

Customer Care System

CS4125 - Systems Analysis and Design



ABC – 0123457

XYZ – 0123456

Table of Contents

1. Narrative Description of Project	4
2. Software Lifecycle Model	6
2.1 Rational Unified Process	6
2.2 Agile Modelling	9
3. Project Plan and Allocation of Roles	11
4. Discussion of System Architecture	12
4.1 Package Diagram showing a three tier structure:	13
4.2 UML Workbench	14
5. Requirements	15
5.1 Functional requirements	15
5.2 Non Functional Requirements	23
5.3 Discussion on Tactics to Support Specific Quality Attributes	24
5.4 Screen Shots	25
5.4.1 Create Ticket	25
5.4.2 Ticket Report Custom View	27
5.5 Layouts/Format for Reports	28
6. Analysis	29
6.1 List of Candidate Classes	29
6.2 Applying Heuristics to Eliminate Poor Candidates	30
6.3 Leftover Entity Classes	30
6.4 Class Diagram	31
6.5 Sequence Interaction Diagram – Change Status	37
6.6 Communication Diagram – Create Ticket	38
6.7 State Chart	38
6.8 Entity Relationship Diagram	40
7. Design	41
7.1 Description of Architectural Style	41
7.1.1 The State Pattern	41
7.1.2 The Observer Pattern	41
7.2 Class Diagram Including MVC	42
7.3 Sequence Diagram Including MVC	43
7.4 Concurrency in Design Time Diagrams	45
8. Technical Architecture Diagram	46
9. Data Dictionary	48
10. Critique of Design Quality	52
11. References	53

CS4125: Systems Analysis

Assignment 1: Semester II, 2012-2013

TEAM SUBMISSION : MARKING SCHEME:Version 2 (25th February 2013)

Name:			ID:		
Name:			ID:		
	Item	Detailed Description	Marks Allocated		Marks Awarded
			Sub-total	Total	
	Presentation	<ul style="list-style-type: none">General PresentationAdherence to guidelines i.e front cover sheet, blank marking scheme, table of contents		2	
4	Narrative	Narrative description of business scenario		1	
5	Software Lifecycle	Is it linear (Waterfall) or iterative (RUP). Discuss risk management strategy?		2	
6	Project Plan	Plan specifying timeline, deliverables, and roles.		1	
7	System Architecture	System architecture diagram with interfaces		3	
8	Requirement	<ul style="list-style-type: none">Use case diagram(s)Structured use case descriptions(s)Non-functional (quality) attributesScreen shots / report formats	1 2 2 1	6	
9	Analysis	<ul style="list-style-type: none">Method used to identify candidate classesClass diagram with generalisation, composition, multiplicity, dialog, control, entity, interface classes, etc.Communication diagramSequence diagramState chart with annotated transition stringsEntity relationship diagram with cardinality	1 3 1 1 1 1	8	
10	Design	<ul style="list-style-type: none">Description of an architectural or design pattern that was evaluated. Cannot use MVC, Broker and Singleton.Pattern incorporated into class diagramRefinement of class diagram from analysis to include MVC architectural pattern.Refined interaction diagram from analysis with MVC elementsSupport for concurrency	2 2 2 1 1	8	
11	Technical Architecture			1	
12	Data dictionary	Fragments to illustrate different artefacts created during requirements, analysis and design.		1	
13	Critique	Assess the quality of the design submitted		2	
14	References			1	
15	Online Assessment	Week 7: Use cases Week 9: Class diagram	2 2	4	
16	Interview Week 13 (Pass/Fail basis)			P/F	
	TOTAL			40	

1. Narrative Description of Project

Tech Ltd wants to create a troubleshooting system for its employees to log both internal system issues, including both hardware and software issues, and customer issues. Each staff member will have a personal login in order to create events in the system, hereby known as tickets. They want the system to have 3 methods of interaction: a Windows/Linux desktop based application (both systems are used in the workplace), a mobile based interface and a web based interface for those away from the office. The mobile based/web based are need only provide ticket functionality and there is no need for reports away from the office.

A ticket will have 5 possible states:

- **New** – This will send an email to the relevant department head
- **Assigned** – The ticket has been assigned to the relevant member of staff and is ready to be set to the “Work” state.
- **Work** – The ticket is actively being worked on by a member of staff.
- **Request More Info** – This will send an email to the ticket author from the staff member assigned to it.
- **Closed** – This will send an email to the author, supervisor and staff member who was assigned.

A ticket is made up of an author, a category and/or subcategory, a summary of the problem area, and a description which should be no less than 512 characters this will to ensure that a description is long enough but not too short as to require a follow up. The severity of the ticket is judged by the category selected (e.g. having no network coverage would be judged as requiring more immediate action than not receiving texts intermittently).

The below categories are required:

- Customer Issue <Sub Categories: Billing, Technical>
- Internal Software Issues
- Internal Hardware Issue
- If there is no sub category, the option will remain greyed out.

The following Categories and Sub Categories should be added by default, with the System Administrator being able to add more.

Customer <Billing>	Customer <Technical>	Internal Software	Internal Hardware
Overcharge	Cannot make/rec calls	Install Software	Faulty Hardware
Change bill date	Cannot make/rec texts	Update Software	Need hardware upgrade
Change payment method	No signal	Not Connecting to Internet	
Cancel Payment/Account	Intermittent signal		
Change Customer Plan	Faulty hardware		
	Faulty Software		

Any member of staff can create a ticket. They must login with their supplied details.

Step 1: Login

Step 2: Choose Category

Step 2.1: Choose Subcategory

Step 3: Enter Summary

Step 4: Enter Description

Step 5: Send

The Ticket Management system auto assigns tickets with a queue system based on who has the least workload, how severe the ticket is and who is suitable to work on the ticket. If a supervisor wishes they can override the choice the ticket management system made and reassign the ticket manually. When a ticket is pending, the person responsible will have a choice of 4 actions in dealing with the ticket:

Work: The user sets this status when they are actively working on a ticket. When the user changes from work to another status they must enter time worked on the ticket. Time entered is used for reports and compared against the time estimated so the company can make more accurate estimates in future.

Assigned: The ticket has been assigned to the relevant member of staff and is ready to be set to the "Work" state. The ticket is in their backlog which they can view when they go to the view tickets section.

Request More Info: This places the ticket on hold and sends an email to the author requesting more information on the problem. If there is no response within 72 hours of the change in status, the ticket is automatically closed and an email is sent to the author and person responsible. Tickets cannot be reopened.

Closed: The issue is either fixed, unfixable or is a duplicate of a previous request. As such, no more can be done with the issue at this present time. A reason for closing can be recorded in the comments section of the ticket.

2. Software Lifecycle Model

2.1 Rational Unified Process

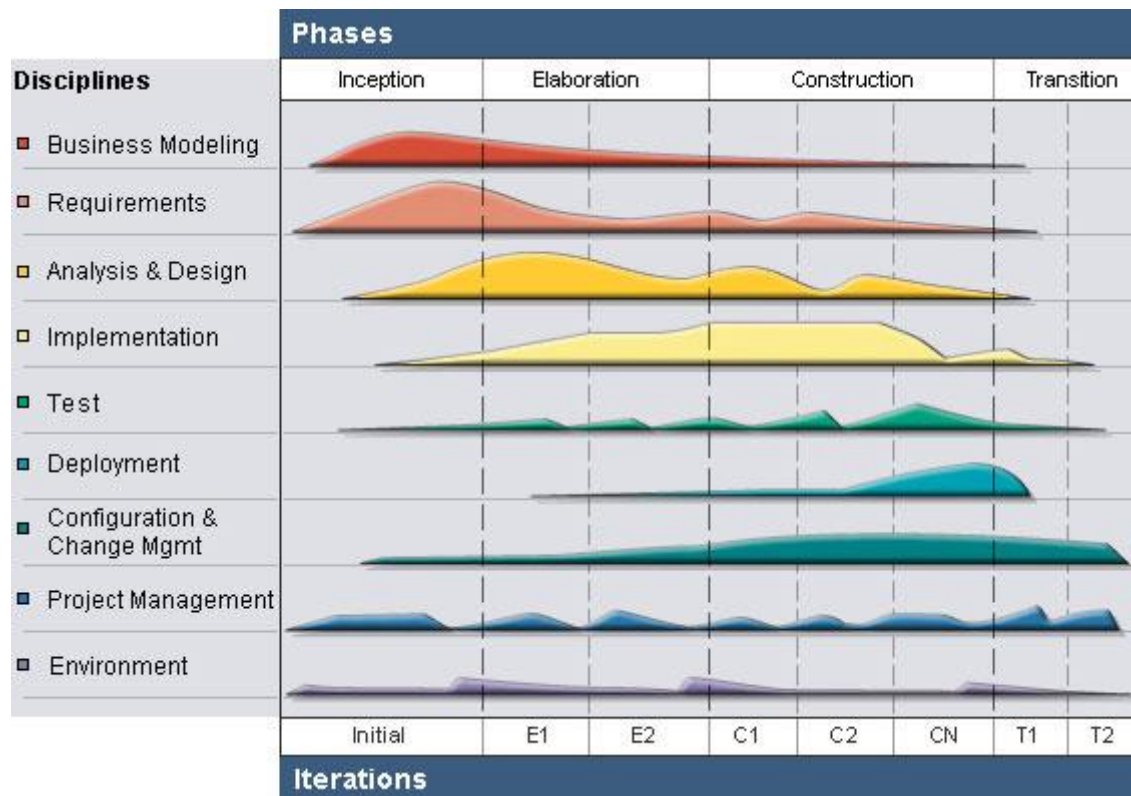


Fig 1. Diagram of the RUP Lifecycle Model

One process that would aid development of this system is RUP (Rational Unified Process). The best practices for this process are

Develop iteratively, with risk as primary iteration driver

Manage requirements

Employ a component based architecture

Model software visually

Continuously verify quality

Control changes

The RUP authors identified a tendency for teams to address the easiest or best known aspects of a project first, in order to get the project moving forward. While a sound idea in theory, it pushes the riskier and less well known elements to the end of the project. These issues should have been given more time in the initial phase, and the failure to acknowledge them can lead to delays in development and increased costs.

RUP is divided into four phases to distinguish between the different focus on activities at different times during the development lifecycle.

Phase	Inception	Elaboration	Construction	Transition
Key Question	Should we build it?	Can we build it?	Are we building it?	Have we delivered it?
Focus	Scope	Risk	Functionality	Delivery

1. Inception

The primary objective of the Inception phase is to validate the cost and budget of the project scope. During this phase, use case models, project plans, risk assessments and the project description are generated. After these are completed, the project is checked against the following criteria.

- Stakeholder concurrence on scope definition and cost schedules
- Requirements Understanding (evidenced by fidelity of primary use cases)
- Credibility of costs, priorities, risks, and development process
- Establishing a baseline to compare actual expenditures versus estimated
- Depth and breadth of architectural prototype used.

If the project does not pass this milestone, it either can be cancelled or repeated after redesign.

2. Elaboration

The primary objective of the Elaboration phase is to mitigate the key risk items identified by analysis up to the end of this phase. This is where the project starts to take shape, the problem domain analysis is made and the architecture of the project gets its basic form. The outcome of the elaboration phase is:

- A use case model in which use cases and actors have been identified and most use case descriptions developed. Approximately 80% completion expected
- A description of the software architecture in a software system development process
- Business case and risk list, which are revised
- Development plan for the overall project
- Prototypes that demonstrably mitigate each identified technical risk

This phase must pass the Lifecycle Architecture Milestone criteria, which is done by answering the following questions.

- Is the vision of the product stable?
- Is the architecture stable?
- Is the construction phase plan sufficiently detailed?
- Is actual expenditure vs. planned expenditure acceptable?
- Do all stakeholders agree that the current vision can be achieved using current plan?
- Does the demonstration indicate that all major risks identified are mitigated?

3. Construction

The primary objective of this phase is to build the software system. The main focus is the development of components and other features of the system. This is where the bulk of the coding takes place. In larger projects, several iterations may be developed in an effort to divide the use cases into manageable segments that produce demonstrable prototypes. Generally, this phase produces the first external release of the software, and it concludes with the Initial Operational Capability Milestone.

- Is the beta product ready to be handed over to the users?
- Has all functionality been developed and sufficient alpha testing completed?
- Have you developed a user manual and a description of the current release?
- Is actual expenditure compared to planned expenditure acceptable?

4. Transition

The primary objective of this phase is to 'transit' the system from development into production, making it available to and understood by the end user. Activities in this phase include training end users and maintainers, beta testing to validate the system against end user expectations. The product is also checked against the quality level provided during the inception phase. If all objectives are met, the Product Release Milestone is reached and development cycle is finished. This is reached by asking the following questions:

- Is the user satisfied?
- Are actual expenditures versus planned expenditures acceptable?

RUP Best Practices:

1. Develop iteratively

It is best to know all requirements in advance, though often, this is not the case. Several software developments processes exist that deal with providing solutions on how to minimise costs in terms of development phases.

2. Manage Requirements

Always keep in mind the requirements set by the users

3. Use components

Breaking down an advanced project is not only recommended, it's often unavoidable. This promotes the ability to test individual components before they are integrated into a larger system. Code reuse, through the Object Oriented paradigm is a big plus.

4. Model visually

Use diagrams to represent all major components, users and their interactions. For example, UML.

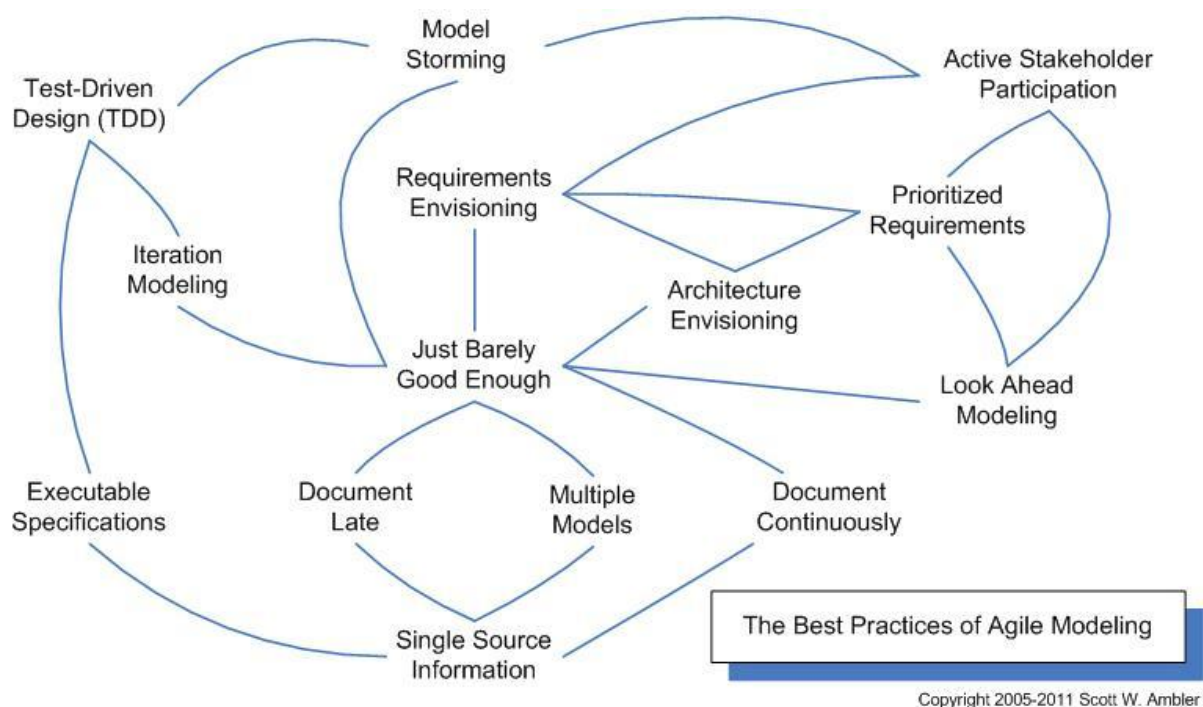
5. Verify quality

Always make testing a major part of the project at any point in time. Testing becomes heavier as the project progresses but earlier testing will find bugs when the system is less complex and will result in cost savings and easier maintainability.

