Alabama Liquid Snake

University of Pretoria

Epi-Use

Botic - Privacy aware chatbot
Process and Methodology

Justin Grenfell - u16028440
Peter Msimanga - u13042352
Alicia Mulder - u14283124
Kyle Gaunt - u15330967
Lesego Mabe - u15055214

Contents

1 Introduction

1.1 Agile Unified Methodology

Using the Agile Unified Methodology to take advantage of agile principles as well as work with a methodology based off the waterfall process, that has a simple linear and uncomplicated progression is what we have chosen to do. We anticipate that the project will not have a high requirements change throughout development, and this makes the waterfall process derivative methodology more fitting.

The Agile Unified Methodology makes use of Test-Driven Development, which makes it easier to develop high quality code.

At the moment, phase 3 has high priority.

2 Planning Phase

2.1 Acquiring Requirements

2.1.1 Specific Functional Requirements and Constraints

2.1.1.1 User Interface

R1.1 The system must allow a customer to enter a query and click on a button to send it.

R1.2 The system must warn a customer of personal information included in a query.

R2.2 The system must be able to highlight personally identifying information according to severity index.

R3.3.2.2 The system must be able to highlight personally identifying information according to severity index.

R3.3.2.3 The system must be able to warn the client representative if they have entered identifying information.

R4.1 The system must allow customers to thumbs up a query response.

R4.2 The system must allow customers to thumbs down a query response.

R5 The system must allow queries to be sent to customer support representatives if not answered satisfactorily.

C1 The system must use an Angular Single Page Application for the user interface.

2.1.1.2 Information Scraper

R2.1 The system must be able to attach a severity to the personally identifying information.

R3.1 The system must scrape its customer query responses for personal information.

R3.3.2.1 The system must be able to identify personal information in a customer representative's response.

C3.1 The system must use word2vec for identifying personal information in customer queries.

C5.2 The system must determine if the response contains personally identifying information.

2.1.1.3 Query Classification

R3.2 The system must be able to classify the user queries.

C3.2 The system must use word2vec for classifying customer queries.

2.1.1.4 Response Generation

R3.3.1 The system must generate a response if it certain that it can.

C5.1 The system must generate an automated response based on the query classification.

2.1.1.5 Chatbot

R1.3 The system must be able to recieve customer queries.

R3.3.2 The system must be able to send the query to a customer support representative if it cannot obtain an appropriate response.

R3.4 The system must be able to send a query response back to a customer.

R4.3 The system must be able to recieve customer feedback.

R5 The system must allow queries to be sent to customer support representatives if not answered satisfactorily.

R8 The system must interface with the currently existing ticket system.

C2 The system must provide an API for the SPA to interact with.

2.1.1.6 Chatbot Trainer

R6.1 The system must store previous customer interactions with positive feedback.

R7 The system must must be trained with previous customer queries and responses.

C4 The system must use Machine Learning or Deep Neural Networks in order to be trained with previous customer queries and responses.

2.1.1.7 Data Persistence

R9.1 The system must scrape customer interation data for personal information before storing.

2.1.2 Functional Clusters and Functional Subsystems

Functional Cluster	Functional Description	System Requirements	Function Subsystem Identified
User Interface	This functional cluster allows customers and customer support representatives to interact with the system	R1.1, R1.2, R2.2, R3.3.2.2, R3.3.2.3, R4.1, R4.2, R5, C1	User Interface Subsystem
Personal Information Identification	This functional cluster identifies personal information within messages	R2.1, R3.1, R3.3.2.1, C3.1, C5.2	Message Scraper Subsystem
Classification of Queries	This functional cluster is responsible for classifying queries.	R3.2, C3.2	Query Classification Subsystem
Automatic Query Response Generation	This functional cluster is responsible for generating query responses	R3.3.1, C5.1	Response Generation Subsystem
Main Program	This function cluster is responsible for coordinating query handling subsystems	R1.3, R3.3.2, R3.4, R4.3, R5, R8, C2	Chatbot Subsystem
ChatBot Training	This functional cluster is responsible for training the system using previous interactions	R6.1, R7, C4	Chatbot Trainer Subsystem
Data Persistence	This functional cluster is responsible for persisting customer interactions	R9.1	Database Subsystem

