes occurred.

#### MoSCoW

The MoSCoW technique has helped in the prioritisation of features in the programming and designing missions. The ‘Must’ tasks are tasks that needed to be completed by the end of the project. The ‘Should’ tasks are tasks that ideally should have been completed by the end of the project as well. ‘Could’ tasks could have been completed to show novelty or innovation within a mission. The ‘Would have’ tasks are tasks that would have been completed if all the other tasks above are, and there was still enough time to finish optional tasks.

Figure 32 Phase 5.

Figure 31 Phase 4.

|  |  |
| --- | --- |
| Must ID | Must description |
| **1** | Team Code of Conduct |
| **2** | Project proposal |
| **3** | Skills audit of the team |
| **4** | Risk analysis |
| **5** | Work breakdown, task prioritisation and task allocation |
| **6** | Schedule with agreed deadlines |
| **7** | Ongoing logs of progress, problems, and solutions |
| **8** | Records of communications pertaining to decisions by the group |
| **9** | UML use case diagram |
| **10** | UML use case descriptions |
| **11** | Class diagram |
| **12** | State machine diagrams |
| **13** | Sequence diagrams |
| **14** | Process for testing and reporting on testing, a record of the test process in use |
| **15** | Functionality to allow an appropriate user to add new regular and one-off tasks. |
| **16** | Functionality to allow the user to add relevant information about tasks, such as the type of task, duration, importance and frequency. |
| **17** | Safeguards against common errors, such as unnecessary duplicate tasks or tasks. |
| **18** | Functionality to present a sortable list of existing tasks. |
| **19** | Functionality to allow the user to select and edit task information. |
| **20** | Documentation of the data model, such as an Entity Relationship Diagram or other appropriate tools. |
| **21** | A method for persistent storage of data. |
| **22** | Appropriate methods that allow other components to query and update the data. |
| **23** | Functionality to allow assign tasks to caretakers allowing for the expected time taken, preferences, individual talents and special conditions. |
| **24** | Functionality to provide a clear statement of daily tasks to each caretaker. |
| **25** | Functionality to provide a sortable list of tasks, clearly indicating tasks that have not been assigned, as well as other key information such as task priority. |
| **26** | Safeguards to prevent common errors, such as the same task being assigned twice. |
| **27** | Functionality to provide a way for users to log on, or otherwise securely indicate their identity. |
| **28** | Functionality for adding new users to the system. |
| **29** | Functionality for removing users from the system. |
| **30** | Functionality for editing user information. |
| **31** | Functionality for categorisation of users with appropriate access features. |
| **32** | Software interfaces provide information to other components when they need to know the identity of the current user. |
| **33** | Functionality to allow users to update security information such as passwords. |
| **34** | Functionality to allow caretakers to log the completion of certain regular and one-off tasks, including key information such as who completed the task, the time and additional comments. |
| **35** | Functionality allows caretakers to easily find tasks assigned to them, either completed or pending. This allows caretakers to quickly log task completion and identify tasks that are close to the deadline (or past the deadline.) |
| **36** | Functionality to allow caretakers and managers to edit the record of tasks completed. |
| **37** | Functionality to provide reports on the current status of tasks, particularly overdue tasks. |
| **38** | Functionality to provide reports on the historic performance of the caretakers, particularly signs of poor performance or unequal workload. |
| **39** | Functionality to provide reports on the historic completion of tasks, particularly tasks that are often overdue or not completed satisfactorily. |
| **Should ID** | **Should description** |
| **40** | Apply SCRUM sprints |
| **41** | PERT analysis to determine critical path |
| **42** | requirements model that is detailed, specific, and includes a thorough consideration of exceptions |
| **43** | Package diagrams |
| **44** | Component/activity diagrams |
| **45** | Use boundary value analysis for testing |
| **46** | User guidance |
| **47** | Undo features |
| **48** | Filtering |
| **49** | Add robust data security in the database by allowing for concurrency or by implementing a reusable/flexible software package |
| **50** | Complex presentation of statistics and filtering |
| **Could ID** | **Could description** |
| **51** | develop a beneficial process to assist the development process |
| **52** | adopt and communicate an innovative approach to solving the client’s problem |
| **53** | communicate complex information not taught on the programme |
| **54** | new and effective, as well as a beneficial approach to testing |
| **55** | make interaction process faster, easier, and more reliable for the client |
| **56** | use advanced database features such as views, triggers, and complex constraints |
| **Would have ID** |  |
| 57 | Not applicable – no ‘would have’s’ have been identified in the brief |

Trello has been used as the main way to allocate tasks for the whole team, using features that come with Trello by default. All team members were added to the Trello board, so they could be ‘assigned’ to a specific mission. As seen in Figure 31, the team member responsible for the mission can be seen in the top right corner of the list, as well as all the tasks that need to be complete in that specific mission. Team members can change the label colours as they progress through their mission, as seen in Figure 34. The purple label named ‘help’ is there to allow team members to ask other members for help, including the project manager. Usually, asking for help would occur through direct messaging, but adding a purple label on a task allows for all team members to see which task a team member is specifically struggling with.

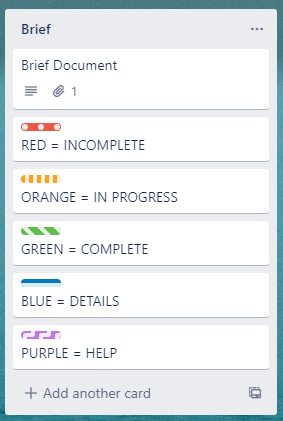


Figure 34 Key for labelling each task within the mission.

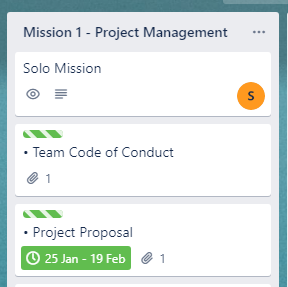


Figure 35 Trello, Task allocation example.

A brief description for each mission has been included in each mission description on Trello, as it was a much more effective and faster way to review the specific qualities that each mission should have, as well as any tasks that could have been completed that go beyond expectations or to show novelty and innovation. The Trello board used for the group project can be found at: <https://trello.com/b/WzBBPqAw/software-engineering>.

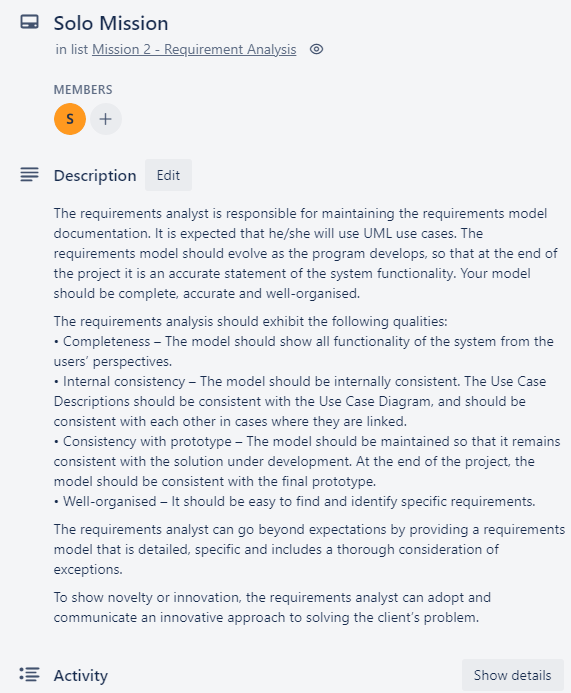


Figure 36 Mission Description.

### Scheduling, Version Control and Revisions

#### Scheduling

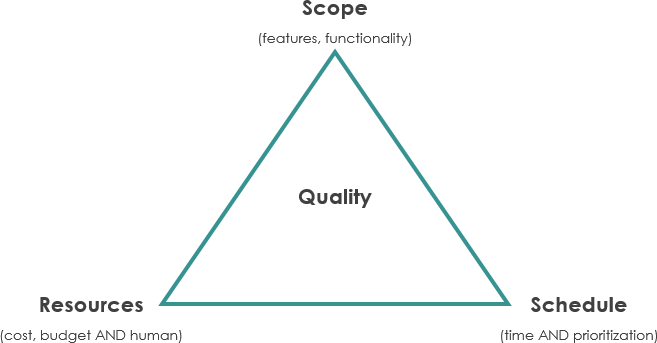
A Gantt chart was a simple technique to plan and schedule project tasks. It displayed a visual flow of when and how long tasks were taking to complete. A column for dates and estimated hours was included to see how many hours in total the project has taken. Milestones in the Gantt chart have been shown as ‘Phases’, and were used to track progress during the project, to see how many more phases there are to go. Phase 1 was dedicated to the project manager finishing all the requirements for the project design and development to properly begin. Phase 2 was the design phase of the UML use case diagrams and descriptions. Phase 3 and 4 overlap with each other and were the main development stages. Phase 4 and 5 overlapped and were the development and testing phases of the project. The original plan for a Gantt chart was to use the Gantt chart addon provided with Trello, but it was deemed not detailed and customizable enough, so a Gantt chart on Microsoft Excel was created instead.

After every milestone is complete, a review has been conducted during meetings to express any concerns, identify any errors or bottlenecks during that phase, checking adherence to standards and checking against requirements. This allowed for a plan to be created if there were any errors during testing in the last phase. Milestones also allowed for potential re-scheduling if a task was not completed on time, or re-allocating of roles if they seemed unfit.

Table

Description automatically generated with low confidence

Figure 37 Gantt chart. Full resolution image available at: <http://unn-w19009505.newnumyspace.co.uk/SEP/GANTT.PNG>

The iron triangle, also known as the project management triangle, has become very familiar to the whole team during the project, as certain constraints have all been traded at some point during the project. The project was able to be completed faster due to decreasing the scope at the beginning of the project, so the most important (M in MoSCoW) tasks were completed first, which was necessary for the functionality of the software. Nearing the end of the project, the functionality of the software has been increased, meaning that team members also had to dedicate more time to the project.

#### Version Control

Version Control Systems are a very valuable tool as they give several advantages, such as:

* Version control
* Repositories
* Collaborative working
* Efficient project management.

Figure 38 The iron triangle.

GitHub was effective at version control because all files could be collaborated on by using different branches for different team members, and when they were ready, they could push to the main branch which is what the final software would be. The main repository has been created on the project manager’s account, for the project manager to be able to manage all the other team members and assign correct permissions. Each file in GitHub has its own history, making it simple to see which changes occurred on which lines specifically, so if any errors have been made, they could be undone quickly and easily. Every team member can review everyone’s code for testing and was a great way to maintain an online backup of the software code.

The main advantage of using a GitHub repository was that team members could collaborate on the same project and update code much easier by using the commit-pull-push method, rather than working in a shared folder on the same set of files, which could have resulted in important code being overwritten or completely deleted. As seen in Figure 39, GitHub has been used frequently and efficiently by all team members, including the use of several branches, and constantly pushing to main.

A screenshot of a computer

Description automatically generated with medium confidence

Figure 39 GitHub Network Graph.

#### Revisions and Methodology

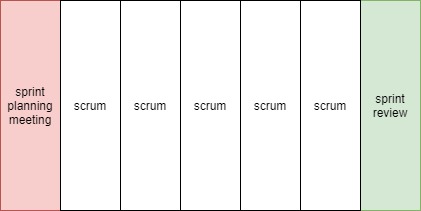
The methodology used for the project was scrum, where in the first meeting, a sprint planning meeting was carried out, and every week during the meeting from them on, a spring planning and a review was to be carried out as well. In the days between the meetings, quick, vertical slices, as seen in Figure 40, included scrum which would be either coding or designing, and closer to the deadline, it would have also included debugging and tuning of the code.

Figure 40 Sprint planning and review.

During the meetings, every team member was asked three questions:

* What have you done since the last meeting?
* What will you do before the next meeting?
* Is anything stopping you from achieving your objectives?

Using a sprint methodology was advantageous, as project control was achieved much easier than with agile methodology, as it is a time-boxed period where the team has to complete a planned, set amount of work. This allowed for meetings to be more productive, as each team member was able to say what has been completed in the last week, what needs to be completed next week, and if there are any constraints that could be addressed straight away before the next week-long sprint begins.

It was the ‘scrum master’s’ job to fix any relevant problems, such as rescheduling or aiding a team member in being able to complete the next sprint successfully. At the end of the final sprint, the software was presented to the company and any relevant stakeholders.