6. Control changes

Many projects are created by many teams, sometimes in various locations, with different platforms etc. As such, it is essential to make sure that changes are synchronised and verified constantly.

2.2 Agile Modelling



Agile Modelling is a methodology for modelling and documentation of a software based system. It can be used in conjunction with practices such as Extreme Programming, Disciplined Agile Delivery and Scrum.

1. Active Stakeholder Participation: Stakeholders should provide information in a timely manner, make decisions in a timely manner, and be as actively involved in the development process through the use of **inclusive tools and techniques**.

2. Architecture Envisioning: At the beginning of an agile project you will need to do some initial, high-level architectural modelling to identify a viable technical strategy for your solution.

- 3. Document Continuously:** Write deliverable documentation throughout the lifecycle in parallel to the creation of the rest of the solution.
- 4. Document Late:** Write deliverable documentation as late as possible, avoiding speculative ideas that are likely to change in favour of stable information.
- 5. Executable Specifications:** Specify requirements in the form of executable "**customer tests**", and your design as executable developer tests, instead of non-executable "static" documentation.
- 6. Iteration Modelling:** At the beginning of each iteration you will do a bit of modelling as part of your iteration planning activities.
- 7. Just Barely Good Enough (JBGE) artefacts:** A model or document needs to be sufficient for the situation at hand and no more.
- 8. Look Ahead Modelling:** Sometimes requirements that are nearing the top of your **priority stack** are fairly complex, motivating you to invest some effort to explore them before they're popped off the top of the work item stack so as to reduce overall risk.
- 9. Model Storming:** Throughout an iteration you will model storm on a just-in-time (JIT) basis for a few minutes to explore the details behind a requirement or to think through a design issue.
- 10. Multiple Models:** Each type of model has its strengths and weaknesses. An effective developer will need a range of models in their intellectual toolkit enabling them to apply the right model in the most appropriate manner for the situation at hand.
- 11. Prioritized Requirements:** Agile teams implement requirements in priority order, as defined by their stakeholders, so as to provide the greatest return on investment (ROI) possible.
- 12. Requirements Envisioning:** At the beginning of an agile project you will need to invest some time to identify the scope of the project and to create the initial **prioritized stack of requirements**.
- 13. Single Source Information:** Strive to capture information in one place and one place only.
- 14. Test-Driven Design (TDD):** Write a single test, either at the requirements or design level, and then just enough code to fulfil that test. TDD is a JIT approach to detailed requirements specification and a confirmatory approach to testing.

3. Project Plan and Allocation of Roles

Narrative Description of Project Week 04 - ABC/XYZ

Software Lifecycle Model Week 08 - ABC

Discussion of System Architecture Week 12 - ABC/XYZ

Requirements:

- | | |
|---|-------------------|
| 1. Functional Requirements | Week 08 - ABC/XYZ |
| 2. Non Functional Requirements | Week 08 - XYZ |
| 3. Discussion on Tactics to Support Specific Quality Attributes | Week 08 - XYZ |
| 4. Screen Shots | Week 09 - XYZ |
| 5. Layouts/Format for both Electronic Hardcopy formats | Week 09 - XYZ |

Analysis:

- | | |
|---------------------------------|-------------------|
| 1. List of Candidate Classes | Week 09 - XYZ |
| 2. Class Diagram | Week 09 - ABC/XYZ |
| 3. Sequence Interaction Diagram | Week 10 - ABC |
| 4. Communication Diagram | Week 10 - XYZ |
| 5. State Chart | Week 10 - ABC |
| 6. Entity Relationship Diagram | Week 11 - XYZ |

Design:

- | | |
|--|-------------------|
| 1. Description of Architectural Style | Week 12 - ABC |
| 2. Class Diagram (Including MVC) | Week 12 – ABC/XYZ |
| 3. Sequence Diagram (Including MVC) | Week 12 – ABC/XYZ |
| 4. Concurrency in Design Time Diagrams | Week 12 - XYZ |

Technical Architecture Diagram Week 12 - ABC

Data Dictionary Week 12 - ABC/XYZ

Critique of Design Quality Week 12 - ABC

References

4. Discussion of System Architecture

We decided that in order to make the system as portable as possible, we needed to use Java as the primary language, as due to the nature of the JVM, it runs on pretty much any system with minimum, if any, changes. This was important to the client who is running both Windows and UNIX based machines, and in the event that they acquire an OSX machine, it would be fairly straightforward to get it running.

We have chosen a Client-Server Architecture for this project. The client is going to be hosted on the computers/devices that are used by the company in their day to day business. The local application running on these devices or website interface running on a web browser is the client side application. As alluded to earlier, there are a variety of devices used, such as, but not limited to, Windows, UNIX, Android, iOS etc. The Server Architecture that we have chosen is Java Enterprise NetBeans, due to our (limited!) familiarity with it from the Distributed Systems module

There are 3 layers in our Application. The User Interface layer contains all the UI options for the application, and also enables us to develop extra interfaces without the need to change any of the layers below. This is especially important as the devices used (Linux, Windows, Android, iOS etc) change regularly and it's important that any changes can be made quickly, easily and without having to look at the business logic on the system.

The next layer is the business logic, and consists of the TicketManagement, ReportManagement, Staff and Ticket packages. This is the core of the application, and handles the information exchange between the previous layer, the UI, and the next layer, the Database/Storage Layer. The TicketManagement system is the core of the application and is responsible for maintaining the business logic of the system. The Business logic is hosted on the GlassFish Application Server.

The Database layer interacts with the business logic by both feeding data to it, and storing data created and changed by the business layer. This data is stored in persistent storage, and in the case of this project, would be managed using an entity-relational database which is easy to manipulate and secure using Java Enterprise Edition.

The UI is the main example of Programming to Interfaces. We have one main UI, but we can have 3 different ways of accessing it, plus we also want to limit what functions can be accessed from within those interfaces. For example, while the Web and Mobile interfaces would be quite similar, the Mobile Interface would load lower resolution images and streamline them in order to give a faster experience on a potentially slower connection, in the same way that the RTE website offers <http://m.rte.ie> for an alternative for <http://www.rte.ie> but with similar content. Both these interfaces will not have the full functionality of the main UI Interface, but both can also be potentially expanded also.

4.1 Package Diagram showing a three tier structure:

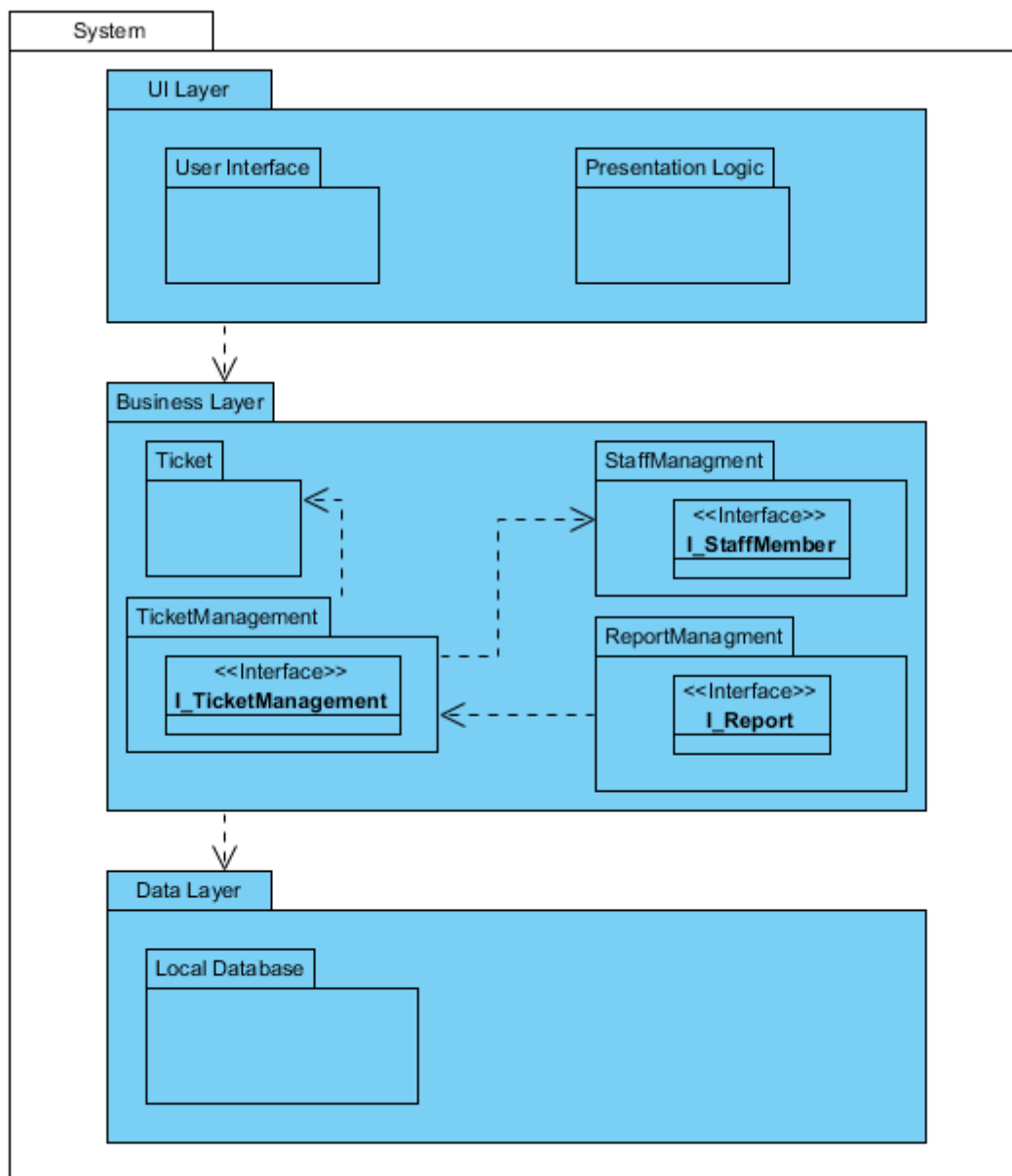


Fig: Package Diagram for the Ticket System

4.2 UML Workbench

The UML Workbench we chose was Visual Paradigm for UML version 10.1. Visual Paradigm (VP) has a community edition that can be used by Students and the Open Source community which is for non-commercial use only. VP is a really nice tool set and the community edition doesn't use a Client-Server architecture used in Select Architect. The implementation of Select Architect is cumbersome for a small team to work with. With our limited resources we were able to work quickly and fluidly. The (.vpp) project file was synchronised in a shared Dropbox folder between team members and all team members were kept up to date with the project. VP uses similar language to Integrated Development Environments (IDE's) such as Eclipse and can even integrate with Eclipse, Netbeans and Visual Studio. The rendering quality of diagrams is far superior to Select Architect with no Anti-Aliasing seen on the diagrams. VP can also be extended with community developed plugins. The only disadvantage VP has is that when you export a file as an image it is covered in watermarks, using a screen capture utility we got around this issue and the screen capped version looked better than the rendered images from VP anyway. Overall our team would recommend Visual Paradigm.