2.1.3 Traceability Matrix

R	UI	Info Scraper	Query Classification	Resp Generation	Chatbot	Chatbot Trainer	Data Persistence
R1.1	X						
R1.2	X						
R1.3				X			
R2.1		X					
R2.2	X						
R3.1		X					
R3.2			X				
R3.3.1				X			
R3.3.2					X		
R3.3.2.1		X					
R3.3.2.2	X						
R3.3.2.3	X						
R3.4					X		
R4.1	X						
R4.2	X						
R4.3	X						
R5	X				X		
R6.1						X	
R7						X	
R8					X		
R9.1							X
C1	X						
C2					X		
C3.1	X						
C3.2			X				
C4						X	
C5.1				X			
C5.2		X					

Key: UI = User Interface, Info Scraper = Information Scraper, Resp Generation = Response Generation.

2.1.4 Nonfunctional Requirements/Quality Attributes

2.1.4.1 Security Requirements

R1.1. The system must be able to authenticate users and authorize them to access system features.

R1.1.1. The system must be able to identify and authenticate customers.

R1.1.2. The system must be able to identify and authenticate customer support representatives.

R1.1.3. The system must be able to deny users who haven't been authenticated to access system features

R1.2. The system must be able to allow new users to register for user profiles for authentication.

R1.3. The system must be able to allow users to update their password.

R1.4. The system must ensure that confidentiality of customer and customer support representative interactions are ensured and maintained across the system.

R1.4.1. The system must ensure that customers can interact with the system in a secured manner.

R1.4.2. The system must ensure that customer queries are sent in a secured manner.

R1.4.3. The system must ensure that customer support representatives care interact with the system in a secured manner.

R1.4.4. The system must ensure that customer support representative response are sent in a secured manner.

R1.4.5. The system must ensure that all queries and responses are processed in a secured manner.

R1.5. The system must ensure that information disclosed during error management is not revealing of internal architecture, design, and configuration information.

2.1.4.2 Availability

R1.1. The system must have high availability to handle customer queries.

R1.1.1. The system should be available at least 99 percent of the time.

R1.2. The system must ensure that errors that occur throughout the system are handled appropriately and provide sufficient information.

R1.2.1. The system must provide error messages when errors occur.

R1.2.2. The system must ensure to keep a traces that show what led to errors.

R1.3. The system must ensure that errors are localized and that their effect is minimized throughout the system.

2.1.4.3 Reliability

R1.1. The system must ensure that responses to customer queries are done in a reliable manner.

R1.1.1. The system must ensure that customer support representative are authorized to respond to customer queries.

R1.1.2. The system must ensure that queries are responses sent throughout the system are complete and consistent.

R1.2. The system must ensure that it is at least 80 percent certain that an autogenerated response is correct before responding to a query.

2.1.4.4 Performance

R1.1. The system must ensure that personal information is highlighted according to severity in real-time.

R1.1.1. The system must ensure that a severity of a word is recieved within a second of it being typed.

R1.1.2. The system must ensure that a word or set of words containing personal information are highlighted in less than a second after recieving the severity.

2.1.4.5 Maintainability

R1.1. The system must allow for system changes and modifications to the user interface to be made as seemlessly as possible.

R1.2. The system must allow for system changes to the database to be made as seemlessly as possible.

2.2 Deriving Use Cases from Requirements

modeling and anaylsis of misuse cases - role based access rights - nonfunctional requirement associations i.e. security requirements, should be associated using the requirement-use case traceability matrix

Using the steps as defined in [1] page 176 to 192, namely: identifying use cases, specifying use case scopes, visualizing use case contexts, reviewing the use cases and diagrams, and finally allocating the use cases to iterations.