### Logging progress, problems, and solutions

#### Communication

The main form of communication during meetings was Discord. This is because it allowed for both voice chat and screen sharing and was something that the whole team already had before the project started, meaning the whole team knew their way around it. It was an efficient and effective way to communicate with each other. Trello has been used as a way to include extra comments in each mission, for example, a file upload for a specific task, and mission management.

#### Logging of all Progress

For logging all the code created during the development process, a GitHub repository was created as it kept tracks of all changes and version history. It was a great way of backing up code, regardless of what happened to a team member’s local machine. Team members who developed the software used three functions in GitHub, also known as:

* Commit – this is where the commit records changes in the repository, this is done locally.
* Push – this is where the most recent commit, which is received locally, is sent to the repository on GitHub.
* Pull – this pulls any changes from the GitHub repository and merges them into the local repository.

#### Meetings Documented

As seen in the Code of Conduct, weekly meetings that were planned, occurred every week during the project, as well as meetings with Dr Matthew Higgs every fortnight to show the progress of the project. This can also be seen on the Gantt chart. During certain weeks, due to unforeseen circumstances such as a team member not being available to attend a meeting, the meeting was moved onto another day as it was important for the whole team to be present during each meeting.

Group roles such as the initiator, information seeker, evaluator, decision-maker, clarifier, and the summariser were randomised during each meeting, so each team member was able to be in a different role, to increase their skills mentioned earlier in the skills audit. These roles also included a timekeeper, which usually was the project manager. Templates and agenda for meetings were created, to allow focus and avoid rambling or drifting at any time, to make use of the meeting time efficiently. Minutes were taken during each meeting so there was a record of each meeting occurring, and what happened during each meeting. The project manager ensured that no team member dominated the meeting or left anyone else out. Interruptions and bad behaviour were also prohibited and did not occur during any meetings. At the beginning of each meeting, tasks completed during the previous week were discussed, as well as next week’s goals and any constraints that might affect the next week. Every team member had a chance to speak on any issues they may have, either with other team members or a task.

Team meetings have been documented, and the following transcripts of each meeting can be seen below, including minutes and screenshots of the discord call and length if applicable.

|  |  |  |
| --- | --- | --- |
| Meeting Nr. | Date | Minutes (brief notes) |
| Template used | *Example date* | **What has been completed during the last week?**  **What are the goals for next week?**  **Is there anything that may affect next week’s goals?** (for example, other module deadlines)  *Notes:*  *Screenshot of discord:*  Screenshot here |
| 1 | *17/02/2021* | **What has been completed during the last week?**   * N/A – first meeting   **What are the goals for next week?**   * Complete the skills audits before this Friday’s deadline and merge everyone’s skills audits into a PDF   **Is there anything that may affect next week’s goals?**   * N/A – skills audits were completed during the meeting   *Notes:*   * Spoke about the creation of skills audits * Came up with skills for the group for task 2 in the skills audits * Discussed allocation missions and gave opinions on who should do each mission and why, keeping the skill audits in mind * Time: 18:00-19:30   *Screenshot of discord call:* |
| 2 | *26/02/2021* | **What has been completed during the last week?**   * Skills audit, sent off to blackboard   **What are the goals for next week?**   * Code of conduct and a start on risk analysis, this will be done by Sandra.   **Is there anything that may affect next week’s goals?**   * Code of conduct and risk analysis may not be completed on time if the whole focus stays on the other module   *Notes:*   * No other goals as another module has an assignment due soon – Intelligent Systems Part 1 * Time: 18:00-18:14   *Screenshot of discord call:*  N/A |
| 3 | *05/03/2021* | **What has been completed during the last week?**   * Code of conduct was completed. Risk analysis was also mostly completed but needs a glance over by all other team members.   **What are the goals for next week?**   * UML case diagrams and descriptions and a start on class diagrams is possible   **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * Time:18:00-18:12   *Screenshot of discord call:*  N/A |
| 4 | *19/03/2021* | **What has been completed during the last week?**   * Risk analysis fully complete   **What are the goals for next week?**   * Potentially look at sequence and package diagrams, package diagrams are optional but will give extra marks   Carry of with the class diagram as it will need constant updating  **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * Created a Trello for easier task management * Time: 18:00-18:40   *Screenshot of discord call:*  This is where the team decided to call in the group chat made for the SEP project specifically, so logs can be screenshotted of the call occurring, and any extra information that can be screenshotted for proof. |
| 5 | *26/03/2021* | **What has been completed during the last week?**   * Class diagram has been updated once again, sequence and all the other diagrams have been started but not finished   **What are the goals for next week?**   * Carry on with the diagrams.   **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * GitHub – all team members added to repo created by project manager * Time: 18:00-18:46   *Screenshot of discord call:*  N/A |
| 6 | *5/04/2021* | **What has been completed during the last week?**   * Progress on diagrams, they will most likely not be finished and will need constant updating as more elements get added to the software.   **What are the goals for next week?**   * Create a GUI for the software, as some code has already been started and needs to be displayed somewhere   **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * Alex had issues with installing jre in Eclipse, although the issue was resolved as the team member was using the wrong Window Builder, this was all fixed before and during the meeting, which is why it took longer than usual. * Time: 17:00-18:06   *Screenshot of discord call:* |
| 7 | *9/04/2021* | **What has been completed during the last week?**   * A GUI was created a few days after last week’s meeting, this is an early version of it:     **What are the goals for next week?**   * Database mission will need to be started so the GUI can be populated with data. * Work on diagrams and update them as database gets added   **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * Several tabs have also been added to the GUI so the client can access different parts of the software, the title font was made bigger on the 08/04/2021 and uploaded to GitHub * Time: 20:00-20:24      * Time:   *Screenshot of discord call:* |
| 8 | *13/04/2021* | **What has been completed during the last week?**   * Database has been developed, a DB plan dictionary was created on Microsoft Excel and shared within the team   **What are the goals for next week?**   * Class diagram was adjusted to fit the developed database. Finish sequence diagrams if not finished completely yet and look at other optional diagrams if time allows.   **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * Mentioned abstract classes for the database that will be reviewed by the whole team in the next meeting. * Time: 19:50-20:42   *Screenshot of discord call:* |
| 9 | *18/04/2021* | **What has been completed during the last week?**   * Looked at abstract and database classes from one of the team members to make sure they are right as the team member asked for the review. * Looked at how to format data for the tasks completed by a staff member.     **What are the goals for next week?**   * Database development, other programming mission’s development   **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * Decided to dedicate one meeting to complete several diagrams. The class diagram and the entity relationship diagram.      * Mentioned the use of undo and redo functions on several programming missions and looked at how to include them in the code, although they have not been implemented yet * Time: 17:00-18:38   *Screenshot of discord call:* |
| 10 | *23/04/2021* | **What has been completed during the last week?**   * Kieran created a user list class to store caretakers and managers in an ArrayList   **What are the goals for next week?**  Alex – create a caretaker task class that stores the names of users associated with the task   * Managers will need to be added to the database as they have not been added yet   **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * Kieran – created a form to allow people to enter data to create a new user, will continue to work on in the user-management branch 19/04/2021 * Kieran – initial add user form created and pushed it to his branch, not going to merge to main until it has more functionality * Development of populating the GUI      * Entity relationship diagram developed further      * Time: 18:00-18:30   *Screenshot of discord call:* |
| 11 | *04/05/2021* | **Mini-meeting between two team members**  *Notes:*   * between Alex and Sam as Sam needed clarification about certain elements in the database * Time: 20:36-20:58   *Screenshot of discord call:* |
| 12 | *05/05/2021* | **What has been completed during the last week?**   * Alex – good chunk of programming that has been merged to main   **What are the goals for next week?**   * Sam – needs to merge to main * Kieran – will check over any merge conflicts * Alex – change around the class diagram   **Is there anything that may affect next week’s goals?**   * *N/A*   *Notes:*   * Full team meeting was not able to occur last week on the first as a flood happened in one of the team members apartments and they were not available at the time * Time: 14:00-14-33   *Screenshot of discord call:* |
| 13 | *09/05/2021* | **What has been completed during the last week?**   * Database can update records and has merged this update to main * Sam – merged code to his own branch, needs to wait for Alex to continue with his database to make any further progress   **What are the goals for next week?**   * Continue with the programming missions and update any class diagrams or ERD if necessary.   **Is there anything that may affect next week’s goals?**   * *Computer Networks assignment deadline*   *Notes:*   * *Merge conflict occurred; it was only the jar file location.* * Time: 18:00-18:23   *Screenshot of discord call:* |
| 14 | *19/05/2021* | **What has been completed during the last week?**   * Kieran – update user and delete user guis are merged to GitHub, functional with editing the database elements to impact the tables   **What are the goals for next week?**   * Complete the coding for all the other missions and updating everything on Trello so the project manager is able to see which tasks still need to be completed, if any.   **Is there anything that may affect next week’s goals?**   * *Computer Networks assignment deadline*   *Notes:*   * Coding missions have taken a considerable amount of time because of another assignment deadline, but the team is on track to complete before the deadline date * Time: 17:45-18:13   *Screenshot of discord call:* |

## Mission 2

### Requirements Analysis

#### A statement of the purpose of the system and how it meets the client’s needs.

The purpose of the system is to move to an electronic system for maintaining the list of tasks and distributing them amongst the caretakers evenly. This is because the company has several buildings on site, meaning an electronic system would be more efficient to use by all staff members, including caretakers and managers.

The system will meet Capytec plc’s needs as it includes the main four features provided in the proposal. The features required were:

* maintaining a list of tasks
* support a team of caretakers in allocating daily tasks amongst themselves
* maintain a record of them certain tasks have been completed and who completed them
* provide various reports on task completion and allocation to interested parties.

#### Requirements modelling - UML Use Case Diagram

A UML case diagram was created to show the main features of the system, including optional buttons that can be pressed by the users using the system. The following UML case in Figure 41 includes both caretakers and managers, and shows which users has permission and can see the specific features of the software.

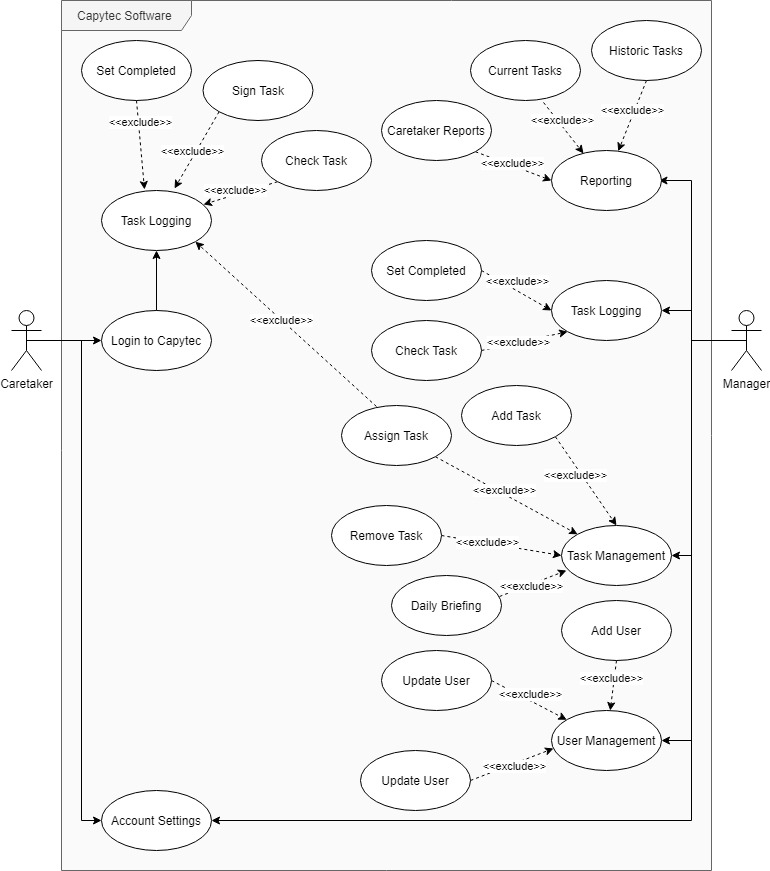


Figure 41 Use Case Diagram.