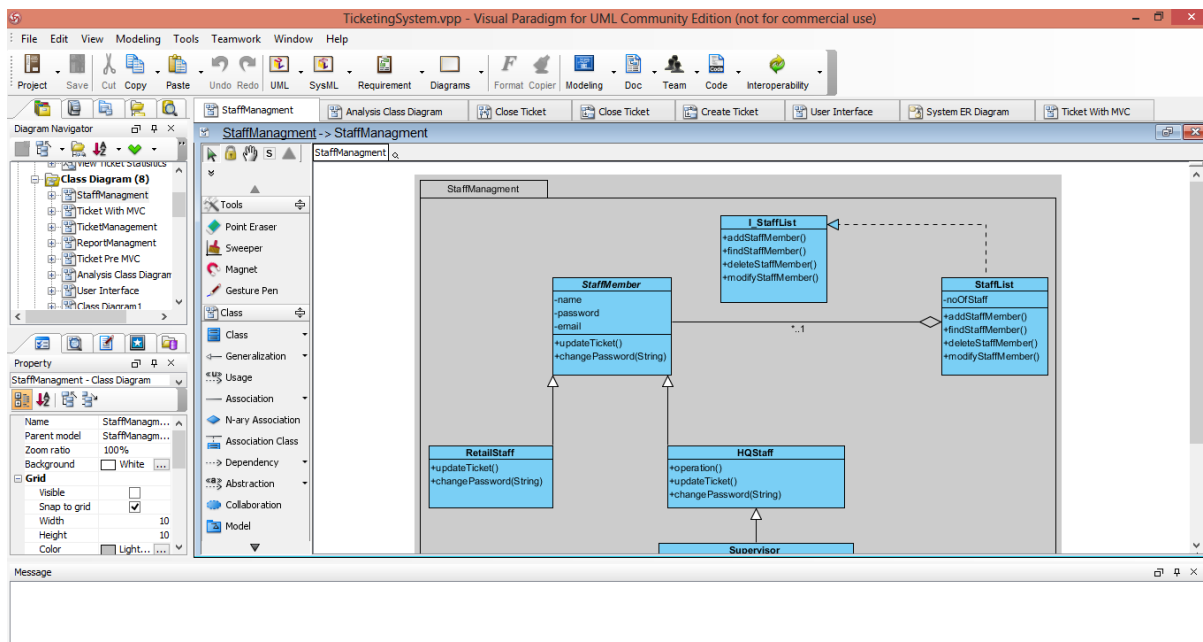


Fig. Screen shot of Visual Paradigm 10.1 in relation to our project

5. Requirements

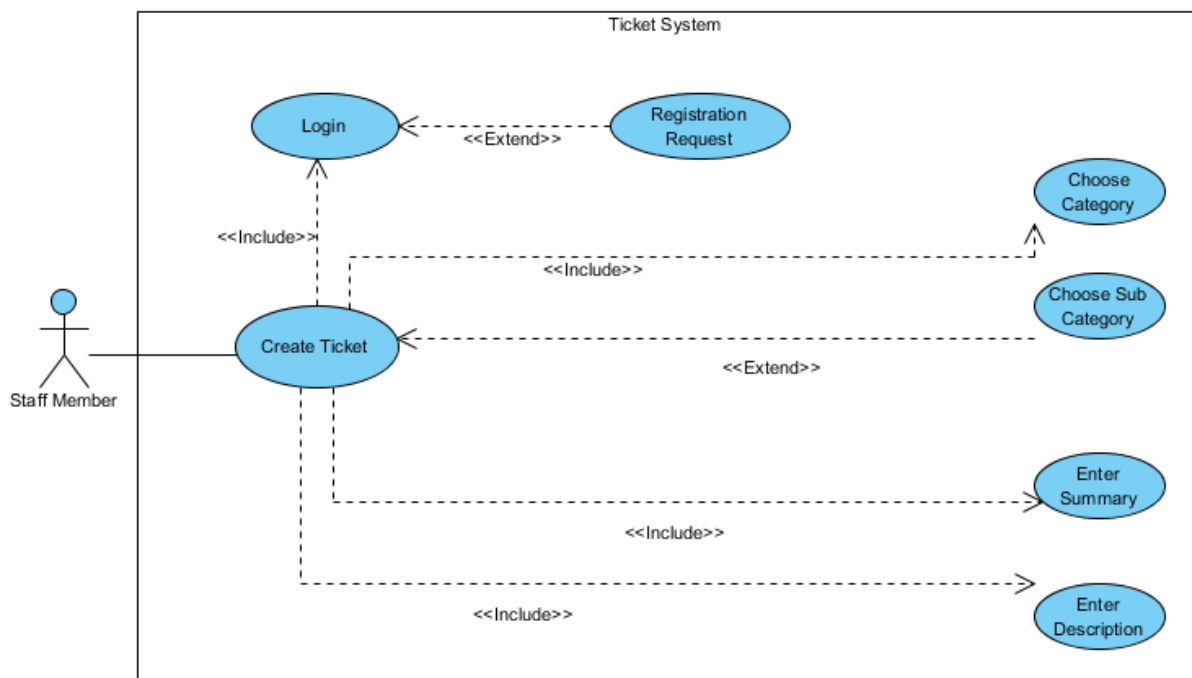
5.1 Functional requirements

Use Case Descriptions:

- Create Ticket
- Search Tickets
- Update Ticket
- View Report
- Assign Tickets
- Create User

USE CASE	Create Ticket	
Goal in Context	User wants to create a ticket to document a support issue.	
Scope & Level	Primary Task	
Preconditions	The user is an employee of company. The user has permission to log a ticket. System is available.	
Success End Condition	The user has logged a ticket. The relevant people have been notified. The ticket is entered into the relevant department for work.	
Failed End Condition	The ticket has not been logged. Remove any reference of ticket from system.	
Primary, Secondary, Actors	Retail Employee, HQ Employee	
Trigger	Select "Create a Ticket"	
DESCRIPTION	Step	Action
	1	Employee starts new ticket
	2	Verification of Employee permission
	3	Check that people are available to deal with ticket
	4	Choose Sub Category
	5	Enter Summary
	6	Enter Description
	7	Upload image
	8	Upload file (Log Files, other files)
EXTENSIONS	Step	Branching Action
VARIATIONS		Branching Action

RELATED INFORMATION	Create Ticket
Priority	Critical – Crux of customer care system
Performance	150ms response from system
Frequency	100 times per day in worst case
Channels to Actors	Database, Auto assign ticket system
OPEN ISSUES	-
Due Date	Alpha Release 1.0
...any other management information...	-
Superordinate's	Login, View Tickets, View Report
Subordinates	-



USE CASE	Search Tickets	
Goal in Context	User wants to view tickets that have been created	
Scope & Level	Primary Task	
Preconditions	The user is an employee of company. Tickets have been previously created	
Success End Condition	A list of chosen tickets appears on screen	
Failed End Condition	No tickets available with selected criteria	
Primary, Secondary, Actors	Retail Employee, HQ Employee	
Trigger	Search for Tickets field entered	
DESCRIPTION	Step	Action
	1	Employee logs in, selects search tickets
	2	Employee enters search criteria
	3	All tickets matching search criteria are displayed
EXTENSIONS	Step	Branching Action
	2.1	Refine Search
VARIATIONS		Branching Action
	2	Tech/Care Staff Click search my tickets– All tickets assigned to and created by user are displayed

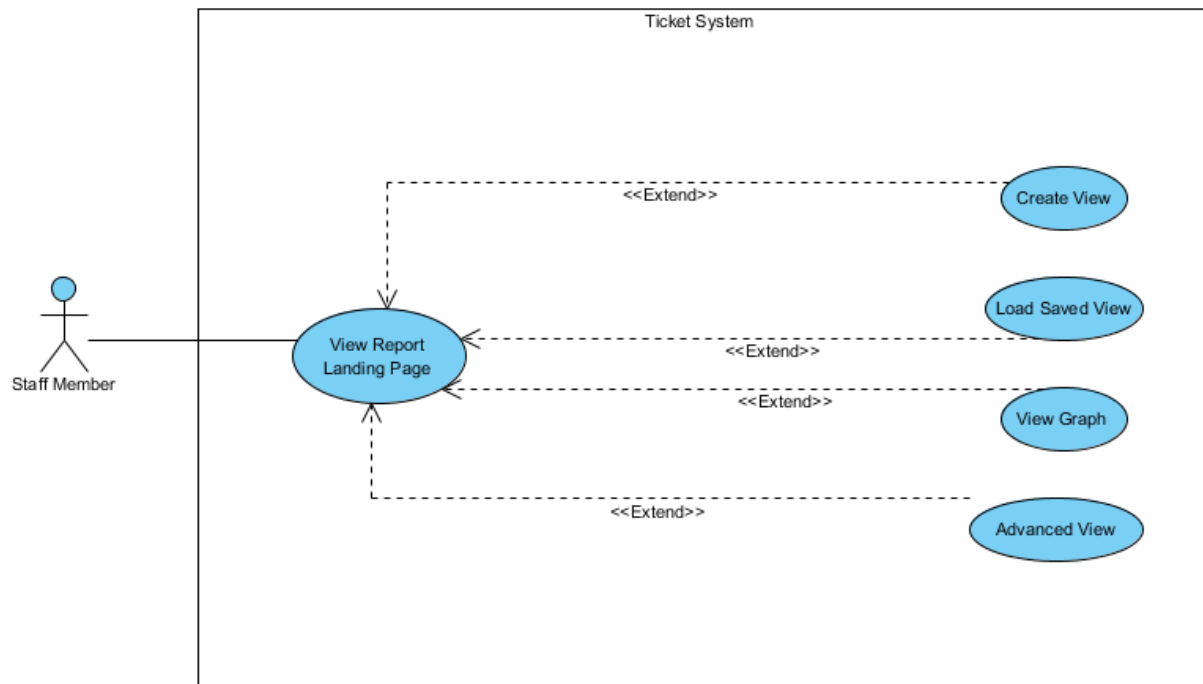
RELATED INFORMATION	Search Tickets
Priority	Important – User needs to find tickets quickly
Performance	500ms response time – higher response times are allowed as lots of data is being returned
Frequency	3-4 times a day for regular use 50-100 times a day for care/tech staff
Channels to Actors	Database
OPEN ISSUES	-
Due Date	Alpha Release 1.0
...any other management information...	-
Superordinate's	Login, View Report, Create Ticket
Subordinates	-

USE CASE	Update Ticket	
Goal in Context	User wants to edit the description of the ticket	
Scope & Level	Secondary Task Less Frequent.	
Preconditions	The user is an employee of company. The user is the author of the ticket or the user has been assigned to the ticket.	
Success End Condition	The user updates the ticket. The relevant people have been notified. The ticket database has been updated.	
Failed End Condition	The ticket has not been updated. Prior state of ticket remains and database not updated.	
Primary, Secondary, Actors	Retail Employee, HQ Employee	
Trigger	User is viewing a ticket and then decides to update it. A request for more information has been served to the user via the notification system and they decide to provide information.	
DESCRIPTION	Step	Action
	1	Employee logs in.
	2	User needs to update ticket
	2	User clicks on ticket, ticket information is displayed
	3	User clicks on update ticket, a text dialog appears
	4	User enters information required
	5	User clicks submit, ticket is updated, user returned to ticket description
EXTENSIONS	Step	Branching Action
	4.1	Option to upload file (jpg, txt, etc.)
VARIATIONS		Branching Action
	2.1	The user decides to update ticket of their own accord
	2.2	The user receives a request for more information on the ticket and has to update the ticket. The request is emailed.

RELATED INFORMATION	Update ticket
Priority	Critical – Crux of customer care system
Performance	200ms response time
Frequency	2-3 times per day for customer 20-30 per day for tech/care staff
Channels to Actors	Database, Email
OPEN ISSUES	-
Due Date	Alpha release 1.0
...any other management information...	-
Superordinate's	Login, Search Tickets, View Report
Subordinates	-

USE CASE	View Report	
Goal in Context	User wants to view an overall report of the tickets in the systems. Metrics like open vs. closed, performance over time and number of blocking tickets that cost lots of time.	
Scope & Level	Sub Function	
Preconditions	The user is an employee of company. The user has permission to view tickets (Team Leader and all management levels above)	
Success End Condition	The report has been displayed to the user. The graph calculations have been made accurately. (Maybe add the option to customise statistic views and save them to send to other managers to highlight a certain area)	
Failed End Condition	Graphs not displayed.	
Primary, Secondary, Actors	HQ Employee who is also a manager or team leader	
Trigger	User with relevant permissions selects "View Report"	
DESCRIPTION	Step	Action
	1	Employee selects View Report
	2	They then select report type
	3	Report is displayed
EXTENSIONS	Step	Branching Action
	3.1	Add Graphs
	3.2	Save current view
VARIATIONS		Branching Action
	1.1	User has permission
	1.2	User does not have permission - open up a contact form to ask the Admin team to add permissions to your account

RELATED INFORMATION	View Report
Priority	Low Priority – This is an add on to the core system to add to products attractiveness in Enterprise
Performance	700ms response time with lots of data from server. JavaScript drawing the graphs needs to be optimised.
Frequency	20 times per day in worst case
Channels to Actors	Database, Authentication System
OPEN ISSUES	Optimise Graph drawing performance
Due Date	Beta release 1.1
...any other management information...	-
Superordinate's	Login, Search Tickets
Subordinates	<optional , depending on tools, links to sub use cases>



USE CASE	Assign Ticket	
Goal in Context	To specify a user to assign the ticket by overriding the auto assigned user after ticket creation.	
Scope & Level	Sub Function	
Preconditions	The user is an employee of company. The user has permission to reassign tickets.	
Success End Condition	The ticket is either assigned or reassigned. The user the ticket has been assigned to be notified. The user the ticket has been un-assigned from be notified.	
Failed End Condition	The ticket maintains its state and database not updated.	
Primary, Secondary, Actors	HQ Employee – This activity can be restricted to team leaders or open to everyone.	
Trigger	User with relevant permissions attempts to change user assigned to ticket.	
DESCRIPTION	Step	Action
	1	Employee logs in.
	2	Ticket is attempted to be reassigned
EXTENSIONS	Step	Branching Action
	2.1	Optionally Add reasoning why ticket was reassigned.
	2.2	Notify management of a serious issue.
VARIATIONS		Branching Action
	1.1	User has permission
	1.2	User does not have permission - open up a contact form to ask the Admin team to add permissions to your account

RELATED INFORMATION	Assign Ticket
Priority	Medium Priority – This is something that is pretty much necessary for the system.
Performance	100ms response time very little data in this transaction
Frequency	5 times per day in worst case
Channels to Actors	Database, Authentication System
OPEN ISSUES	-
Due Date	Alpha release 1.0
...any other management information...	-
Superordinate's	Login, Search Tickets, Update Ticket, Create Ticket
Subordinates	<optional , depending on tools, links to sub use cases>

USE CASE	Create User	
Goal in Context	Add a new user to the system when a new person joins a team.	
Scope & Level	Primary	
Preconditions	The user is an employee of company. The user is a System Administrator.	
Success End Condition	The user is added to a team. The Manager and Team Leader are notified. The user is set up with the relevant permissions.	
Failed End Condition	The user is not added to the system.	
Primary, Secondary, Actors	System Administrator and Manager.	
Trigger	User with relevant permissions attempts to change user assigned to ticket.	
DESCRIPTION	Step	Action
	1	System Administrator Logs In
	2	User clicks on Create User
	3	Adds information of new employee
	4	Confirms employee's status (Whether admin, manager or regular employee)
EXTENSIONS	Step	Branching Action
	2.1	Choose whether to notify Manager and Team Leader
VARIATIONS		Branching Action
	1.1	User has permission
	1.2	User does not have permission – State that functionality is for admins only.

RELATED INFORMATION	Create User
Priority	High Priority – System needs users
Performance	300ms response time
Frequency	3 times per day in worst case
Channels to Actors	Database, Authentication System
OPEN ISSUES	-
Due Date	Alpha release 1.0
...any other management information...	-
Superordinate's	Login, Search Tickets, Update Ticket, Create Ticket
Subordinates	<optional , depending on tools, links to sub use cases>

5.2 Non Functional Requirements

Performance: The Company has over 1 million customers and needs the system to be able to cope with a large number of Support Tickets into the system. A poor performing system affects both our staff in doing their jobs and wastes customer time.

Reliability: If the system is down our customer care service is effectively down. The company prides themselves on reputation so an unreliable system directly affects this. A way of backing up the systems database should be available and we should take this into consideration when choosing a DBMS.

Portability: Our non-technical staff uses Windows while our technical software and hardware staff use Linux. A system that is accessible from all platforms is important. There will be instances where off-site staff may require access so a web/mobile interface would be needed also.

Usability: The system needs to be Usable so staff can deal with customers efficiently. If our staff are getting held up because the system is difficult to use then the system has failed.

Standards: Our in-house Quality team abides by the ISO 9001 standard for Software Quality. ISO accreditation is important to many of our business customers.

Implementation: The product must have a certain standard of design to meet with the standards of other products we have produced and so these products speak a similar design language.