#### Requirements modelling - UML Use Case Descriptions

UML use case descriptions have been made to showcase certain functionalities of the system before it is fully developed, to discover any alternative or exceptional scenarios that may occur and could be thought about before the development began. Use case descriptions helped to capture requirements of the system, described the functionality of the most important features users would be using, and allows for the developers to see how the user would complete a sequence of steps to complete each use case.

|  |  |
| --- | --- |
| Use Case | Logging in |
| Summary | Logging in as a Caretaker by filling in username and password. |
| Actor | Caretaker |
| Trigger | ‘Login’ button on the first screen when Capytec software is run. |
| Primary Scenario | 1. User clicks on the ‘username’ field 2. User clicks on the ‘password’ field 3. User presses the ‘login’ button **[A1: User has not filled in any fields] [E1: User has incorrectly input their username or password]** 4. User has successfully logged into the Capytec software 5. Use case terminates |
| Alternative Scenario | **A1 User has not filled in any fields**   1. User is shown a message stating ‘ Invalid username or password’ 2. User is prompted to enter a username 3. User is promoted to enter a password 4. Continue to primary scenario from point 4 |
| Exceptional Scenario | **E1 User has incorrectly input their username or password**   1. User is prompted to not leave fields empty and to try again. 2. Continue to primary scenario from point 4 |
| Pre-Conditions | User has the Capytec software installed. |
| Post-Conditions | User has successfully logged into the Capytec system. |
| Assumptions | User has a PC. |

|  |  |
| --- | --- |
| Use Case | Reviewing current tasks |
| Summary | In the Reporting tab, review Current Tasks |
| Actor | Manager |
| Trigger | ‘Current Tasks’ button on the Reporting tab of the Capytec software. |
| Primary Scenario | 1. User clicks on ‘Current Tasks’ button 2. User is taken to a new ‘Current tasks’ screen 3. User clicks on ‘Generate Report’ button 4. User is able to view current tasks, including one off tasks and repeated tasks **[E1: User is unable to see any tasks on the screen]** 5. Use case terminates |
| Alternative Scenario | N/A |
| Exceptional Scenario | **E1 User is unable to see any tasks on the screen**   1. This means that no tasks are currently added into the database, or all current tasks are completed 2. User does not need to review any current tasks unless new tasks are added 3. Continue to primary scenario from point 5 |
| Pre-Conditions | User has the Capytec software installed. |
| Post-Conditions | User has successfully reviewed current available tasks. |
| Assumptions | User has a PC. |

|  |  |
| --- | --- |
| Use Case | Sign task |
| Summary | In Task Logging tab, sign a task |
| Actor | Caretaker |
| Trigger | ‘Sign task’ button on the Task Logging tab. |
| Primary Scenario | 1. User clicks on ‘sign task’ button 2. User is prompted with a new ‘sign task’ window 3. User is asked to select a task ID to sign **[E1: User cannot see any task ID’s]** 4. User clicks on ‘set task as signed’ 5. Use case terminates |
| Alternative Scenario | N/A |
| Exceptional Scenario | **E1** **User cannot see any task ID’s**   1. This means that no tasks need to be signed off. 2. Continue to primary scenario from point 5 |
| Pre-Conditions | User has the Capytec software installed. |
| Post-Conditions | User has successfully signed a task. |
| Assumptions | User has a PC. |

|  |  |
| --- | --- |
| Use Case | Add user |
| Summary | Add user to Capytec software. |
| Actor | Manager |
| Trigger | ‘Add user’ button in the User Management tab. |
| Primary Scenario | 1. User clicks on ‘add user’ button 2. User is shown a new pop-up windows and asked to fill all fields. 3. User inputs first name **[A1: User is missing first name]** 4. User inputs surname **[A2: User is missing surname]** 5. User inputs username **[A3: User is missing username]** 6. User inputs password **[A4: User is missing password]** 7. User is prompted to confirm password **[A5: User did not confirm password]** 8. User is promoted to select a position **[A6: User did not select a position]** 9. User is promoted to select three skills **[A7: User did not select all three skills]** 10. User clicks on ‘ insert user’ button 11. User has successfully added a new user to the system 12. Use case terminates |
| Alternative Scenario | **A1 User is missing first name**   1. User is asked to input a valid first name. 2. Continue to primary scenario from point 3   **A2 User is missing surname**   1. User is asked to input a valid surname. 2. Continue to primary scenario from point 4   **A3 User is missing username**   1. User is asked to input a valid username. 2. Continue to primary scenario from point 5   **A4 User is missing password**   1. User is asked to input a password 2. Continue to primary scenario from point 6   **A5 User did not confirm password**   1. User is asked to confirm the password 2. Continue to primary scenario from point 7   **A6 User did not select a position**   1. User is asked to select either a caretaker or a manager position   Continue to primary scenario from point 8  **A7 User did not select all three skills**   1. User is asked to select ALL three skills   Continue to primary scenario from point 9 |
| Exceptional Scenario | N/A |
| Pre-Conditions | User has the Capytec software installed. |
| Post-Conditions | User has successfully added a user to the system. |
| Assumptions | User has a PC. |

## Mission 3

### Software Analysis

#### Class diagram

The class diagram is a UML diagram which was created to plan out the structure of the program before development, so each programmer understood how each of their individual systems will interact and relate with one another before starting development. Due to its layout being the same as the final product classes, it is useful as a tool for individuals to easily understand the layout of the program without having to read through the code due to its visual nature. The diagram shows information such as the classes within the program, as well as the variables and methods they contain alongside their respective access modifiers. The arrows linking the individual classes together also indicate the relationship between the different classes of the program, providing insight into how they interact with one another.

Diagram

Description automatically generated

Figure 42 Class diagram.

#### Sequence diagrams

Sequence diagrams were used in order to show the process over time of individual tasks within the system being completed, and how the various objects of the program communicate with one another during this process to reach the desired outcome of the user.

Sequence Diagram 1:

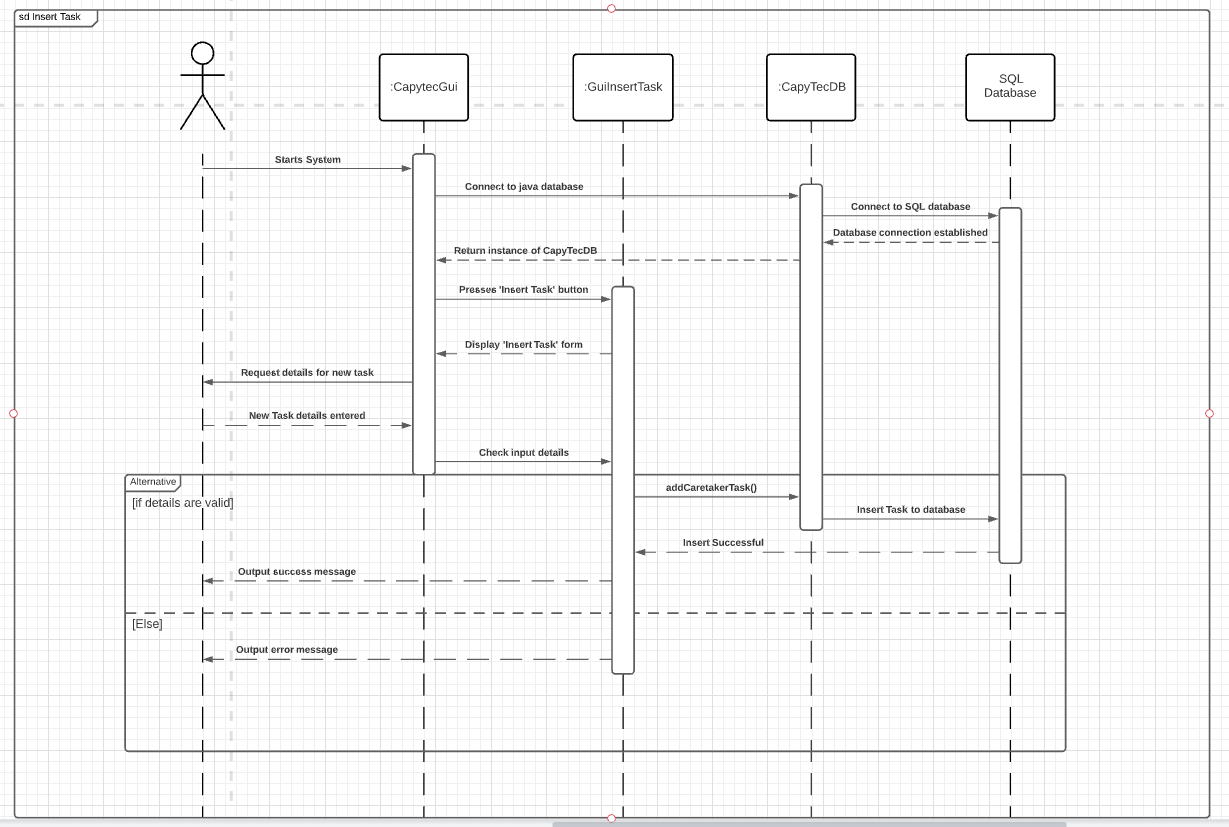


Figure 43 Insert task

Sequence Diagram 2:

Updating task details, to set a task as signed.

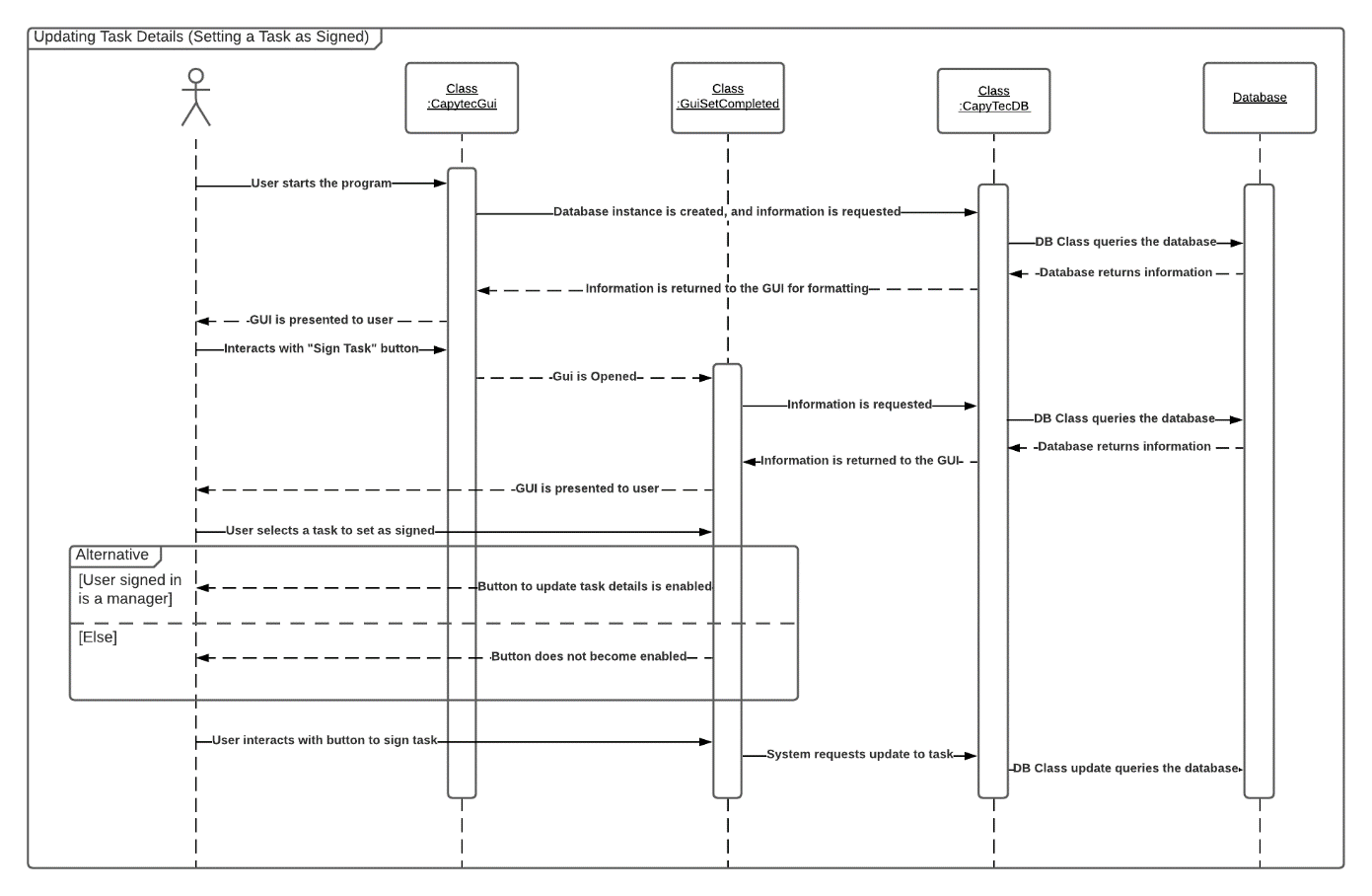


Figure 44 Updating task details.