Modifiability: The system should have a modular nature so that extra functionality can be added later.

Recoverability: The system should be reasonably rapid to recover from downtime and outages. For example, the Ticket Management system crashing should not lead to a halt to the system. Tickets should be queued and stored until the system is restored.

Documentation: A full training manual needs to be provided in order to educate staff quickly and to ensure the efficient use of the system is maximised. There will need to be manuals for each level of Staff member.

Scalability: The system needs to be able to handle multiple tickets being created at the same time. The most that can be created would logically be one per staff member at any given time, assuming that each staff member may only be logged into any one interface at a time. With 800 staff, we should be able to handle 800 tickets being created at once.

5.3 Discussion on Tactics to Support Specific Quality Attributes

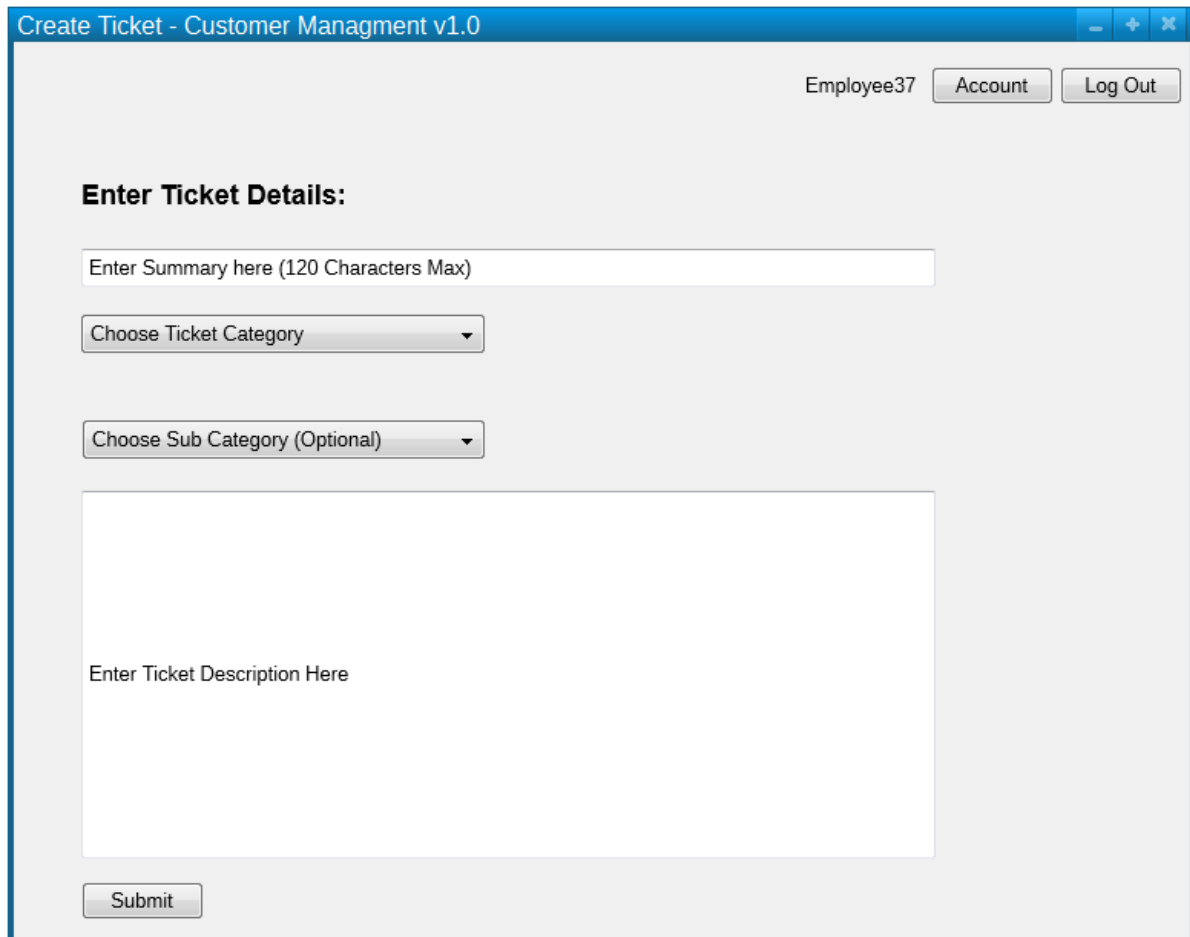
After extensive research it was decided that we will choose Java as our object orientated language. We have chosen Java for its portability and especially for the Java EE 6 platform. Java EE 6 allows us to implement a Model View Control system whereby a remote client can login to our system with a web site and a local client can login to the system with a Java application. Our website back end will be JSP, Servlet and Java Enterprise Beans based and our front end will be enhanced with JavaScript.

Staff using the application on the retail side will use a web based system and staff using the application at company Headquarters will use a local Java client using Java's native UI. Engineers who are out and about doing work can access through either the web based system or the mobile based system. While Java isn't known for its performance, we feel that its benefits in portability far outweigh its negatives in response times.

Unfortunately, Java also has a bad reputation for security breaches, and we will make an effort to secure our application but any major security effort is outside the scope of the project. By choosing Java, our developers can develop faster and create a rough prototype in a few weeks compared to languages like C++, where development is a protracted process. Overall, the benefits of Java are clearly what the business needs right now. We will try to develop in a modular way so that porting to other languages is possible if required but the load the system can take and the ease of access from multiple locations along with reliability is more important than saving nanoseconds.

5.4 Screen Shots

5.4.1 Create Ticket



The screenshot displays a web application window titled "Create Ticket - Customer Management v1.0". In the top right corner, the text "Employee37" is shown next to two buttons: "Account" and "Log Out". The main content area is headed by "Enter Ticket Details:". Below this header, there is a text input field with the placeholder "Enter Summary here (120 Characters Max)". This is followed by two dropdown menus: "Choose Ticket Category" and "Choose Sub Category (Optional)". Below these is a large text area with the placeholder "Enter Ticket Description Here". At the bottom left of the form, there is a "Submit" button.

Fig. Example of a Ticket Creation UI (created in Pencil)

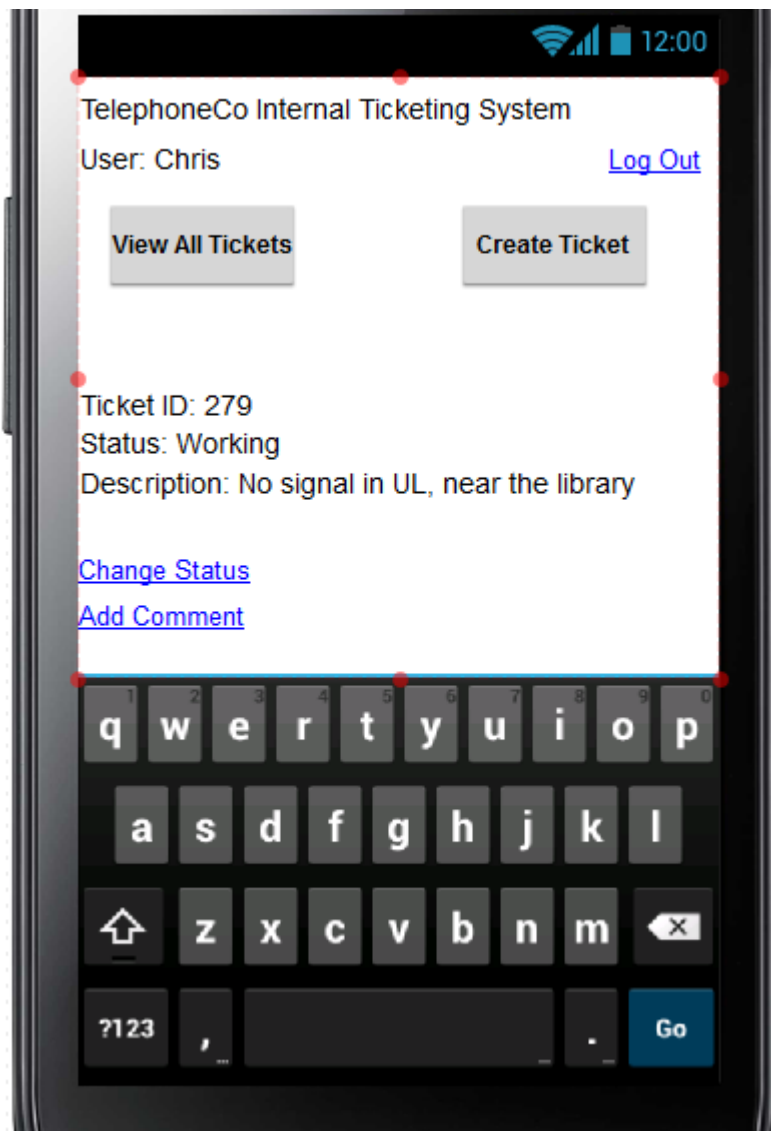


Fig: Example screen shot of limited site on a mobile device.

5.4.2 Ticket Report Custom View

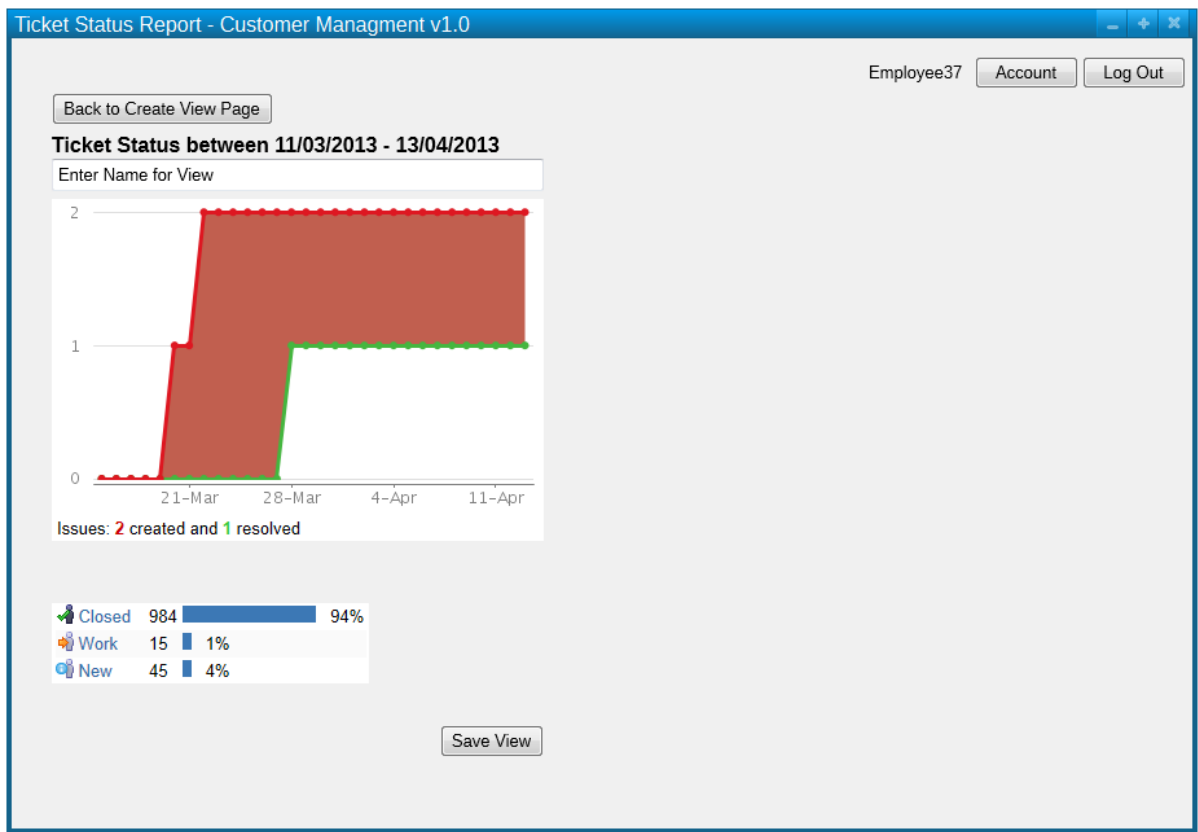


Fig. Example of a Report showing Ticket statuses (Created in Pencil)

5.5 Layouts/Format for Reports

Employee Reports - Customer Managment v1.0

Supervisor38 Account Log Out

Workload Report:

Employee	Assigned Issues	Workload
Shane Whelan	14	2 Weeks
Chris O'Brien	13	2.5 Weeks
J.J. Collins	999	> 1 Year

Refresh Report

Print Report Send Report

Fig. Example of a Supervisor report on Staff that he is responsible for (Created in Pencil)

6. Analysis

6.1 List of Candidate Classes

We used Data Driven Design for key domain abstraction using the noun identification technique. We underlined noun and noun phrases in each Use Case Description in one diagram. Using this technique we identified some candidate classes.

Authentication	Author	Customer
Customer Care System	Database	Day
Department	Email	Employee
File	Graph	HQ Employee
Image	Information	Manager
Notification	People	Performance
Person	Products	Request
Retail Employee	Report	Support
System	Team	Team Leader
Ticket	User	View

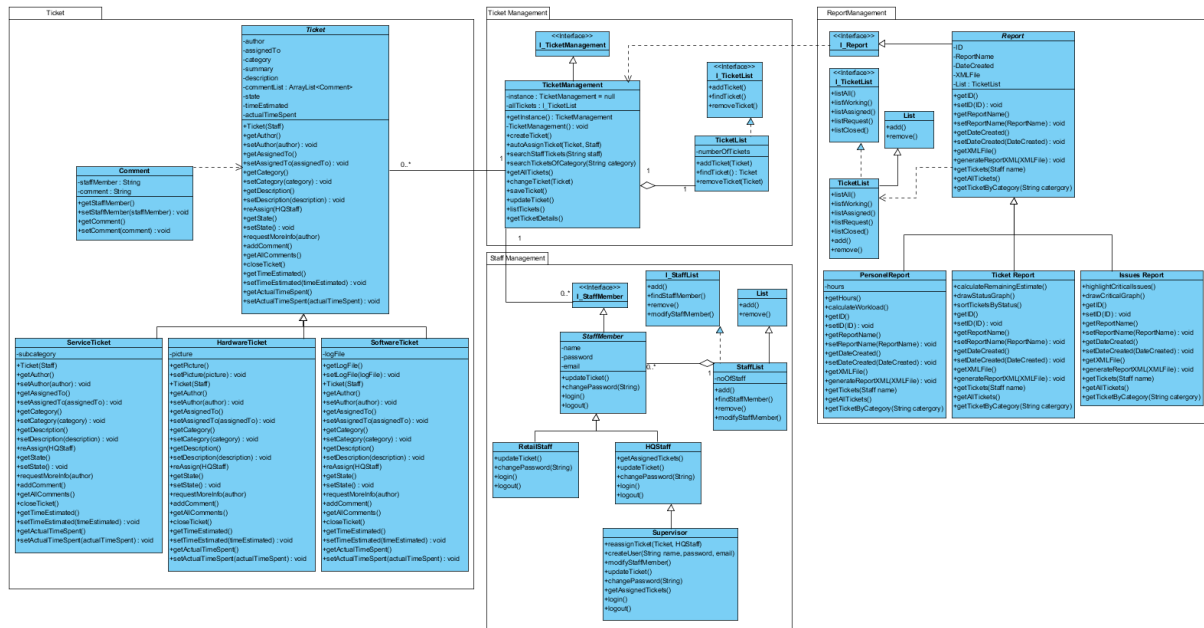
6.2 Applying Heuristics to Eliminate Poor Candidates

Authentication	Author	Customer
Customer Care System	Database	Day
Department	Email	Employee
File	Graph	HQ Employee
Image	Information	Manager
Notification	People	Performance
Person	Products	Request
Retail Employee	Report	Support
System	Team	Team Leader
Ticket	User	View

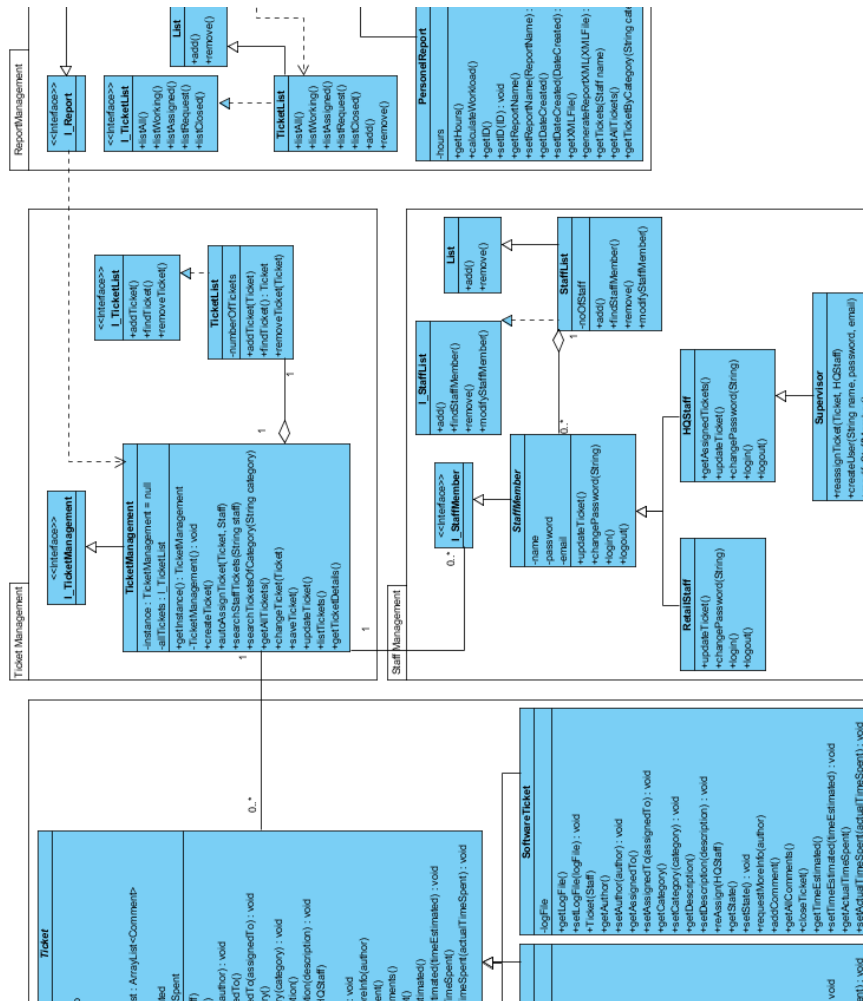
6.3 Leftover Entity Classes

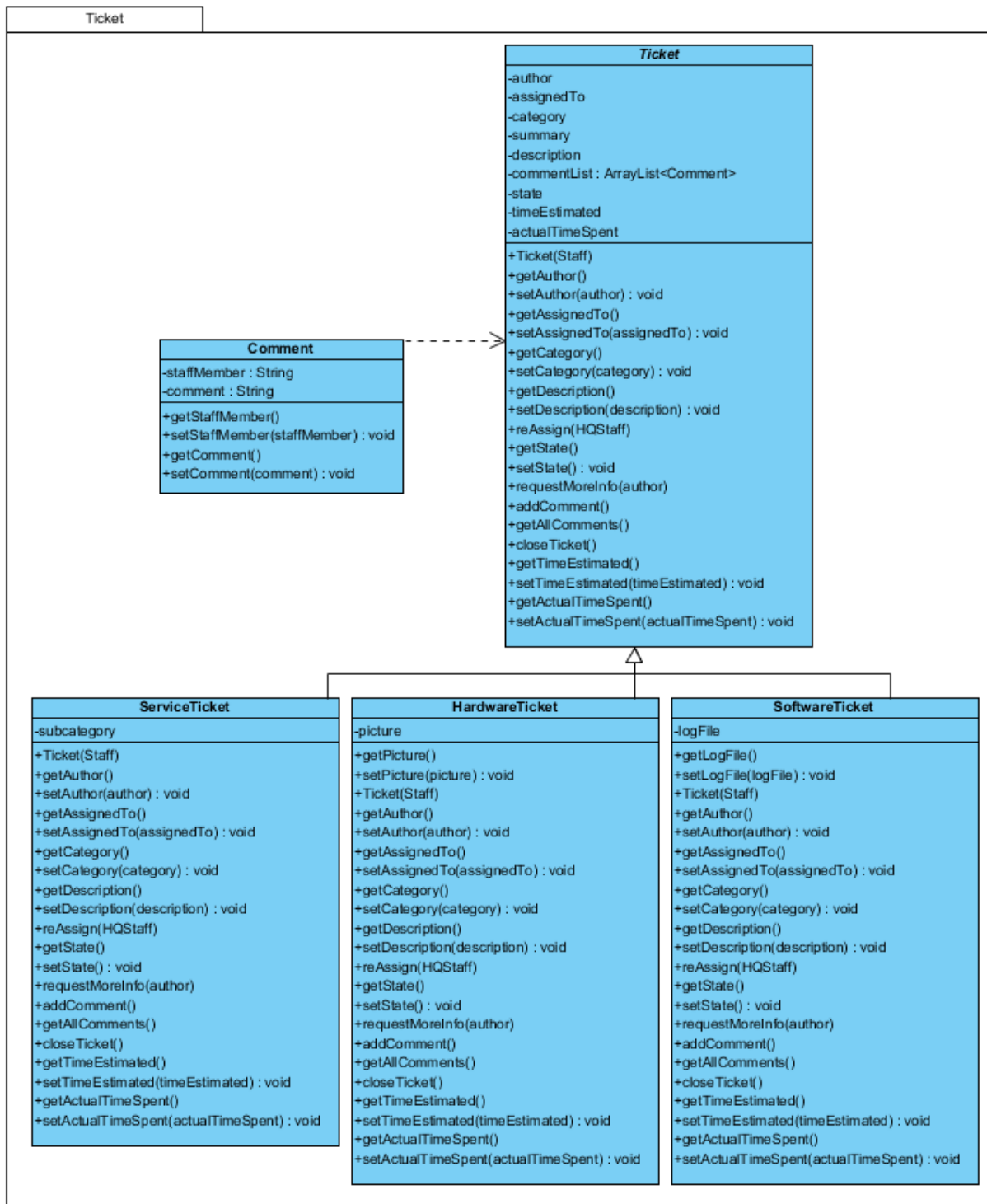
Authentication
Employee
Graph
HQ Employee
Manager/Team Leader
Notification
Request
Retail Employee
Report
Team Leader
Ticket
View

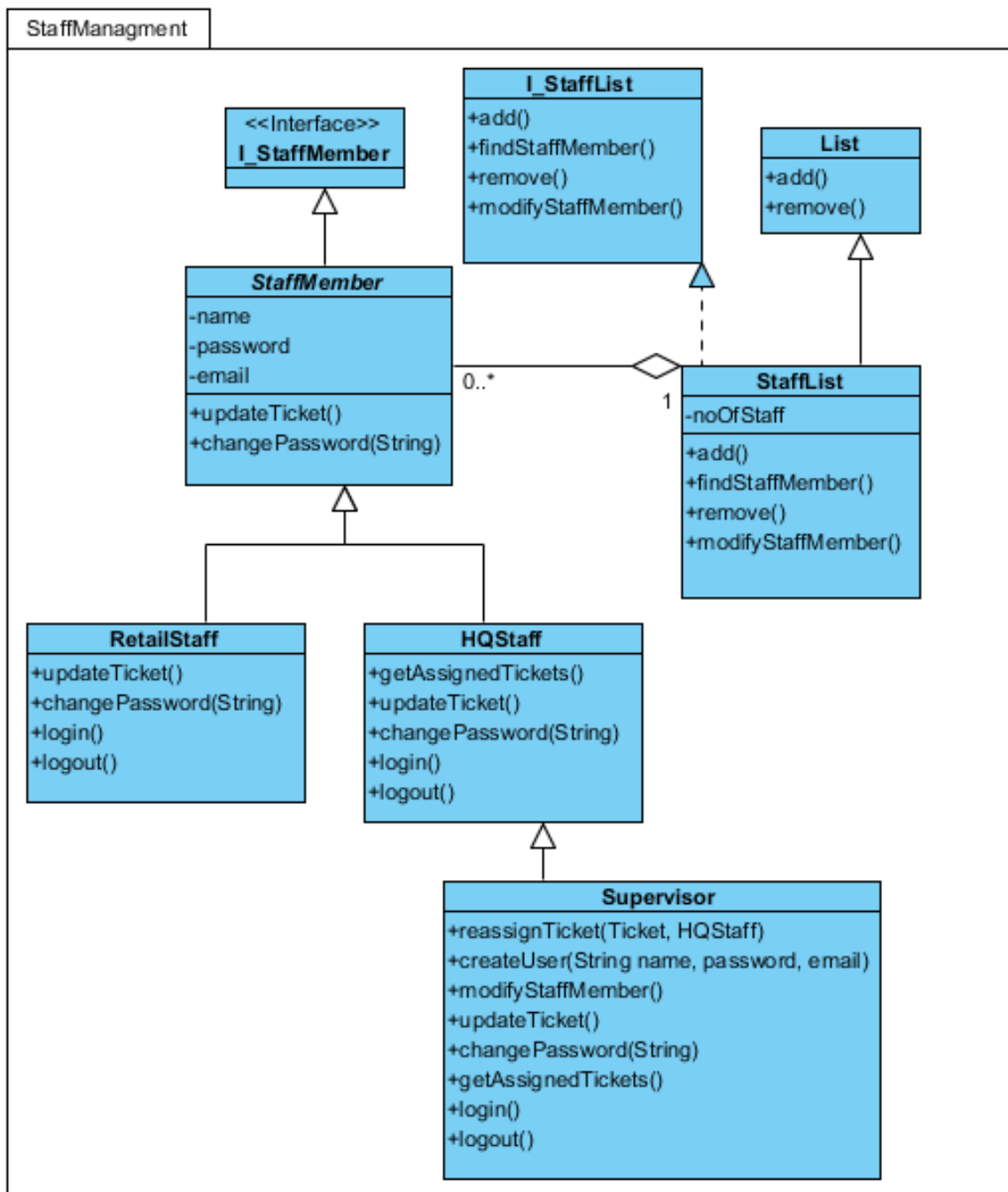
6.4 Class Diagram



Note: Landscape version on next page.







OCL Constraints are present on the StaffMangement Package

"A Staff Member cannot have a password that is less than 6 characters"