Sequence Diagram 3:

Insertion of a Caretaker user into the system database

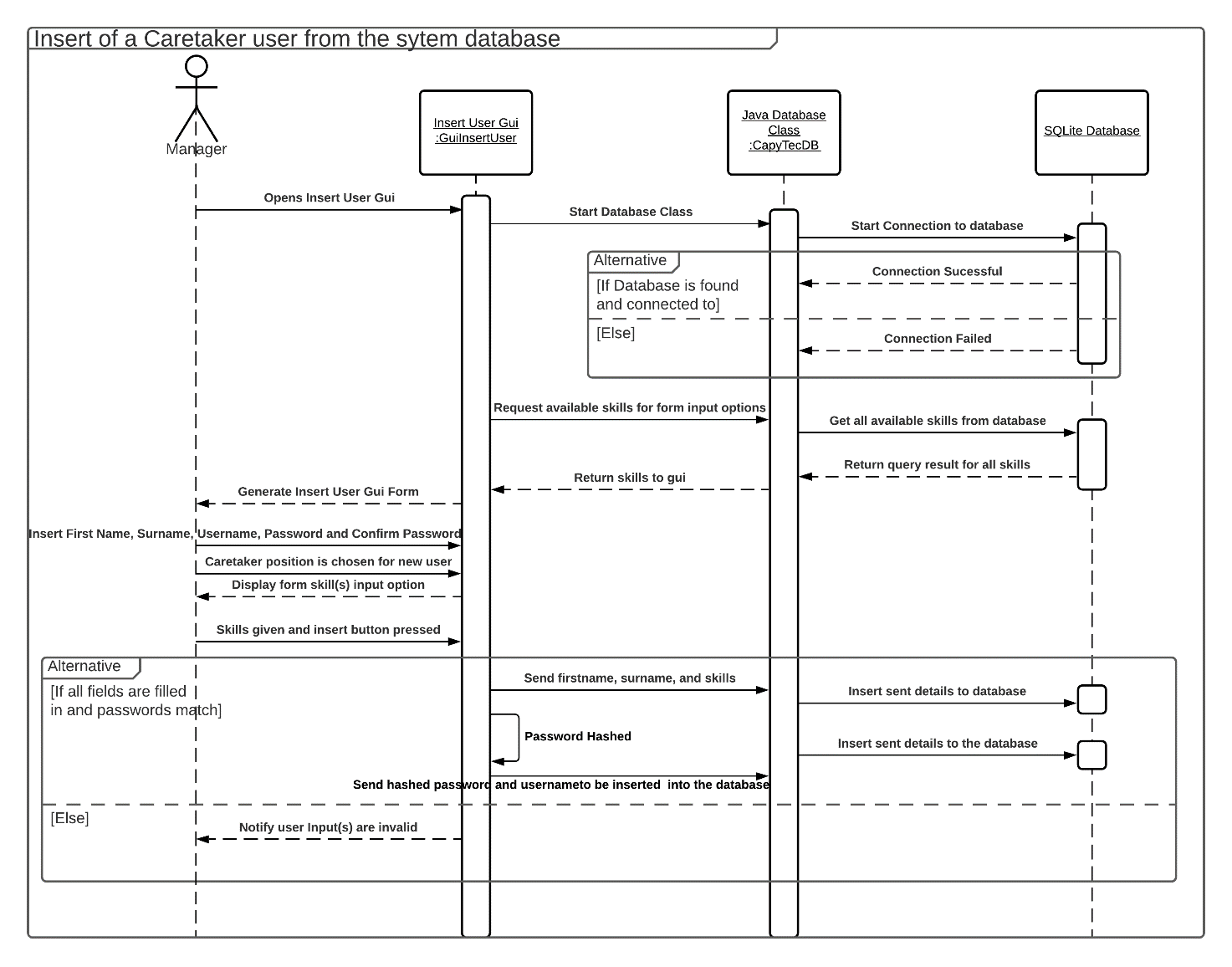


Figure 45 Insert of a caretaker user from the system database.

#### Activity Diagram

An activity diagram was created to showcase the functionality of inserting a user into the system. At the beginning, it is assumed that the user has Capytec software installed. The user needs to enter username and password, and if the user does not log in as an admin, the activity diagram ends. Otherwise, the software allowed the user to continue and press on the add user button. After that, a fork has been added where the user is able to input the first name, surname etc. at the same time/at any time, then a condition is called for each field, checking if it is empty or not. After the condition is checked, the user is added onto the system and the activity diagram ends.

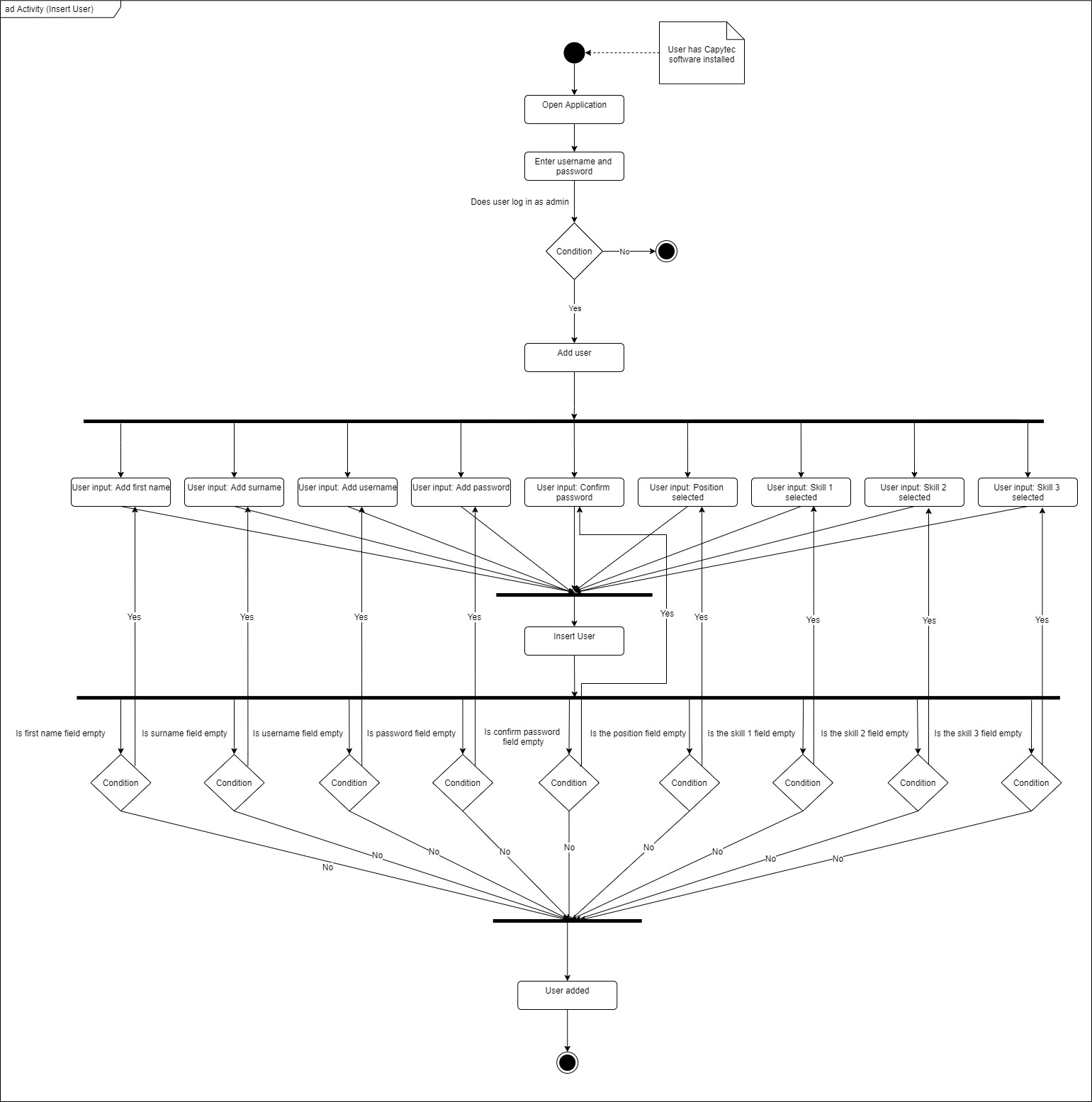


Figure 46 Activity Diagram

## Mission 4

### Testing

Testing was done using a test plan, and was mostly compromised of functional testing, as most of the test cases tested what the system does. Non-functional testing would have been useful if the appearance of the software was something important, and if Capytec requested the software to look in a specific way. Individual elements of the application were tested from a client-side perspective, and the expected result of the test was compared with the actual result, allowing the tester to determine if the functionality of program was performing the functionality as intended. Tests were then labelled as either a success or fail, allowing the tests that did not produce the intended output to be easily identified, and then looked at in detail through the use of bug reports, allowing potential solutions to the problem to be identified. Screenshots were included throughout the testing process as evidence that the tests were either working or not working as intended. Some tests had to be conducted from the perspective of multiple potential logged in users, due to the difference in features available to caretakers and managers within the program, so that it could be shown that certain features were only enabled when intended for that user.

Alpha testing was designed at the beginning of the development, using the use cases in Mission 2, and was carried out near the end of the project, where team members reported any bugs that occurred during the testing. The alpha testing was done on the first prototype of the Capytec software, using the test plan that can be seen below. White-box testing is a testing technique that was used to check the internal functioning of the system, where the programmers were aware of how the software system worked internally. Black-box testing also occurred, as testing the user-side of the software where the internal functioning is not known provided valuable information and any bugs which a potential user might come across could be fixed before the release of the software.

Ad hoc testing was started at an early stage of the project, where the programmers would randomly test classes as they were being created and fixed any minor errors as they occurred. The main reason for ad hoc testing was to find defects as the software was being developed, and did not require any documentation or planning, making it an easy and effective way to test smaller chunks of the program. However, ad hoc testing which produced defects would have been hard to reproduce, as there would be no testing steps or requirements mapped to it.

Unit testing was conducted on the CapyTecDB class to test the functionality of the methods that were manipulating the database used within the application. This was done using another class designated for database testing called TestingDBMethods. The main reason and benefit for utilising unit testing for the database methods was to test that the functionality of the database was working as intended, whilst removing any external variables that could affect the outcome of the SQL methods such as the GUI gathering user values incorrectly, which was done by isolating the tests for the database from the rest of the program. Dummy data was created during the unit tests when required, which were used for the remainder of the test, and then removed from the database at the conclusion of the test therefore leaving no lasting implications on the database that would otherwise interfere with the operation of the application.

#### Bug tracking system

A bug tracking system was created to report any tests that might have failed, therefore creating a defect in the software. Bug tracking has been a valuable tool within the project as it allowed team members to communicate more effectively about defects in the system, the template also provides an easy way to document every bug found consistently.

The following template includes:

* A defect ID, which is a number, given to the test that failed, usually in chronological order
* Product version, which is the version of the software being tested, it will most likely be the prototype
* Test ID which is the same ID from the testing table
* Defect description which is a description of the test that failed
* Date raised: date that the defect was found
* Reporter: owner of the test that failed
* Date closed: date that the defect was fixed
* Fixed by: name of team member that fixed the following defect
* Steps to reproduce: any of the team members are able to reproduce the defect and fix it by following the steps written
* Severity: how much does the defect affect the whole system
* Priority: how quickly does the defect need to be fixed
* Status and date: Status of the current defect and an area for dates to document when the defect was first open, assigned, retested, and closed

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Defect ID |  | Product Version |  | Test ID | |  |
| Defect description | | **Date raised** |  | **Reporter** | |  |
|  | | **Date closed** |  | **Fixed by** | |  |
| **Severity** | | | | |
| Steps to reproduce | | **Critical** | **Major** | | **Minor** | |
|  | | **Priority** | | | | |
| **High** | **Medium** | | **Low** | |
| **Status**  **Date** | **Open** | **Assigned** | **Retest** | **Closed** |
|  |  |  |  |

#### Process for testing

Testing has been split among four team members evenly, giving each team member one ‘tab’ to test.