Context StaffMember

inv: Staff.password.length > 6

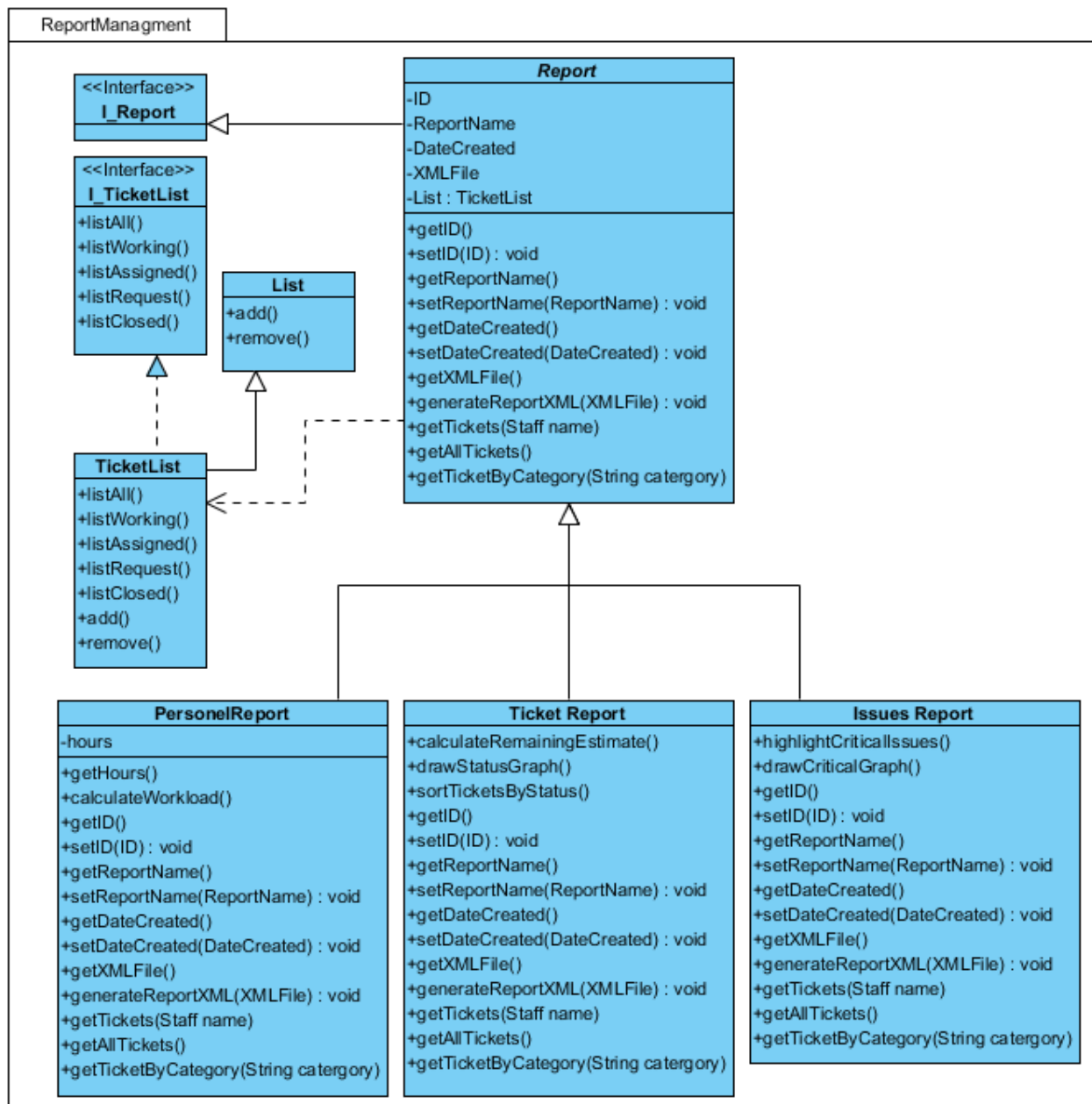


Fig: ReportManagement Package

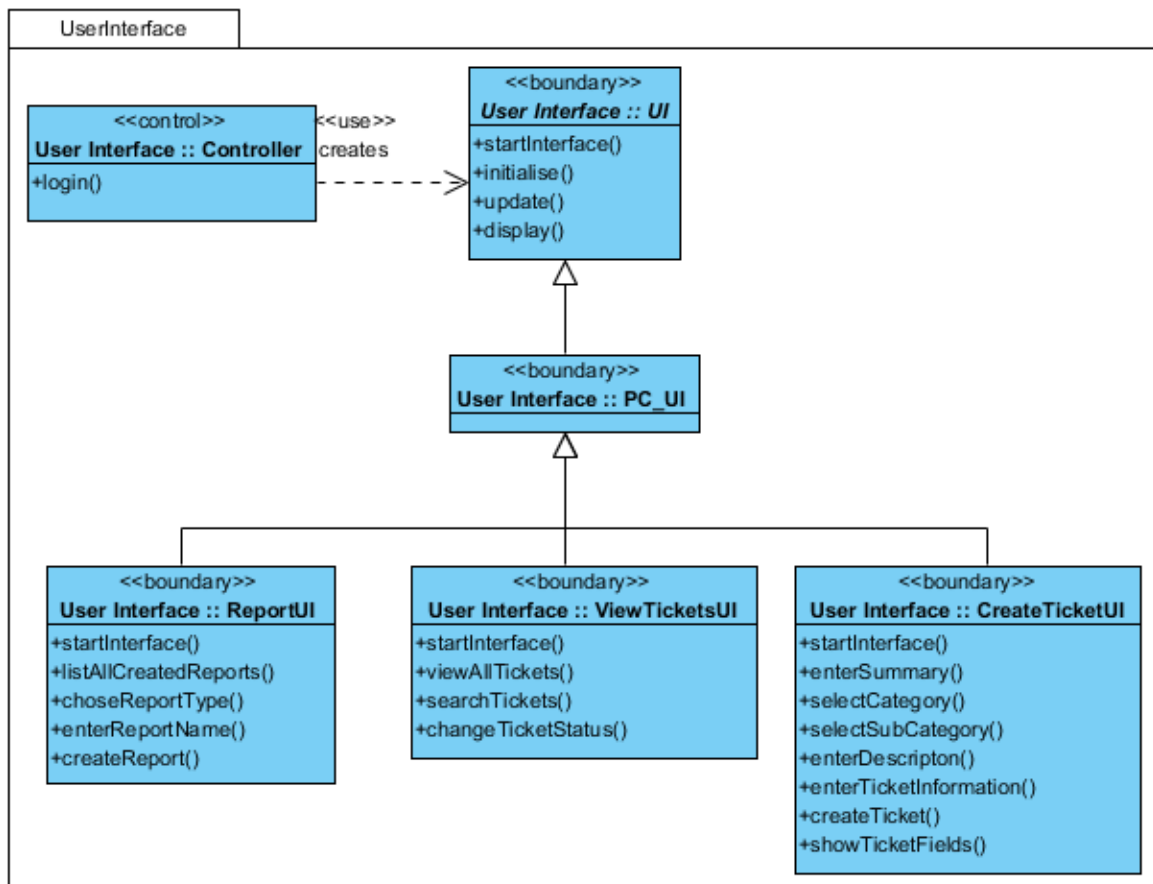
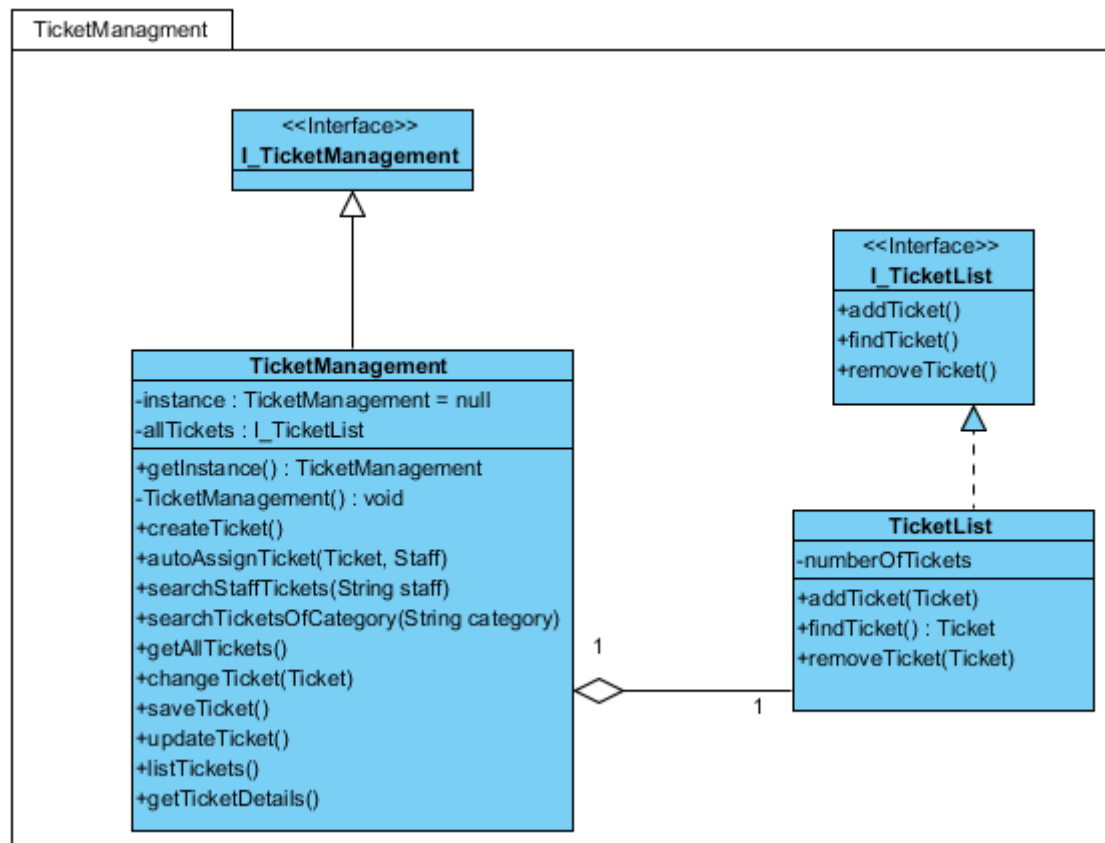


Fig: User Interface Class Diagram

6.5 Sequence Interaction Diagram – Change Status

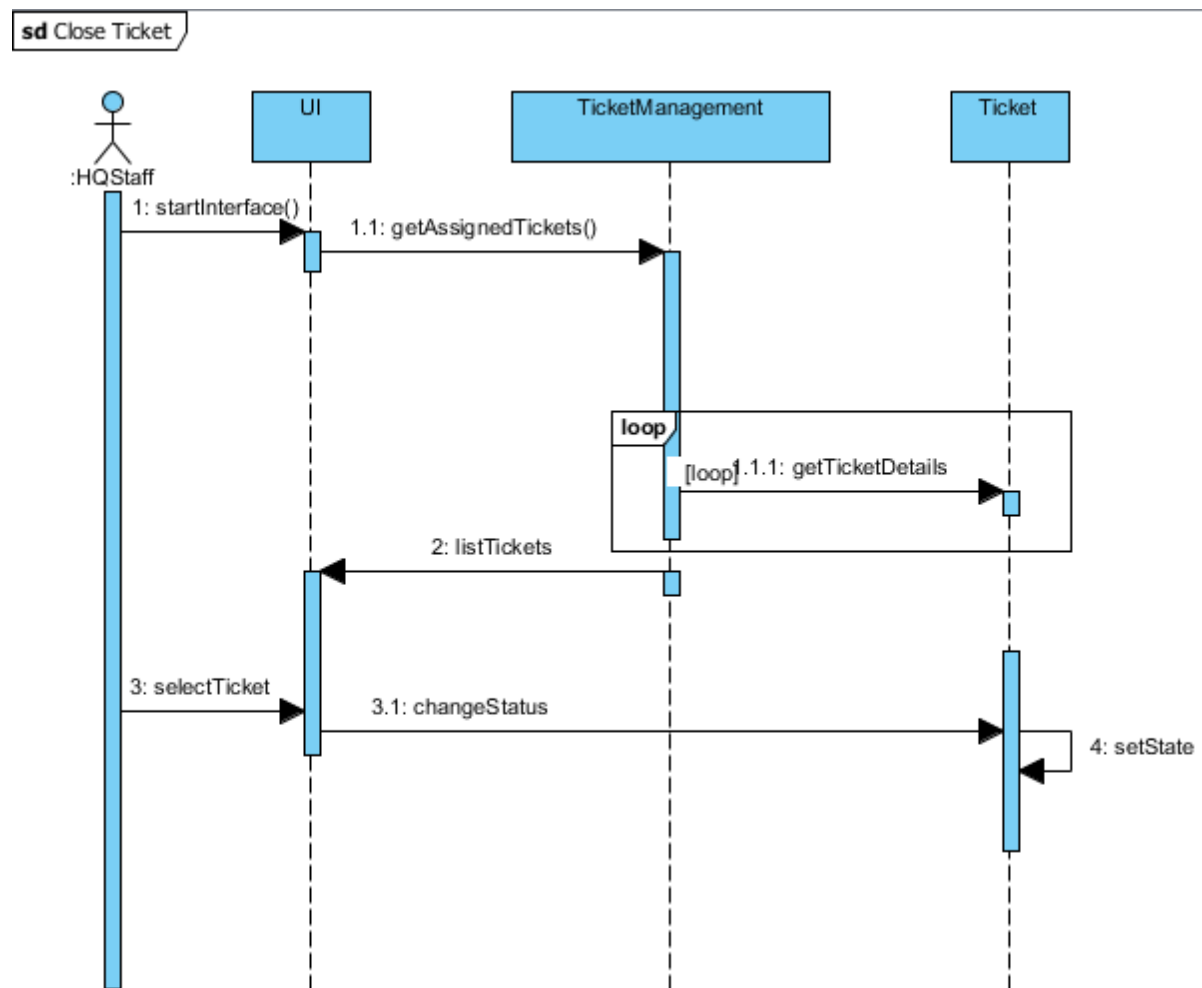


Fig: How to change the status of an existing Ticket

6.6 Communication Diagram – Create Ticket

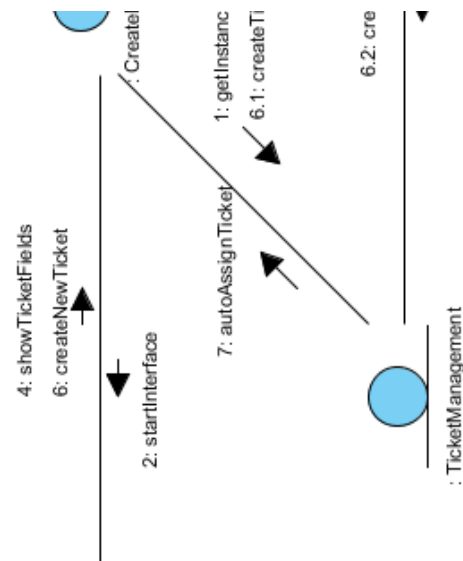


Fig. Communication diagram for Creating a new Ticket

Pre and Post Condition Example

Context	TicketManagement
Operation Specification	createTicket()
Operation Intent	A ticket is created
Operation Signature	TicketManagement::createTicket()
Logic Description	
Pre	Self->exists(), Staff->loggedIn() = true
Post	TicketManagement::autoAssignTicket()
Other Operations Called	None
Events Transmitted	None
Attributes set	None
Response to Exceptions	None defined
Non-functional requirements	None defined

6.7 State Chart

This is a State chart for a Ticket object, which incorporates the State design pattern.

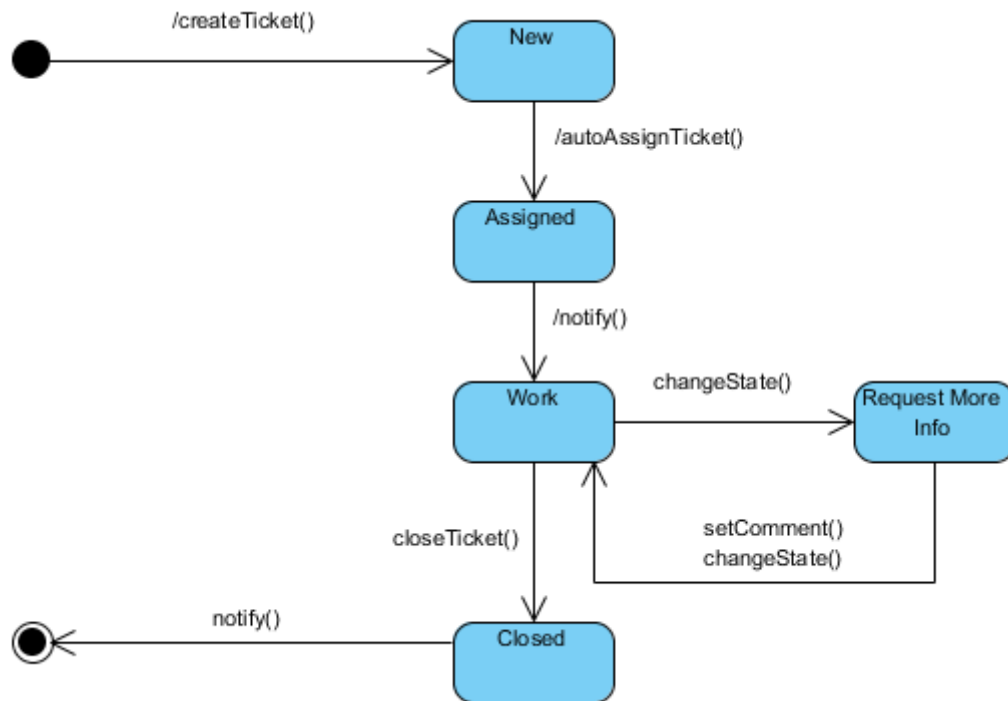


Fig: State Chart showing the lifecycle and possible States that a Ticket may be in

6.8 Entity Relationship Diagram



7. Design

7.1 Description of Architectural Style

7.1.1 The State Pattern

The State Pattern is used in our project in relation to the ticket class and in conjunction with the Observer Pattern, detailed below. Each ticket may travel between a number of states during its lifecycle. It uses the Observer Pattern to notify relevant parties.

7.1.2 The Observer Pattern

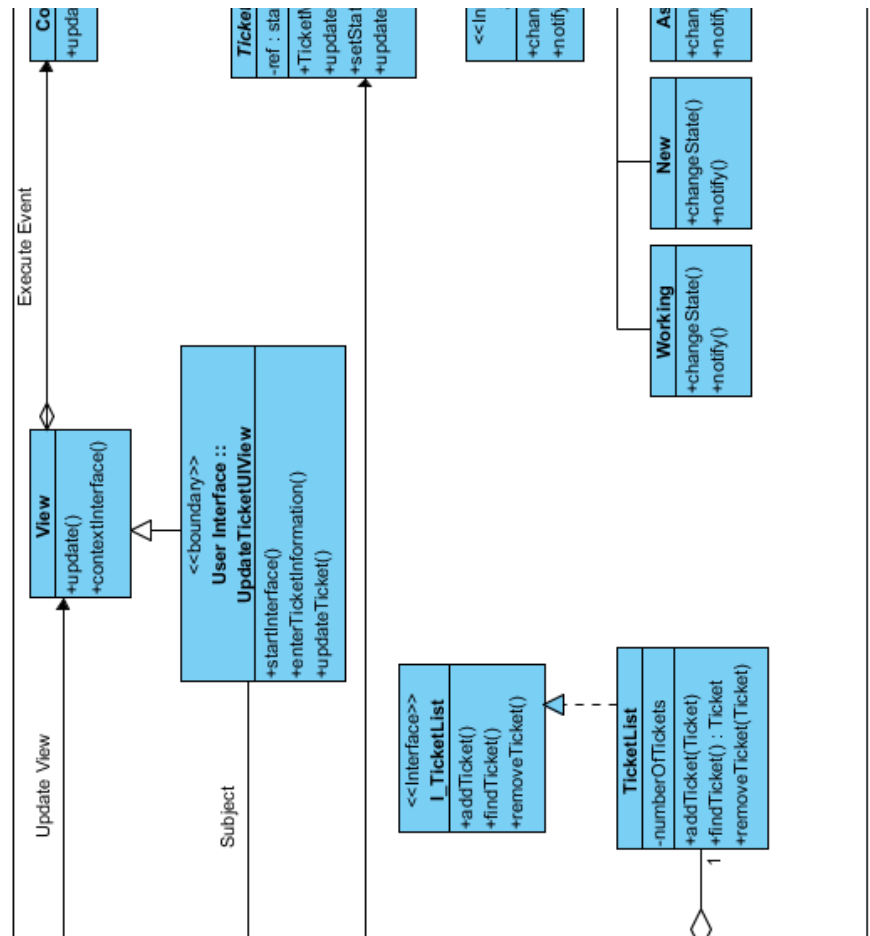
The Observer Pattern is used in our project in relation to Tickets, their owners and assignees (both members of the Staff class) – In the event of a change of a Tickets state, the notify() method is called, and depending on the state the ticket is currently in, the dependents are notified.

In our system, the Ticket class is the subject which maintains a list of dependents, in this case instances of the abstract class Staff, which contains the sub classes RetailStaff, HQStaff and Supervisor. Upon change of a Ticket state, notify is invoked. Table illustrates this.

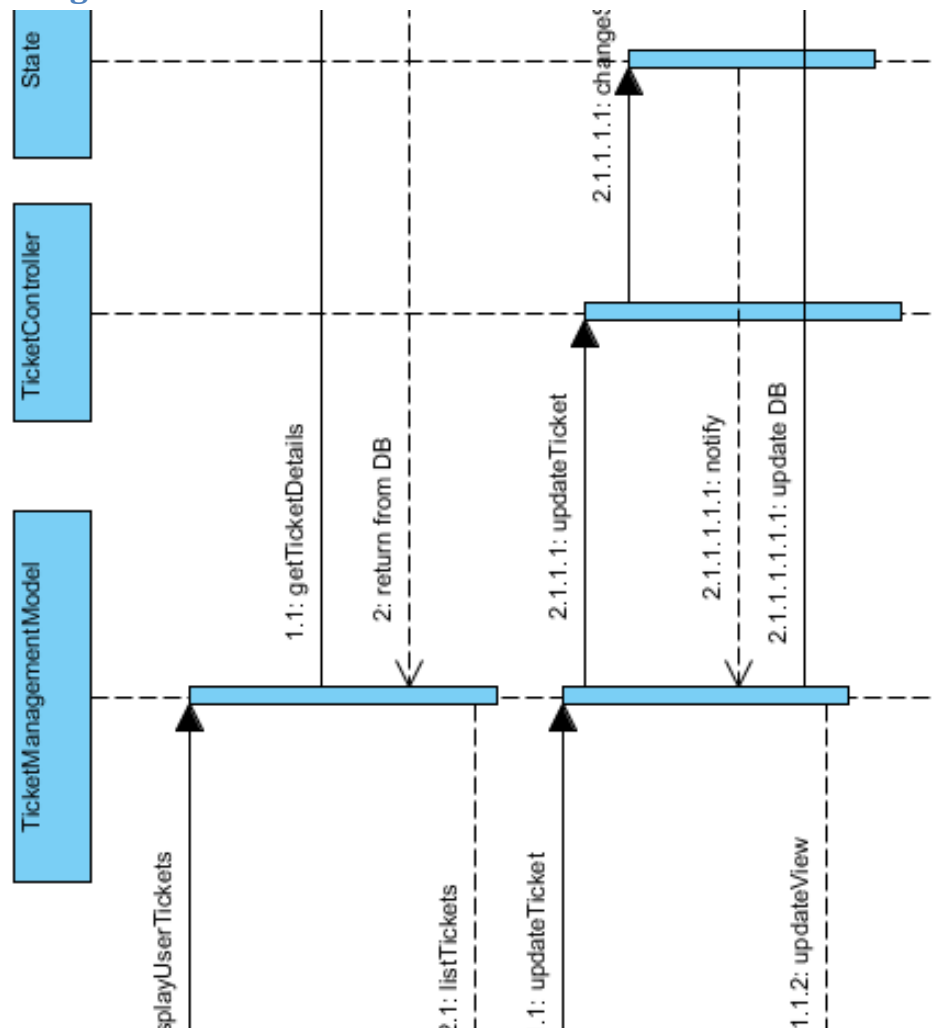
<u>Ticket State</u>		<u>Staff Emailed</u>
New	<u><i>Notify() invoked</i></u>	Ticket Author
Pending	<u><i>Notify() invoked</i></u>	Ticket Assignee
Working	<u><i>Notify() invoked</i></u>	Ticket Author
Ticket Updated (i.e. Comments)	<u><i>Notify() invoked</i></u>	Ticket Author, Ticket Assignee
Request More Info	<u><i>Notify() invoked</i></u>	Ticket Author
Ticket Closed	<u><i>Notify() invoked</i></u>	Ticket Author, Ticket Assignee

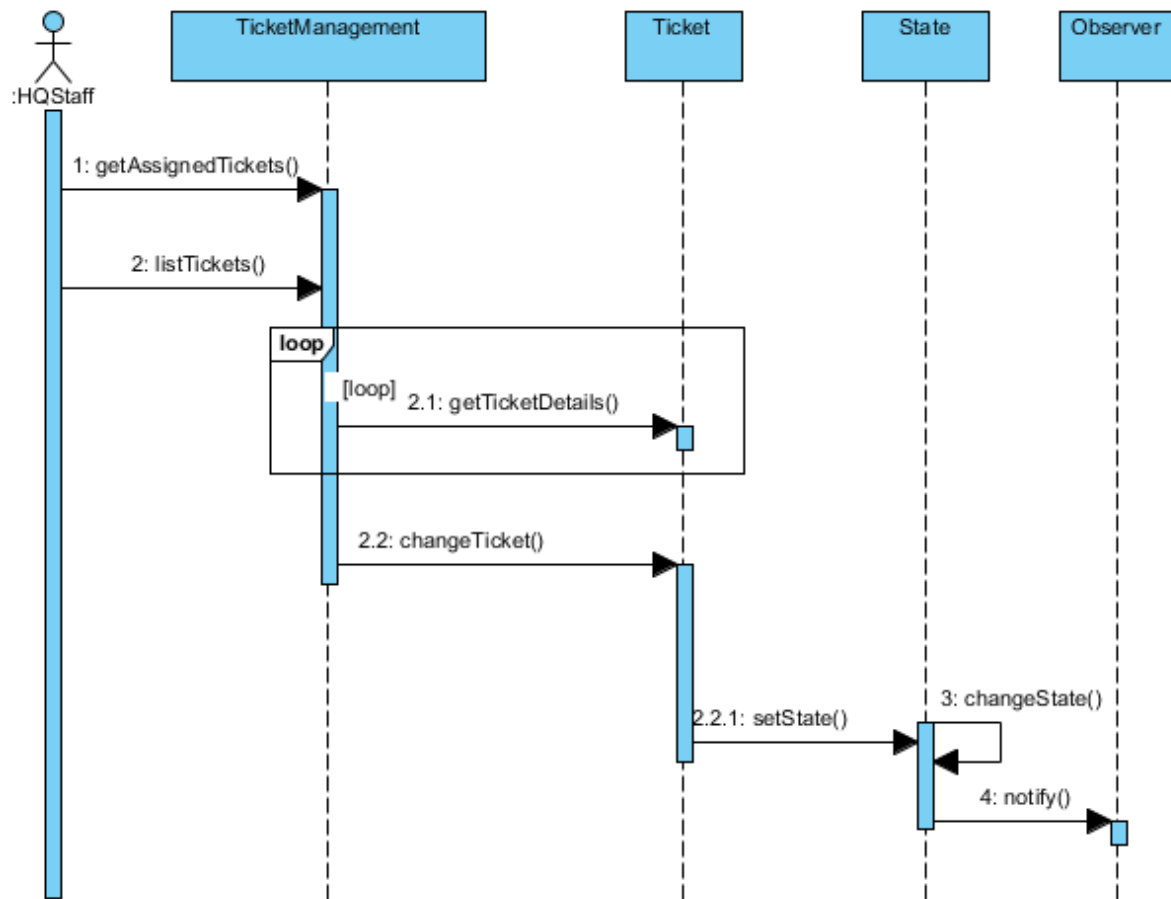
The list of dependents on a ticket can be modified, but only by an object that is a Supervisor instance. The parameters of the reassignStaff() operation must contain a valid HQStaff instance, or it will fail the precondition. Notify() will also be called in this event to notify the Ticket Author, Previous Assignee and New Assignee of the change. A ticket author cannot be changed and a new ticket must be created if a different person wishes to be notified. With this pattern however, it is relatively straightforward to enable Tickets to have multiple observers with minor changes to the notify() method and the creation of a watchTicket() method which will add a Staff object to the List of Observers associated with that Ticket object.

7.2 Class Diagram Including MVC



7.3 Sequence Diagram Including MVC





7.4 Concurrency in Design Time Diagrams

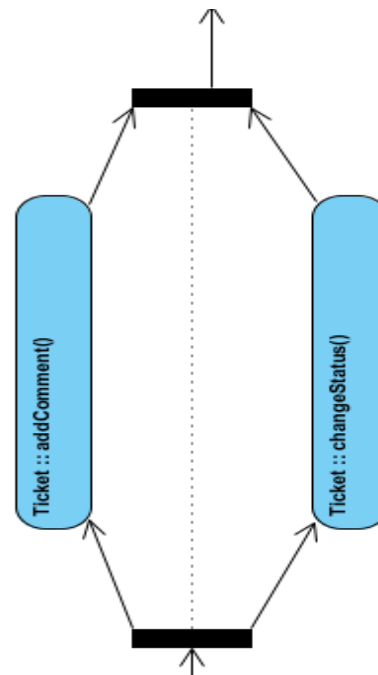


Fig: Concurrency Diagram – When changing the status of a Ticket, a comment must be added. The UI object initialises the data change, which is updated synchronously and then saved by the TicketManagement system.