* *User Management* – Alex Heaton
* *Task Management* – Kieran Robinson
* *Task Logging* – Sam Farnworth
* *Reporting* – Sandra Czernik

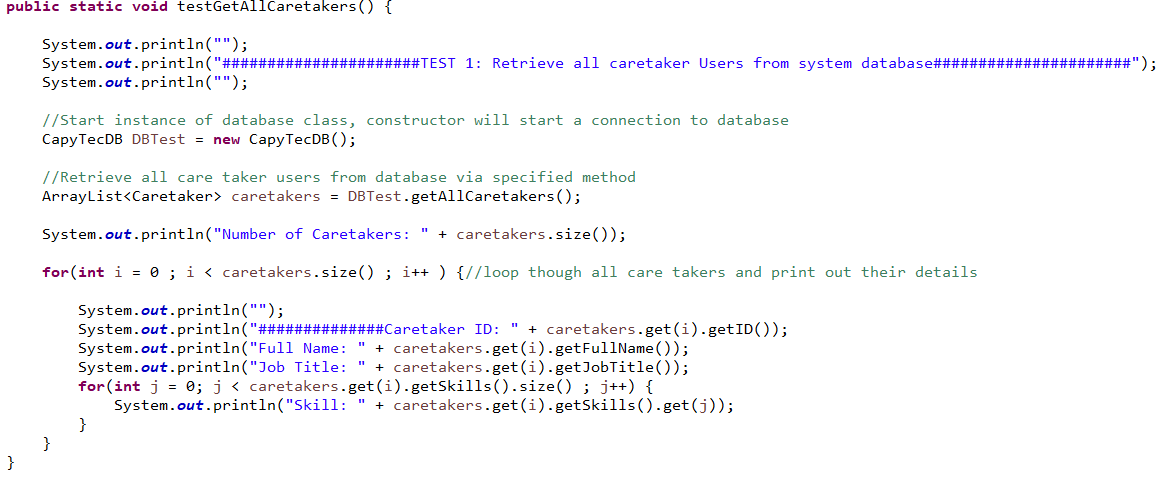
#### Full record of test process in use

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| ID | Test Owner | Title | Description | | Expected | Actual | | Result (screenshot) | | |
| **1** | **KR** | Dynamically display tasks within a table from the database. | Update table dynamically displayed on the task management tab with tasks from the database. | | All tasks stored within the database are displayed within the table, along with their corresponding details. | SUCCESS  Tasks from the database are displayed within the task management table alongside their stored details. | |  | | |
| **2** | **KR** | Insert task form appears when user presses ’Add Task’ within the Task Management tab. | Insert task form appears to the user with fields to complete in order to insert a task on the pressing of the ‘Add Task’ button. | | On pressing of ‘Add Task’ button, the ‘Insert Task’ form appears to the user. | SUCCESS  Insert task form appears when user selects the ‘Add Task’ button on the task management tab as intended. | |  | | |
| **3** | **KR** | Task inserted to database on completion of insert task form with entered details, when all required fields are completed. | User must complete all required fields (name, description, at least one type, start date, and due date) and press the ‘Insert Task’ button. This will insert a task with the same details to the database, which will be viewable within the task management table. | | Task will only be inserted if all required fields are completed, otherwise an error message will display within the GUI and the task will not be added. | SUCCESS  Task would only insert after name, description and name fields were completed else would throw an error message. Task was successfully added to the database after required fields were completed as can be seen through the addition of the task to the Task Management table, which is dynamically updated from the database. | | Inserting image... | |
| **4** | **KR** | Assign task form appears when user presses ‘Assign Task’ button | Assign task form appears to the user with a dropdown menu to assign the currently logged in user with a task when the ‘Assign Task’ button is pressed. Only possible when logged in as caretaker. | | Should only see Assign Task button when logged in as caretaker. Form should appear on button click. | SUCCESS  Assign task form appears when user selects the ‘Assign Task’ button on the task management tab as intended. | | Inserting image... | |
| **5** | **KR** | Task assigned to currently logged in user on completion of allocate task form, but only if task is not yet assigned. | Currently assigned user should be set to currently logged in user for the task that was chosen within the Assign Task dropdown. | | Database should be updated on completion of the Assign Task form, but only if the task was not already assigned otherwise an error will be displayed. Should be visible within task management table. | SUCCESS  ID of the caretaker who is currently logged in populates the column of the task management table indicating the assigned user for the task. As the table is dynamically updated from the database, this shows the task was successfully allocated to the logged in user. | |  | |
| **6** | **KR** | Delete task form appears when user presses ‘Remove Task’ button | Delete task form appears to the user asking for an ID of a task to be deleted. | | Delete task form should appear on ‘Remove Task’ button click within the task management tab | SUCCESS  Delete task form appears when user selects the Remove Task’ button on the task management tab as intended. | | Inserting image... | |
| **7** | **KR** | Task removed from database when user enters valid ID of existing task to be removed | Task should be removed from the database when the user enters an existing ID for a task, removing it from the task management table. | | Task should be removed from the database, but only if a valid existing ID for a task is input by the user. Change should be visible within the table on the task management tab. | SUCCESS  Task 7 was successfully removed from the table and database when the task ID was entered into the ‘Delete Task’ form. Task would not be deleted if invalid ID was entered and would throw an error message. | | Inserting image...  Inserting image... | |
| **8** | **KR** | Daily briefing appears for currently logged in user when user presses ‘Daily Briefing’ button | Daily briefing should appear for the currently logged in caretaker when pressing ‘Daily Briefing’. Button should only be visible when logged in as a caretaker | | Daily briefing button should only be visible if logged in as caretaker. Briefing should display assigned tasks with descriptions and due dates. | SUCCESS  Daily briefing appears for the current logged in caretaker showing the tasks, description, and related due dates. | | Button appears as caretaker    Button hidden as manager | |
| **9** | **KR** | Column headers can be clicked within the task management table to sort the table by their values. | Column headers within the task management table can be clicked in order to sort them in either ascending or descending order. | | Tasks should be sorted by due date by clicking on the due date column. This will be done in both ascending and descending order. | SUCCESS  Tasks were successfully sorted by due date in both ascending and descending order. | |  | |
| **10** | **KR** | Minimum and maximum character lengths for task name, description and extra considerations | When inserting task, task name must be between 7 and 30 characters, description must be between 10 and 75 characters, and extra considerations can be no longer than 75 characters. Error message should be shown and task not added if this is not followed. | | If task name is below 7 characters of above 30 errors print an error and do not insert task. The same outcome should take place if description is shorter than 10 characters or over 75 characters, or extra considerations is longer than 75 characters. | SUCCESS  If input lengths were unsupported error messages were printed and task insertion did not take place. Test was successful. | |  | |
| **11** | **SC** | Caretaker reports button | Testing for the functionality of the caretaker button once pressed. | | The button will open a new screen with the caretaker history, along with a drop-down menu that allows for selection of a caretaker. Generate report button should not be present at this time. | SUCCESS  Button shows a new screen with a dropdown menu for a ‘selected caretaker’ with the dropdown menu functioning. Generate report button is not present at this stage. | |  | |
| **12** | **SC** | Current tasks button | Testing for the functionality of the current tasks button once pressed. | | The current tasks button should take the user to the current tasks screen, which should be unpopulated. | SUCCESS  Button shows a new screen with the heading ‘ current tasks’ and does not show a drop-down menu, and only the generate report button, as expected. | |  | |
| **13** | **SC** | Historic tasks button | Testing for the functionality of the historic tasks button once pressed | | The historic tasks button should take the user to the task history screen, which should be unpopulated and should include a drop-down menu. Generate report button should not be present at this time. | SUCCESS  Pressing on the historic tasks button takes the user to the task history screen and displays a drop-down menu with tasks to be selected. No generate report button can be seen. | |  | |
| **14** | **SC** | Generating caretaker reports | Testing for the dropdown menu and generating caretaker reports button. | | The button for generating reports in the caretaker history button should not be visible until a caretaker has been selected from the drop-down menu. | SUCCESS  The generate reports button cannot be seen when a caretaker has not been selected yet. | | Graphical user interface, text, application  Description automatically generated | |
| **15** | **SC** | Generating current tasks | Testing for the dropdown menu and generating current tasks button. | | The button for generating reports should be there as soon as the current tasks screen is shown, as there is no need to select any data before displaying. | SUCCESS  The current tasks screen shows the generate report button as soon as the button to access this screen this pressed. | | Graphical user interface, text, application  Description automatically generated | |
| **16** | **SC** | Generating historic tasks | Testing for the dropdown menu and generating historic tasks. | | The button for generating reports in the historic tasks button should not be visible until a tasks has been selected from the drop-down menu. | SUCCESS  The button for generating reports in the task history screen cannot be seen until a task is selected from the drop-down menu. | | A picture containing text, athletic game, screenshot  Description automatically generated  A screenshot of a computer  Description automatically generated | |
| **17** | **SC** | Changing to different tab | Testing if pressing on a different tab will change the tab from ‘Reporting’ to the new tab. | | While on the ‘Reporting tab’ an admin should be able to press on any other tab, such as ‘user management’ for example, and the user should successfully be taken to that selected page. | SUCCESS  While on the ‘Reporting’ tab, another tab could successfully be pressed and accessed, and generates the content suitable for that screen. | | Graphical user interface, text, application, table  Description automatically generated | |
| **18** | **SC** | Changing element in the Caretaker history dropdown menu | Testing to see if changing one of the elements from the dropdown menu changes the relevant information once ‘generate report’ button has been pressed | | Once ‘caretaker reports; has been pressed and the user is on the caretaker history screen, selecting different caretakers and pressing ‘ generate report’ should display different values on the screen, meaning the database is successfully updating information as the drop-down elements get changed. | SUCCESS  When selecting caretaker ‘2’ from the drop-down menu, 2 tasks can be seen in the completion history. When the element ‘2’ was selected, and the generate report button pressed, the screen changed to show that the completion history is empty, meaning the task has not been completed yet. | | Graphical user interface, text, application  Description automatically generated  Shape  Description automatically generated with low confidence | |
| **19** | **SC** | Displaying one-off tasks in the ‘current tasks’ section | Testing to see if one-off tasks are printed out correctly in the ‘current tasks’ section | | Pressing ‘generate report’ in the current tasks screen should successfully print one-off tasks on the screen, with the first task at the top and the last task at the bottom, chronologically sorted. | SUCCESS  The current task screen successfully shows one-off tasks in chronological order. | | Text  Description automatically generated | |
| **20** | **SC** | Displaying repeated tasks in the ‘current tasks’ section | Testing to see if repeated tasks are printed out correctly in the ‘current tasks’ section | | Pressing ‘generate report’ in the current tasks screen should successfully print repeated tasks on the screen, just below the one-off tasks, with the first task at the top and the last task at the bottom, chronologically sorted. | SUCCESS  The current task screen shows repeated tasks in chronological order after the one-off task list is displayed. | | Graphical user interface, text, application, email  Description automatically generated | |
|  | **Task Logging Tab** | | | | | | | | |
| **21** | **SF** | Dynamic display of tasks within a table from the database | Testing to see if table loads in all tasks correctly, in the order that is expected. | | Upon starting the program and opening the tab, all tasks should be presented to the user, with on-going one off tasks being displayed first, followed by repeated tasks, followed by completed one-off tasks. | SUCCESS  Table correctly loads, showing all tasks, ongoing, repeated, completed, in the correct order. Correctly displays all information, including date completed, due, and whether or not the task needs signing or checking, as expected. | |  | | |
| **22** | **SF** | Open submenu in order to set a task a task as having been completed. | In order to set a task as having been completed, the “Set Completed” submenu must be opened first, which is done by pressing the “Set Complete” button. | | The “Set Completed” submenu will open. | SUCCESS  After pressing the button, the submenu / interface correctly opened, allowing the user to start the process of setting a task as having been completed, as expected. | |  | | |
| **23** | **SF** | Open submenu in order to set a task as having been signed. | In order to sign a task, the “Sign Task” submenu must be opened first, which is done by pressing on the “Sign Task” button. | | The “Sign Task” submenu will open. | SUCCESS  After pressing the button, the GUI correctly opened, allowing the user to select a task to set as signed, as expected. | |  | | |
| **24** | **SF** | Viewing the dropdown menu for tasks to change, within the set task as complete submenu. | Within the “Set Complete” submenu is a dropdown box which allows the user to select a task to set as completed. This dropdown should only contain task IDs for tasks that the logged in user is assigned. | | As the logged in user has the ID 6, only a single task with the ID of task 5 should appear in the dropdown. | SUCCESS  After clicking on the dropdown box, it expanded revealing there only being the default “select a task” ID option, as well as the expected ID of 5. | |  | | |
| **25** | **SF** | View the dropdown menu for tasks to change, within the set task as peer checked submenu. | Within the “Check Task” submenu is a dropdown box which allows the user to select a task to set as checked. This dropdown should only contain task IDs for tasks that the logged in user is not assigned to, as caretakers cannot check their own task. | | As the logged in user has the ID 6, the only task that should be visible within the dropdown is the task with ID 4. | SUCCESS  After clicking on the dropdown box and having it expand, it revealed that the only option, besides the default, was the expected task with ID 4. | |  | | |
| **26** | **SF** | Selecting a task to sign a task within the “Sign Task” submenu, when not logged in as a manager. | Within the “Sign Task” submenu is a dropdown, allowing the user to select a task to set as having been signed. This dropdown should only contain task IDs for tasks that need signing and haven’t been. However, only managers have permission to sign tasks, and thus a regular caretaker should not be able to. | | As beforehand, the only task that required signing was a repeated task that had already been signed (task 3), it had to be set as completed, resetting its state as having been signed. After this when the task is selected as to be signed while logged in as user ID 6, who is not a manager, the button should be disabled and not allow this to occur. | SUCCESS  After selecting the only task available, 3, the button remained disabled and wouldn’t allow the user to sign the task as completed, as expected. | |  | | |
| **27** | **SF** | Selecting a task to sign a task within the “Sign Task” submenu, when logged in as a manager. | As before, within the “Sign Task” submenu is a dropdown, allowing the user to select a task to be signed. While signed in as a manager, the option to sign the task should remain enabled | | After selecting the task to be signed (task 3), the button should remain enabled and let the user set the task to signed, as the user who is logged in, ID 2, is a manager. | SUCCESS  After selecting the available task, the button was set to enabled, allowing the user to sign off on the task, as expected. | |  | | |
| **28** | **SF** | Selecting a task to set as completed within the “Set Completed” submenu, when the task is not ready to be set as complete. | Within the “Set Completed” submenu is a dropdown, which allows the user to select a task to set as completed. However, it should only be possible to set a task as completed if it is ready to, which may require it to be signed by a manager or checked by another caretaker. | | After opening the submenu, while logged in as user ID 7, task 3 should appear in the dropdown box, however the user should not be able to set the task as complete. This is because the task requires signing and hasn’t been signed by a manager. | SUCCESS  After selecting the available task, the button remained disabled, meaning that the user was unable to mark the task as complete, as expected. | |  | | |
| **29** | **SF** | Selecting a task to set as completed within the “Set Completed” submenu, when the task is ready to be set as complete. | As before, the “Set Completed” submenu has a dropdown, allowing users to set a task as completed. However, this should only be possible if a task is ready to be marked as complete. | | After opening the submenu, while logged in as user ID 7, task 3 should appear in the dropdown box, after having been signed. From this point, it should be possible to mark the task as completed. | SUCCESS  After selecting the task, the button became enabled, allowing the user to mark the task as completed, updating its completion date. | |  | | |
| **30** | **SF** | Dynamic display of tasks within table from database, after an attribute has been updated. | While in a previous test it was shown that the table was able to load in data, task details can change, either due to tasks being completed, signed or checked, so this should also be checked. | | A one-off task would be selected, before being checked or signed, if required. Following this, the task would be marked as completed, which should set the completion date to the current date, as well as storing who signed or checked the task as complete. | PARTIAL SUCCESS  As seen in the before and after pictures, while the “completed date” for Task 2 updated to the current date, as expected, a mistake also occurred.  While this happened, the assigned caretakers on the task went from caretaker 5, to the task not being assigned. Upon analysing the code for the set completed GUI, there is no point where the assigned caretakers should change. | | Before    After: | | |
| **31** | **AH** | Logging in as any user | Testing the functionality of the login feature in the login GUI. | | Logging in with a correct username and password should log the end user into and account that has corresponding username and password. | SUCCESS  With the username of Alex and the password admin1 the login feature successfully recognised that the username and hashed form of the password matches up and the main Gui was opened up while disabling the login Gui. | |  |
| **32** | **AH** | Generating GUI for managers | Testing to discover how the main GUI generates the User Management tab in accordance to a manager being logged in | | The User management tab should generate if a manager is logged into the system | SUCCESS  The user management tab is generated, left end of the red circling, as manager user username Sam with password of samf is used. The other tabs with for manager type users are also generated such as the reporting tab, task logging and task management tab. | | Inserting image... |
| **33** | **AH** | Generating GUI for caretakers | Testing to see how the main GUI generates depending on there being a caretaker logged in. | | The user management tab should not generate if a caretaker logged in. | SUCCESS  The inputs for the login was that of a caretaker with the caretaker account username input of Dan and the password for the caretaker being caretakerdan. This was a valid caretaker account that was recognised by the system and only the task management tab and the task logging tab was generated for the account. | |  |
| **34** | **AH** | Adding a caretaker user to the database | Testing to see how the system deals with processing the inputs from the user and then adding them to the database | | The add user GUI should add the manager with the user inputted fileds to the database as well as hashing the password and storing it on the database | PARTIAL SUCCESS  With the inputs as seen in the first image the new caretaker account with the username of J.Robinson was created and upon the restart of the program the new caretaker can be seen in the table displayed proving that the records have been added to he database however due to a problem with the display of skills of the accounts none can be seen. | |  |
| **35** | **AH** | Deleting user from database | Testing to see how the system deals with taking the order to delete a user on the database from the end user’s specific input. | | The specified user account should be removed from the database. This includes records on user table, login table, user skill table. | SUCCESS  Upon deciding to delete the user account that was added from the last test the id for that user was found by observing the user management table and finding the id of the user James Robinson. The id was entered into the Gui and the Delete user button was pressed.  This followed by the notification indicating what user was deleted by the inputted id.  When the table of users was reloaded the user was gone. | | Inserting image... |
| **36** | **AH** | Generate add user form relative to position (job title) of the user | Testing the add user form's ability to hide or show certain elements of the form depending on what type of user account is selected. | | The insert user GUI should only show the option to assign skills to a caretaker and not a manager therefore hides the option from the manager | SUCCESS  Within the insert user Gui the position of the user was initially empty. The position was then changed to caretaker. Upon the moment caretaker was selected the Gui adjusted itself to allow 1-3 skill inputs since this is exclusive for caretakers.  When the position was changed to manager the options to give skills was taken away as managers shouldn’t have this option. | |  |
| **37** | **AH** | Adding a manager user to the database | Testing the addition of a caretaker account to the database via the insert user GUI | | The caretaker user account should have all its fields inputted by the end user added to the database while also adding the hashed version of the user password. The caretaker skills should also be inputted into the corresponding tables within the database | SUCCESS  The example manager Dominic was inputted in the insert user form. Since they are a manager the manager position is selected.  When the insert button was pressed and the operation got carried out the new user was added as can be seen in the updated table as seen in the bottom image. | |  |
| **38** | **AH** | Providing the incorrect/invalid username on login | Testing to see how the system deals with an incorrect username input | | The login Gui should display a message to the user explaining what is wrong with their input | PARTIAL SUCCESS  When an incorrect username was inputted to the login Gui the Gui updated a label to notify the user that the username but also the password is invalid since it isn’t entered. The login process doesn’t start either showing that the system recognises not to carry out the full process.  One issue with the Gui however is when there is an input in both username and password the message disappears. | |  |
| **39** | **AH** | Providing the incorrect/invalid password upon login | Testing to see how the system deals with an incorrect password input | | The login Gui should display a message to the user explaining what is wrong with their input | PARTIAL SUCCESS  When an irrelevant password is inputted into the login Gui by itself there is an outcome like in the previous test where the Gui notifies the user that the password or username is invalid however this message disappears when there is also a user name input | |  |
| **40** | **AH** | Invalid inputs entered in the insert user form | Testing to see how the insert user Gui system handles incorrect/invalid inputs from the user. Since the database doesn’t support emojis some will be inputted to test how the system deals with them | | The user insert process should remove any unwanted characters from the input given by the user. | PARTIAL SUCCESS  The system didn’t put the new insert on the database which proves that it won’t allow invalid characters such as emojis to be added however this isn’t the expected outcome as the input should’ve occurred to the database but with the invalid characters removed. | |  |
| **41** | **KR** | Database modifications when using CapytecMain class and the login system is enabled. | When the CapytecMain class is used to launch the program use a method that manipulates the database such as Insert Task or Remove Task. | | The database should be modified as expected when running the program through the CapytecMain class instead of the CapytecGui class. Login system will be enabled and should not impact the program negatively. | FAIL  The database shows as being locked when attempting to manipulate the database when running the program through the CapytecMain method, in order to add the login system to the program. | |  |
| **42** | **KR** | Insert Task with multiple types. | A task should be able to be inserted with 2 or 3 types, which should update the database and be displayable in the task management table. | | Database should have a new task added, with three types all stored and displayed within the table. | FAIL  The task is inserted and all tasks passed, although the database only inserts a task with the first type. | |  |
| **43** | | **SF** | Generate a report for specific caretaker. | After making a modification to the code to set combo boxes to only be of type string, this test was re-tested (14). | | The reporting system should still be able to generate reports based off a specific caretaker. | FAIL  The report was not able to be generated and the program crashed. | |  | |