8. Technical Architecture Diagram

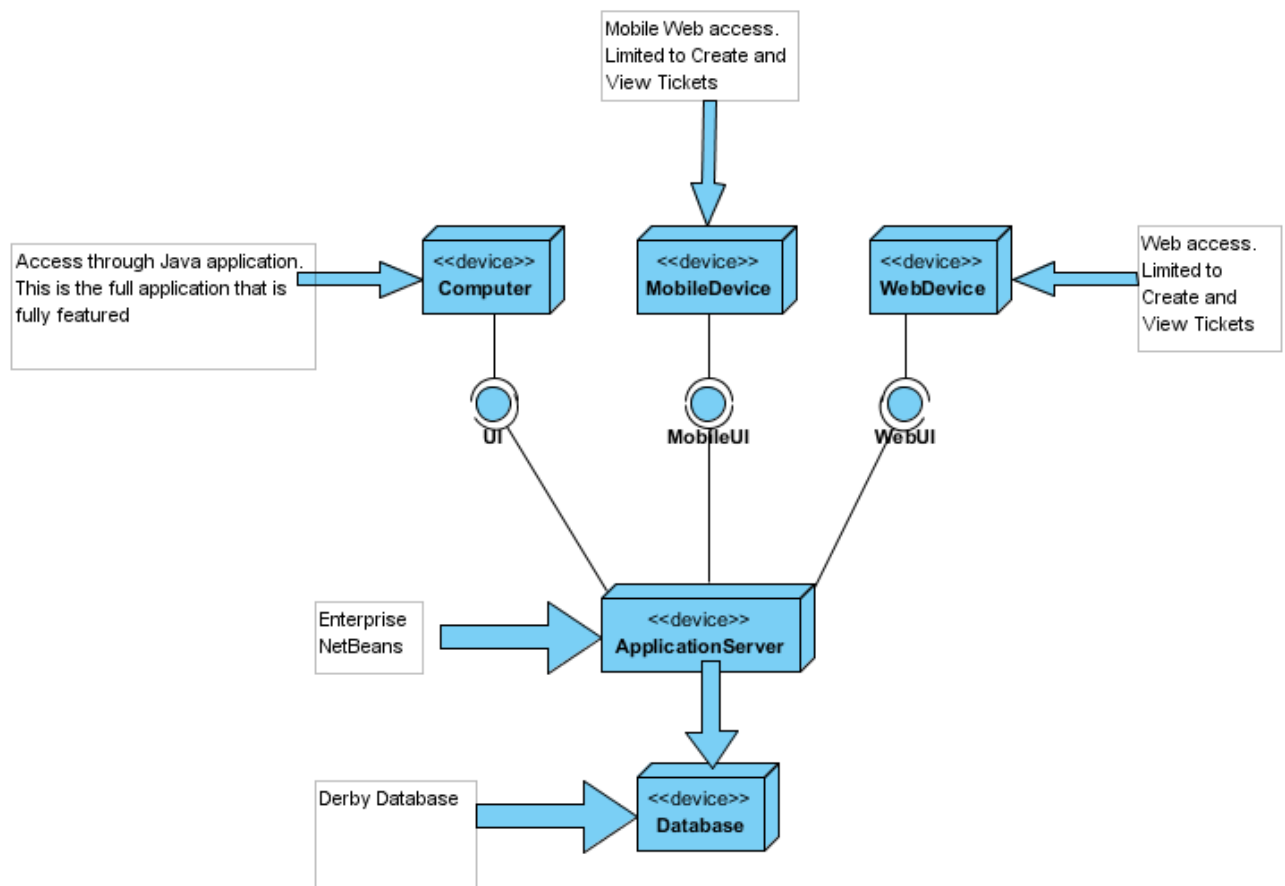


Fig. Technical Architecture Diagram using the 3 methods of using the system

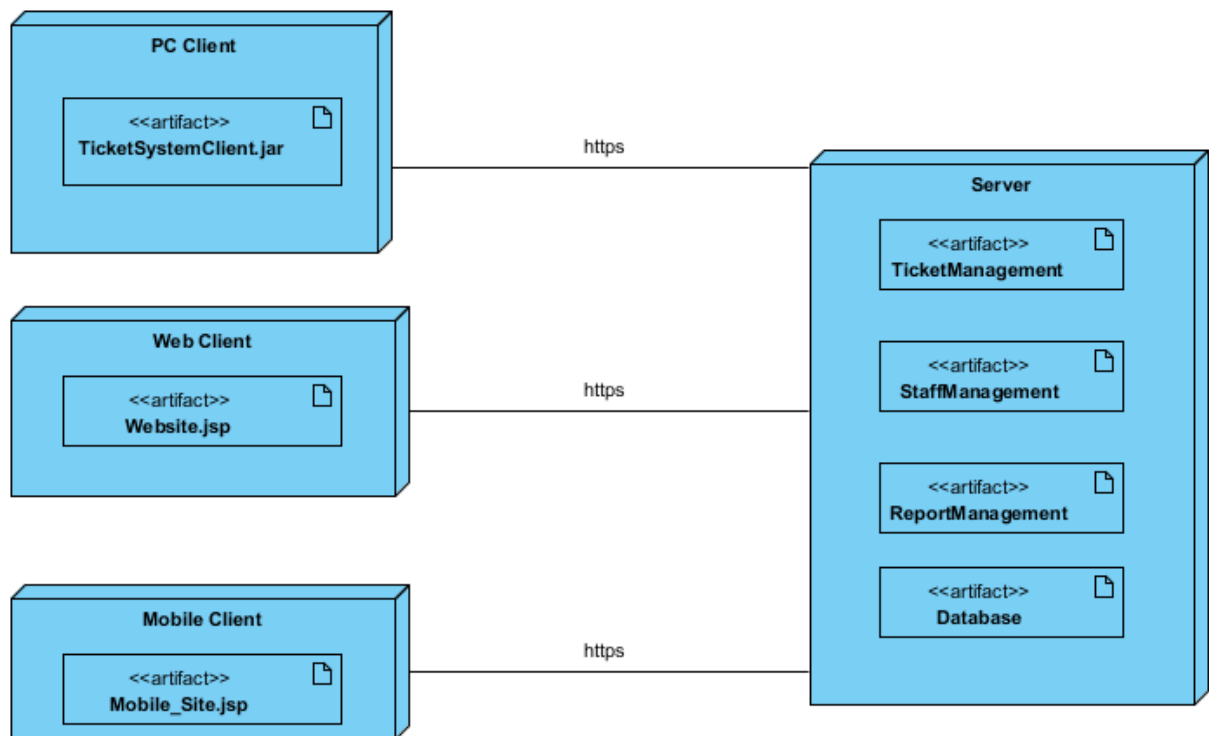


Fig: Example of System Deployment

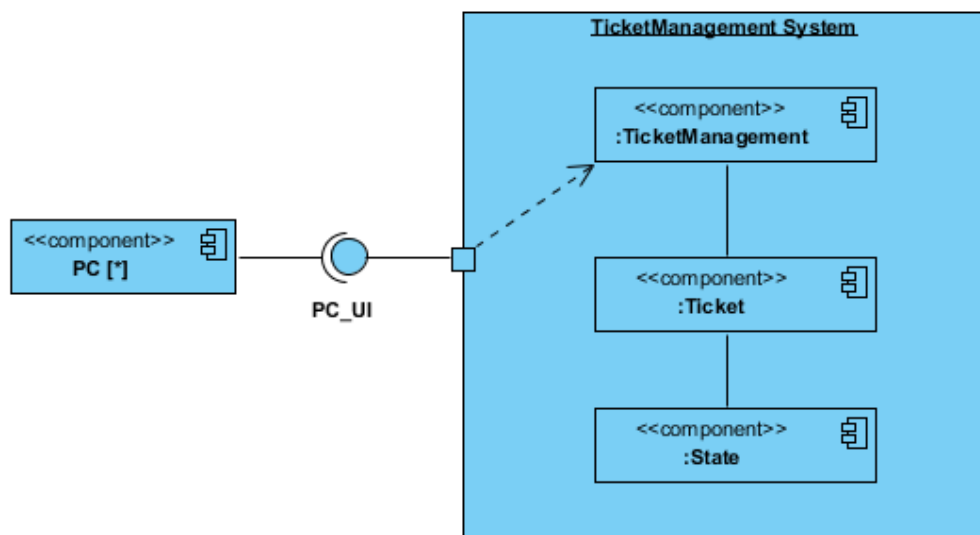


Fig: Example of a Component Diagram

9. Data Dictionary

Name	<i>Staff</i>
Type	<i>Abstract Class</i>
Description	<i>The Staff class provides the base functionality for its subclasses: RetailStaff, HQStaff and Supervisor</i>

Name	<i>RetailStaff</i>
Type	<i>Actor</i>
Description	<i>The RetailStaff actor is a member of customer facing staff who has the ability to communicate problems to those responsible for fixing them. In the scope of this system, this actor may</i> <ul style="list-style-type: none">- <i>Create a new ticket</i>- <i>Search for tickets</i>- <i>Add a comment to a ticket</i>

Name	<i>HQStaff</i>
Type	<i>Actor</i>
Description	<i>The HQStaff actor is a member of non-customer facing staff who has the ability to fix and resolve issues in tickets created by, though not exclusively, RetailStaff. They may change the status of a ticket to Request More Info or Closed.</i> <ul style="list-style-type: none">- <i>Create a new ticket</i>- <i>Search for tickets</i>- <i>Change status of a ticket</i>- <i>Add a comment to a ticket</i>

Name	<i>Supervisor</i>
Type	<i>Actor</i>
Description	<i>A supervisor is a subclass of HQStaff and can perform all the staff functions but is responsible for managing staff creation, monitoring sets of staff and reassigning, if necessary, Tickets to more qualified Staff</i> <ul style="list-style-type: none">- <i>Create a new ticket</i>- <i>Search for tickets</i>- <i>Add a comment to a ticket</i>- <i>Staff creation</i>- <i>Reassign Staff to Ticket</i>- <i>Change status of a Ticket</i>

Name	<i>Ticket</i>
Type	<i>Abstract Class</i>
Description	<i>The superclass for all Ticket instances that provides the default and common attributes and operations for its subclasses</i>
Name	<i>CareTicket</i>
Type	<i>Logical Entity</i>
Description	<i>A ticket created for Customer Care issues, such as billing, coverage issues etc</i>
Name	<i>HardwareTicket</i>
Type	<i>Logical Entity</i>
Description	<i>A ticket created for hardware faults within the company.</i>
Name	<i>SoftwareTicket</i>
Type	<i>Logical Entity</i>
Description	<i>A ticket created for software faults within the company, unrelated to customer issues. A prime example is forgetting login password.</i>
Name	<i>Observer</i>
Type	<i>Interface(?)</i>
Description	<i>Design Pattern to notify Staff objects who are registered to be aware of any changes to a Ticket</i>
Name	<i>Report</i>
Type	<i>Abstract Class</i>
Description	<i>A superclass to provide basic report functionality for its subclasses</i>
Name	<i>TicketList</i>
Type	<i>Logical Entity</i>
Description	<i>A list of Tickets required to generate a particular report</i>
Name	<i>I_TicketList</i>
Type	<i>Interface</i>
Description	<i>Interface for TicketList which provides the functionality needs to display all the Ticket information contained within the TicketList instance</i>

Name	<i>PersonnelReport</i>
Type	<i>Logical Entity</i>
Description	<i>A class which provides the template and retrieved the information for the Personnel Report, which shows Staff and how many tickets they are responsible for and the relevant states for said tickets.</i>

Name	<i>TicketReport</i>
Type	<i>Logical Entity</i>
Description	<i>A class which provides the template and retrieved the information for the Ticket Report, which is a variety of information about the tickets such as time open, number of tickets pending etc</i>

Name	<i>IssuesReport</i>
Type	<i>Logical Entity</i>
Description	<i>A class which provides the template and retrieved the information for the Issues Report, which is a list of Tickets outstanding by category</i>

Name	<i>UserInterface</i>
Type	<i>Abstract Class</i>
Description	<i>This is an abstract class to provide base functionality to its subclasses</i>

Name	<i>PC_UI</i>
Type	<i>Logical Entity</i>
Description	<i>An instance of the PC/Desktop UI</i>

Name	<i>Web_UI</i>
Type	<i>Logical Entity</i>
Description	<i>An instance of the Web/Browser UI</i>

Name	<i>Mobile_UI</i>
Type	<i>Logical Entity</i>
Description	<i>An instance of the Web/Browser UI, streamlined for mobile devices. This formats the site to fit a smaller screen and uses lower quality, therefore faster to load, artefacts.</i>

Name	<i>ViewTicketsUI</i>
Type	<i>Interface</i>
Description	<i>The UI for Viewing Tickets</i>

Name	<i>CreateTicketUI</i>
Type	<i>Interface</i>
Description	<i>The UI for Creating a Ticket</i>

Name	<i>ViewReportUI</i>
Type	<i>Interface</i>
Description	<i>The UI for Viewing Reports</i>

Name	<i>CreateReportUI</i>
Type	<i>Interface</i>
Description	<i>The UI for Creating Reports</i>

Name	<i>State</i>
Type	<i>Abstract Class</i>
Description	<i>An abstract class, from the State Pattern design, that cannot be instantiated, but provides base functionality for its subclasses</i>

Name	<i>New</i>
Type	<i>Logical Entity</i>
Description	<i>An instance of a subclass of the State class</i>

Name	<i>Assigned</i>
Type	<i>Logical Entity</i>
Description	<i>An instance of a subclass of the State class</i>

Name	<i>Working</i>
Type	<i>Interface</i>
Description	<i>An instance of a subclass of the State class</i>

Name	<i>RequestMore</i>
Type	<i>Interface</i>
Description	<i>An instance of a subclass of the State class</i>

Name	<i>Closed</i>
Type	<i>Interface</i>
Description	<i>An instance of a subclass of the State class</i>

10. Critique of Design Quality

While this report was always a work in progress, with a lot of going back and forth as we learn and find out about new things, we do feel the design is overall a good foundation for a real product. We feel that our diagrams make sense, and give a good understanding of how this system should be built and that our architectural design would support the construction and maintenance of a completed project of this scope.

There was also a learning curve in the actual use of the Visual Paradigm software as well as getting to grips with the plethora of notations and diagrams contained within both VP and the different versions of UML (1.x and 2.x) with different notations for things like control, entity and boundary classes.

When we designed our classes, we ensured that they complied with the Liskov Substitution Principle. The abstract class Staff which has 3 subclasses where each subclass has more functionality than the base Staff class. For example, any member of the Staff class would be able to create a ticket but only a member of the HQStaff class or any class that inherits from it would be allowed to modify the status of a ticket.

We briefly touched on OCL in terms of the Staff class again, to give some form of password integrity by not allowing passwords to be shorter than a certain length. This could be expanded to make a password contain a character from a set of characters (such as !"@\$# etc) but for the scope of this project, basic use will suffice. OCL is a key part in moving from a model to a more details specification in the project by introducing constraints on entities. These are extremely important, as the lack of constraints can lead to errors in the business logic such as two tickets having the same id or a ticket being assigned to a Staff member who does not have an email address etc.

11. References

- Aked, M. (2011). *RUP In Brief*. Retrieved from <http://www.ibm.com/developerworks/rational/library/1826.html#N100E4>.
- CodeProject.com. (n.d.). *MVC Class Diagram*. Retrieved from <http://www.codeproject.com/Articles/42830/Model-View-Controller-Model-View-Presenter-and-Mod>.
- Community Edition License*. (n.d.). Retrieved from VisualParadigm Software: <http://www.visual-paradigm.com/product/vpuml/editions/community.jsp>
- Geoghean, C., & Frawley, D. (2008). *Module Webpage - Sample Project*. Retrieved from CSIS: <http://www.csis.ul.ie/coursemodule/CS4125>
- IBM. (n.d.). *RUP Diagram*. Retrieved from <http://www.ibm.com/developerworks/rational/library/dec06/ambler/image001.jpg>.
- Initial Operation Capability Milestone*. (n.d.). Retrieved from IRIS Process Central Sandbox: <http://process.osellus.com/Process%20Documents/OpenUP%20SDM/PPM/HTML/PS-00001/ac-00049.html>
- Montreal, E. P. (2011). *Product Release Milestone*. Retrieved from http://www.upedu.org/process/itrwkfls/ms_pr.htm
- Pencil Project for GUI Evolus Software*. (n.d.). Retrieved from <http://pencil.evolus.vn/>
- Schach, S. (2004). *Classical and Object-Oriented Software Engineering 6th Edition*. WCB McGraw Hill, New York.
- Scott W. Ambler, M. L. (n.d.). *Agile Modeling Best Practices*. Retrieved from <http://www.agilemodeling.com/essays/bestPractices.htm>
- Screenshot Graph and Chart Retrieved from QAD JIRA Project Tracking*. (n.d.).
- UML Diagrams*. (n.d.). Retrieved from <http://www.uml-diagrams.org/package-diagrams-overview.html>.