#### Unit Testing

Test 1: Retrieve all caretakers from the system database

Code:

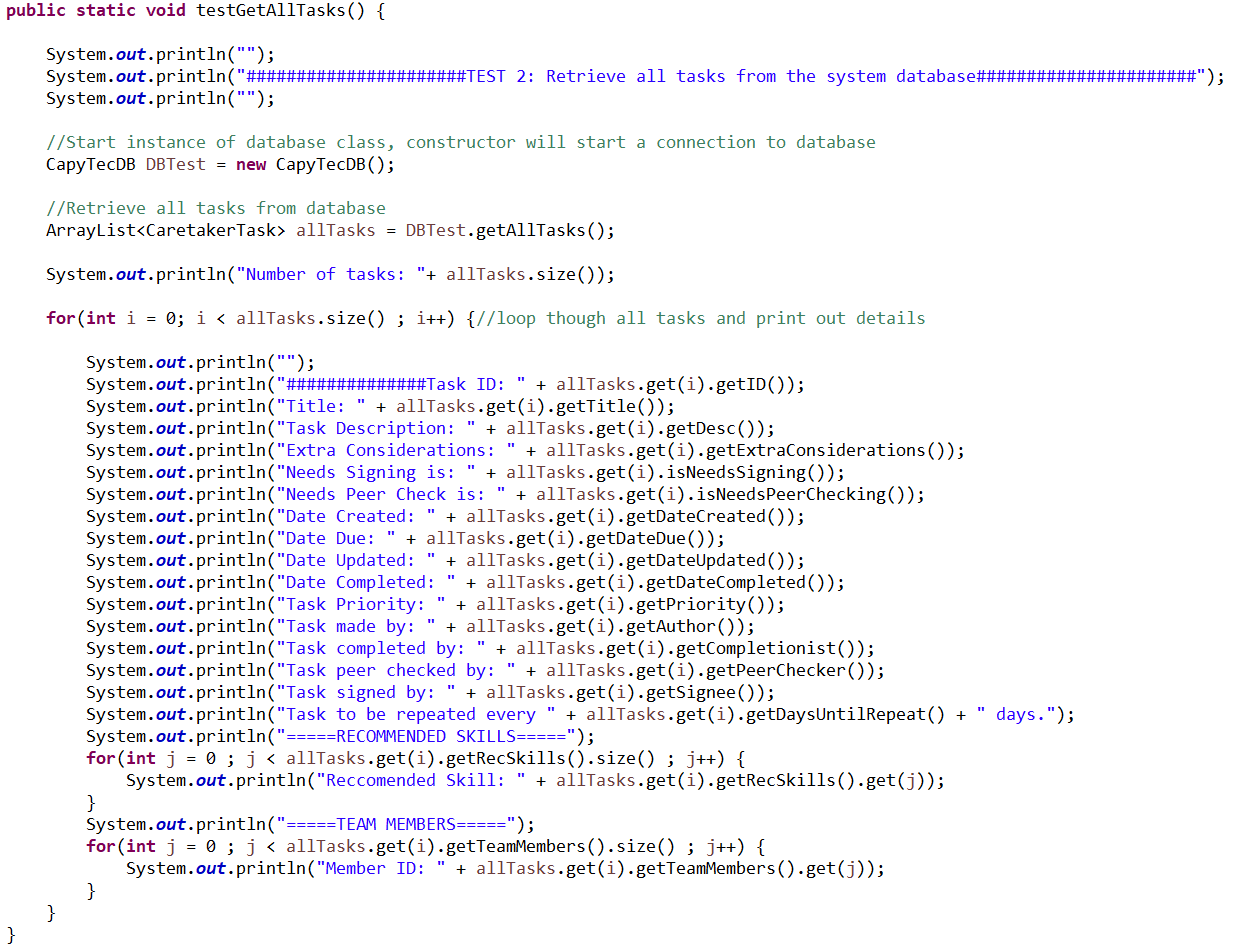


Console Output:



Test 2: Output all tasks from the system database

Code:

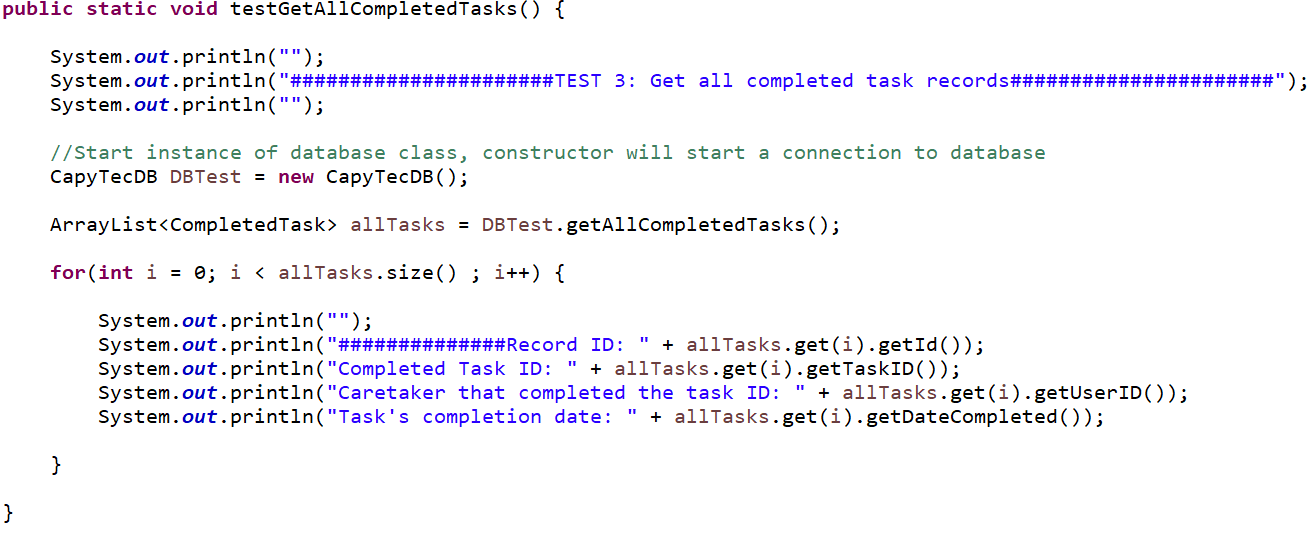


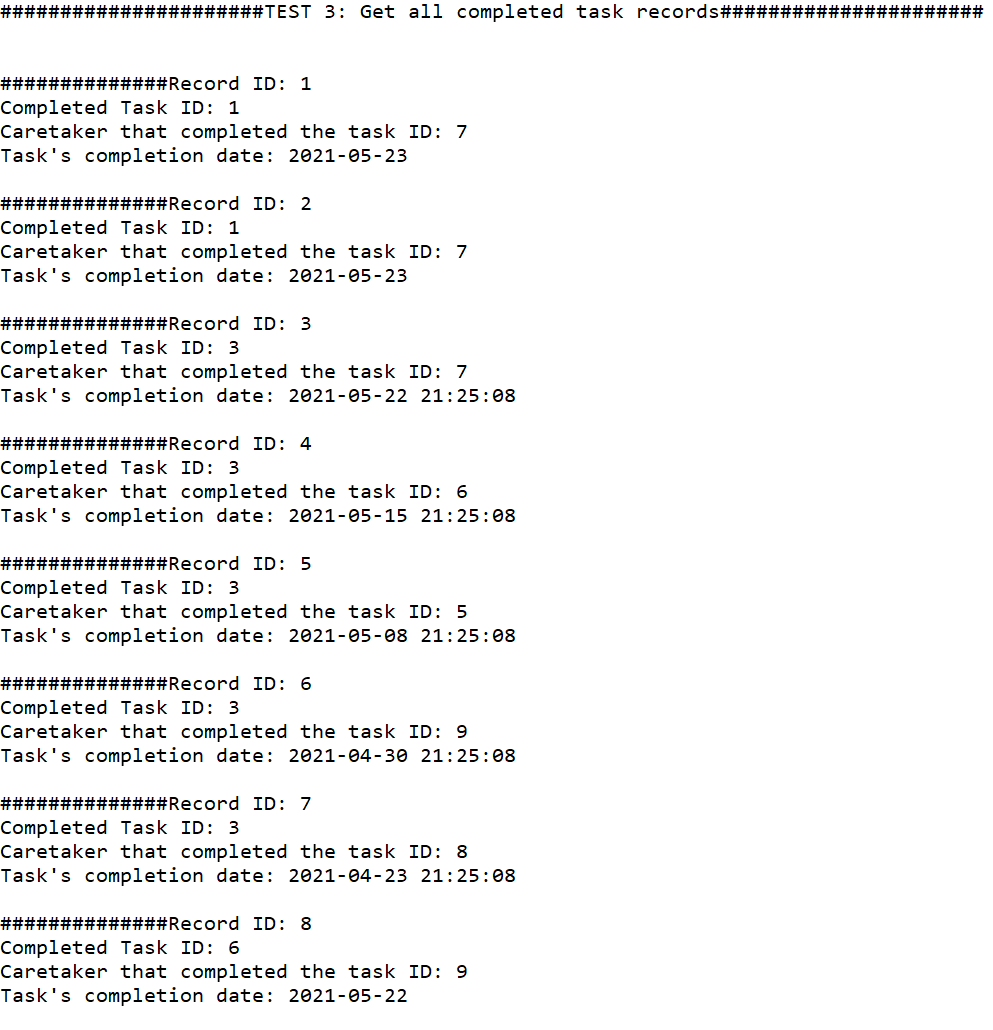
Console output:



Test 3: Get all completed task records from system database

Code:

Console output:



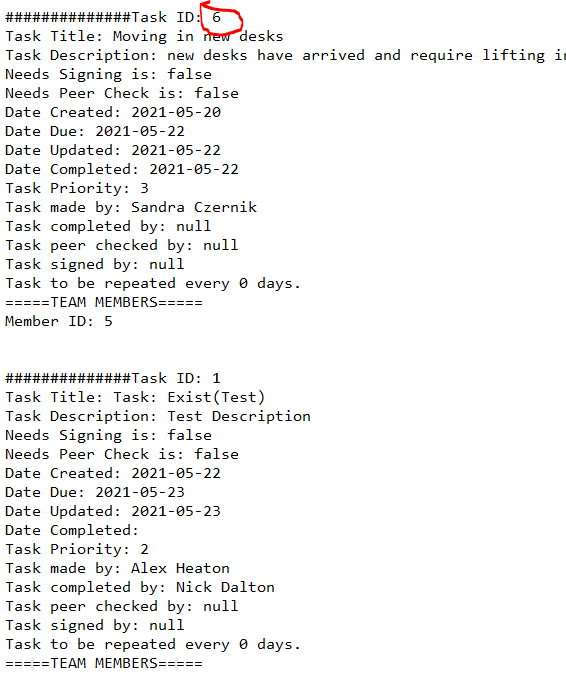
Test 4: Insert a task into the system database then delete it

Code:

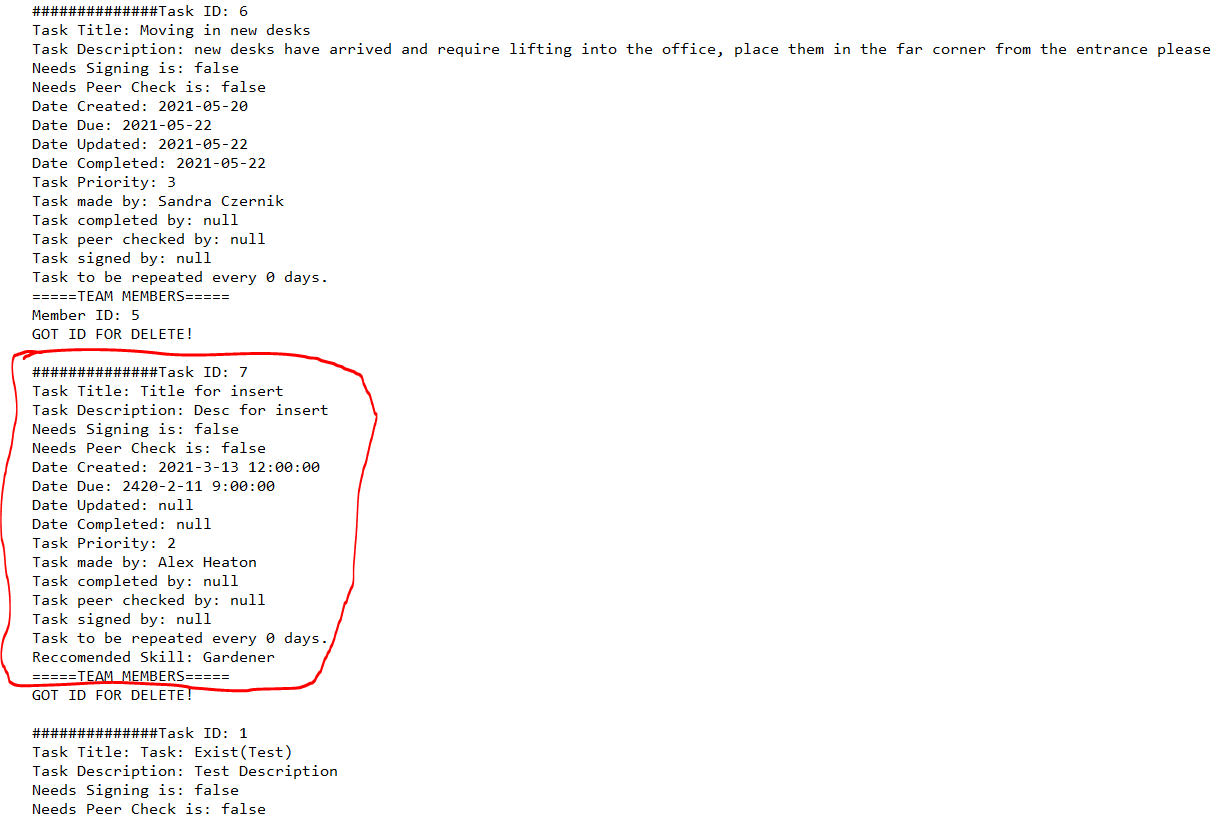


Console Output:

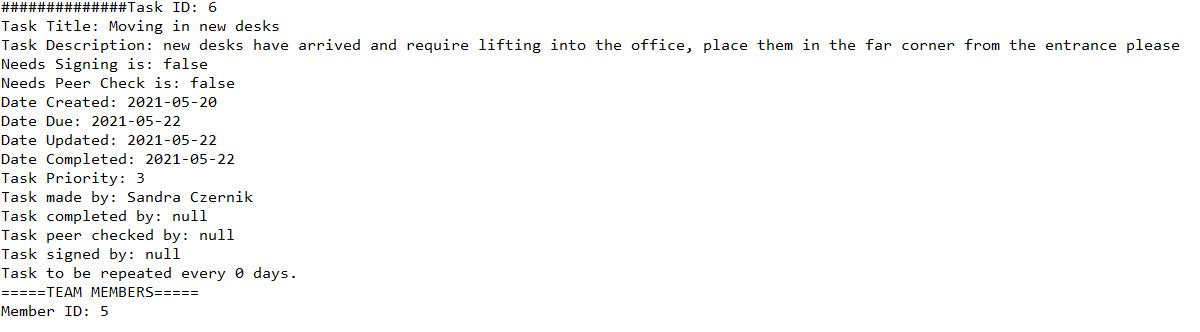
1. Records before insert end at id 6



1. Upon second reading of the tasks the newly inserted task is visible

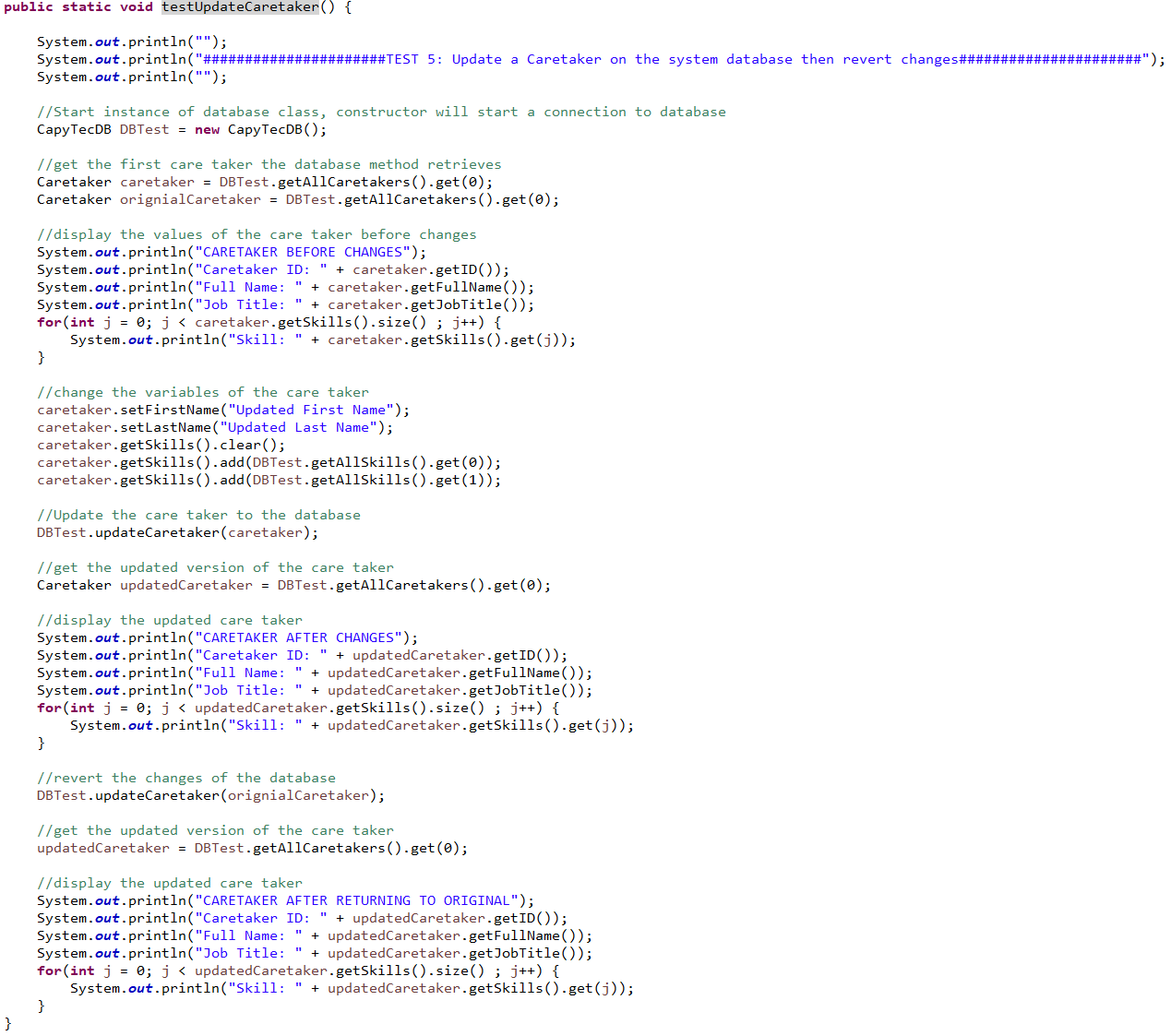


1. When the third reading comes in the 7th task is gone. List ends at 6

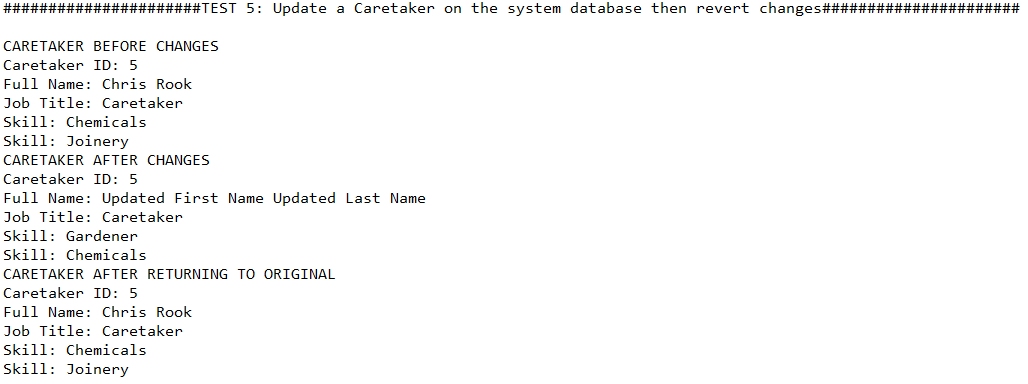


Test 5: Update a caretaker on the system database then revert changes

Code:



Console Output:

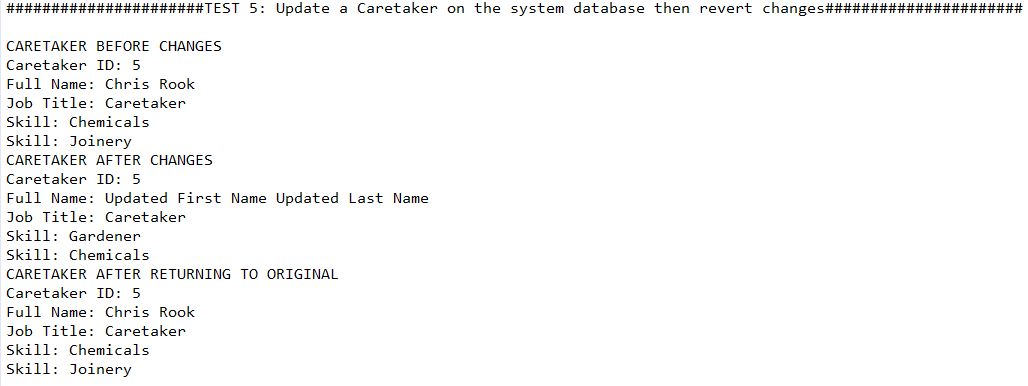


Test 6: Update a task on the system database then revert changes

Code:



Console Output:

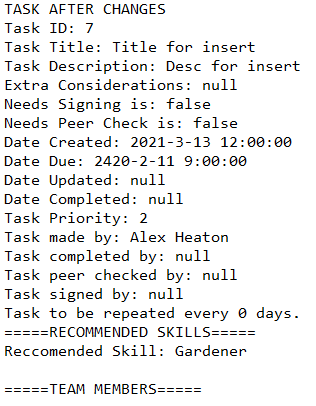


Test 7: Insert new task with multiple methods

Code:



Console output:



#### Bug Reports

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Defect ID | 1 | Product Version | Prototype | Test ID | | 41 |
| Defect description | | **Date raised** | **23/05/21** | **Reporter** | | **Sam Farnworth** |
| Database manipulation fails when program is ran through the CapytecMain class rather than the CapytecGui class. Database is locked error. | | **Date closed** | **-** | **Fixed by** | | **-** |
|  | | **Severity** | | | | |
| Steps to reproduce | | **Critical** | **Major** | | **Minor** | |
| Use a feature within the application that manipulates the database (Task addition, deletion etc), whilst the program has been launched through the CapytecMain class. | | **Priority** | | | | |
|  | | **High** | **Medium** | | **Low** | |
|  | | **Status**  **Date** | **Open** | **Assigned** | **Retest** | **Closed** |
|  | |  | **23/05/21** |  |  |

This bug is a critical issue which prevents database manipulation from taking place, severely limiting the functionality of the application when it occurs. This error occurs when the program is launched through the CapytecMain class and the login system is enabled, and can be avoided by running the program through the CapytecGui class, although the login system should be disabled within the application, and the userID and permission will have to be manually modified within the code rather than dynamically through the login system as a result. A potential fix for this issue would be to implement the database as a singleton, therefore preventing multiple SQLite instances running simultaneously and preventing the database from being locked, as SQLite only supports a single instance being modified at any one time.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Defect ID | 2 | Product Version | Prototype | Test ID | | 42 |
| Defect description | | **Date raised** | **24/05/21** | **Reporter** | | **Kieran Robinson** |
| When a task is inserted with multiple types, only the first type is inserted into the database. | | **Date closed** | **24/05/21** | **Fixed by** | | **Alex Heaton** |
|  | | **Severity** | | | | |
| Steps to reproduce | | **Critical** | **Major** | | **Minor** | |
| Insert a task with more than one type selected. Task will be added to the database and be visible within the task management table, although only the first task type will be added. | | **Priority** | | | | |
|  | | **High** | **Medium** | | **Low** | |
|  | | **Status**  **Date** | **Open** | **Assigned** | **Retest** | **Closed** |
|  | | **24/05/2021** |  |  |  |

This bug is a major issue that limits the functionality of the application by limiting the ability for a user to insert a task with multiple types. In order to determine whether the issue was on the GUI or database side of the program, both white box and black box testing was utilised in order to determine where the issue was located. It was determined that the issue was with the insert task method within the database functions rather than the GUI through unit testing. The unit test for this issue is test 7. This bug was fixed though the correct handling of ids for the task skill inserts to the task\_skill table in the system database. Both the task id and the skill id required fixing as both inputs were incorrect. This was fixed by assigning the correct task and skill id to the table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Defect ID | 3 | Product Version | Prototype | Test ID | | 43 |
| Defect description | | **Date raised** | **24/05/21** | **Reporter** | | **Sam Farnworth** |
| When a report based off of a caretaker is generated, the report cannot be made and the program crashes. | | **Date closed** | **24/05/21** | **Fixed by** | | **Sam Farnworth** |
|  | | **Severity** | | | | |
| Steps to reproduce | | **Critical** | **Major** | | **Minor** | |
| Press the generate report button while on the “Caretaker History” part of reports with a caretaker selected. | | **Priority** | | | | |
|  | | **High** | **Medium** | | **Low** | |
|  | | **Status**  **Date** | **Open** | **Assigned** | **Retest** | **Closed** |
|  | | **24/05/2021** |  |  |  |

This big is a major issue that may not limit other sections of the application, however it does cause a critical failure within the program causing it to crash. Fortunately, this issue was able to be fixed. The cause behind this issue was due to the combo box (dropdown box) item being changed to a string type, comparison could no longer take place while using a “==” comparison, and instead any comparisons had to be replaced with a .equals() comparison. After making this change, the program performed correctly and no longer crashed.

## **Mission 6**

### *Entity relationship diagram*

In order to document the data component that being an SQLite database an entity relationship diagram (ERD) was created showing the makeup of the relational model. And ERD was chosen as it provides a clear and concise view of the overall database. Each table displays the columns of the database with some details such as the column names, what keys the columns may be and what data type they are.

The ERD also presents the relations between the tables providing a sense of multiplicity as only one user can be on a team record however a user can have many team records as they may be carrying out multiple tasks. Composite keys are used for 3 tables that being team, user\_skill and task\_skill. This is so tasks, users and skill can remain as an individual record without redundancy since a user and a task can only pick from the same list of skills the information on skills would be made redundant as it would be in two tables there fore two table one between user and skill and task and skill is used. And since the database can allow for multiple users to do one task there is also a team table.

Diagram

Description automatically generated

Figure 47 Entity relationship diagram.