2.2.1 Identifying Use Cases

2.2.1.1 Deriving Use Cases, Actors, and Subsystems

Verb-Noun Phrase	Is it a business process?	Does it begin with an actor?	Does it end with the actor?	Does it accomplish a business task for the actor?	Is it a use case?	Actor	Subsystem
Startup System	Y	Y	Y	Y	Y	Admin	Botic
Shutdown System	Y	Y	Y	Y	Y	Admin	Botic
Ask Bot Assistance	Y	Y	Y	Y	Y	Customer	Botic
Ask Human Assistance	Y	Y	Y	Y	Y	Customer	Botic
Respond to Query	Y	Y	Y	Y	Y	Customer Support Rep	Botic
Provide Feedback	Y	Y	Y	Y	Y	Customer	Botic
Train with Interactions	Y	Y	Y	Y	Y	Admin	Botic
Provide Historic Interactions	Y	Y	Y	Y	Y	Admin	Botic
Login	Y	Y	Y	Y	Y	Customer	Botic
Logout	Y	Y	Y	Y	Y	Customer	Botic
Login	Y	Y	Y	Y	Y	Customer Support Rep	Botic
Logout	Y	Y	Y	Y	Y	Customer Support Rep	Botic
Register	Y	Y	Y	Y	Y	Customer	Botic
Register	Y	Y	Y	Y	Y	Customer Support Rep	Botic
Deregister	Y	Y	Y	Y	Y	Customer	Botic
Register User	Y	Y	Y	Y	Y	Admin	Botic
Deregister User	Y	Y	Y	Y	Y	Admin	Botic
Update Password	Y	Y	Y	Y	Y	Customer	Botic
Update Password	Y	Y	Y	Y	Y	Customer Support Rep	Botic
Update Password	Y	Y	Y	Y	Y	Admin	Botic
Login	Y	Y	Y	Y	Y	Admin	Botic
Logout	Y	Y	Y	Y	Y	Admin	Botic

2.2.1.2 Rearranging Use Cases Among Subsystems

Using role-based partitioning we produce the following use case groupings:

Botic/Admin	Botic/Customer	Botic/Customer Support Rep
UC1: Startup System	UC3: Ask Bot Assistance	UC5: Respond to Query
UC2: Shutdown System	UC4: Ask Human Assistance	UC11: Login
UC7: Train with Interactions	UC6: Provide Feedback	UC12: Logout
UC8: Provide Historic Interactions	UC9: Login	UC14: Register
UC16: Register User	UC10: Logout	UC19: Update Password
UC17: Deregister User	UC13: Register	
U20: Update Password	UC15: Deregister	
U21: Login	UC18: Update Password	
U22: Logout		

2.2.1.3 Constructing A Traceability Matrix

A requirements use case traceability matrix is useful for a number of reasons, one them being associating the none functional quality attributes to the use cases and another being to see which use cases are not required as well as to see which requirements are not being satisfied. Here is the Requirements Use Case Traceability Matrix for the system, below:

Requirement	Priority	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10
R1.1	5			X	X						
R1.2	5			X	X						
R1.3	5			X	X						
R2.1	5			X	X	X			X		
R2.2	4			X	X						
R3.1	5			X	X	X					
R3.2	4			X	X	X			X		
R3.3.1	4			X							
R3.3.2	5			X							
R3.3.2.1	5				X						
R3.3.2.2	5					X					
R3.3.2.3	5					X					
R3.4	5					X					
R4.1	3			X			X				
R4.2	3			X			X				
R4.3	3			X			X				
R5	3			X							
R6.1	3			X	X						
R7	4							X			
R8	3										
R9.1	4								X		
UC Priority		5	5	5	5	5	4	4	4	5	5

Requirement	Priority	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19	UC20	UC21	UC22
R1.1	5												
R1.2	5												
R1.3	5												
R2.1	5												
R2.2	4												
R3.1	5												
R3.2	4												
R3.3.1	4												
R3.3.2	5												
R3.3.2.1	5												
R3.3.2.2	5												
R3.3.2.3	5												
R3.4	5												
R4.1	3												
R4.2	3												
R4.3	3												
R5	3												
R6.1	3												
R7	4												
R8	3												
R9.1	4												
UC Priority		4	4	4	4	4	4	3	4	4	4	4	4

In order to associate the quality attributes to the use cases, these below are the non-functional use case traceability matrixes for the system, below:

Security	Priority	UC1	UC2	UC3	UC4	UC5	UC6	UC7	UC8	UC9	UC10
R1.1.1	5									X	
R1.1.2	5									X	
R1.1.3	5	X	X	X	X	X	X	X	X	X	X
R1.2	5										
R1.3	5										
R1.4.1	5			X	X					X	X
R1.4.2	5			X	X						
R1.4.3	5					X				X	X
R1.4.4	5					X					
R1.4.5	5			X	X	X					
R1.5	5	X	X	X	X	X	X	X	X	X	X
Availability											
R1.1.1	5	X	X	X	X	X	X	X	X	X	X
R1.2.1	5	X	X	X	X	X	X	X	X	X	X
R1.2.2	5	X	X	X	X	X	X	X	X	X	X
R1.3	5	X	X	X	X	X	X	X	X	X	X
Reliability											
R1.1.1	4					X					
R1.1.2	4			X	X	X			X		
R1.2	4			X							
Performance											
R1.1.1	3			X	X	X					
R1.1.2	3			X	X	X					
Maintainability											
R1.1	2	X	X	X	X	X	X	X	X	X	X
R1.2	2								X	X	X
UC Priority		5	5	5	5	5	4	4	4	5	5

Security	Priority	UC11	UC12	UC13	UC14	UC15	UC16	UC17	UC18	UC19	U20	UC21	UC22
R1.1.1	5			X		X			X				
R1.1.2	5	X	X		X					X			
R1.1.3	5					X		X	X	X	X	X	X
R1.2	5			X	X		X						
R1.3	5								X	X	X		
R1.4.1	5			X		X			X				
R1.4.2	5												
R1.4.3	5	X	X		X					X			
R1.4.4	5												
R1.4.5	5												
R1.5	5	X	X	X	X	X	X	X	X	X	X	X	X
Availability													
R1.1.1	5	X	X	X	X	X	X	X	X	X	X	X	X
R1.2.1	5	X	X	X	X	X	X	X	X	X	X	X	X
R1.2.2	5	X	X	X	X	X	X	X	X	X	X	X	X
R1.3	5	X	X	X	X	X	X	X	X	X	X	X	X
Reliability													
R1.1.1	4												
R1.1.2	4												
R1.2	4												
Performance													
R1.1.1	3												
R1.1.2	3												
Maintainability													
R1.1	2	X	X	X	X	X	X	X	X	X	X	X	X
R1.2	2	X	X	X	X	X	X	X	X	X	X	X	X
UC Priority		4	4	4	4	4	4	3	4	4	4	4	

2.2.2 Specifying Use Case Scopes

The specification of use case scopes will help us define where the use cases start and where they end. This specification results high-level use cases from the abstract use cases that were previously defined. Below are the high-level use cases for this system:

UC1. Startup System
TUCBW an administrator user clicking on the "Startup System" button (or enters a command) on the Botic Admin dashboard.
TUCEW the administrator seeing a confirmation message that the system is up along with the configuration of the running system.

UC2. Shutdown System
TUCBW an administrator user clicking on the "Shutdown System" button (or enters a command) on the Botic Admin page.
TUCEW an administrator seeing a confirmation message that the system has been shut down.

UC3. Ask Bot Assistance
TUCBW a customer typing a message into the input box in the customer chat page.
TUCEW a customer recieving a "Session has ended" message.

UC4. Ask Human Assistance
TUCBW a customer clicks on the "Ask Human" button in the customer chat page.
TUCEW a customer recieving a "Session has ended" message.

UC5. Respond to Query
TUCBW a customer support representative clicking on a customer query in the customer support chat page.
TUCEW a customer support representative recieves a "Query Resolved" message.

UC6. Provide Feedback
TUCBW a customer clicking either the thumbs up or thumbs down buttons in the customer chat page.
TUCEW when the feedback button (either thumbs up or thumbs down) changes color; in the case that it is thumbs down, the "Ask Human" button should appear.

UC7. Train with Interactions
TUCBW an administrator clicks the "Train AI" button in the Botic Admin page.
TUCEW the administrator sees the message "AI Has Been Trained" accompanied with the report.

UC8. Provide Historic Interactions
TUCBW an administrator clicks the "Load Historic Data" button in the Botic Admin page.
TUCEW the administrator sees the message "Data has been Recieved" accompanied with the report.

UC9. Login
TUCBW a customer clicks the "Sign In" button on the home page.
TUCEW a customer gets redirected to the customer chat page (and it is opened).

UC10. Logout
TUCBW a customer clicks the "Sign Out" button on the customer chat page.
TUCEW a customer gets redirected to the home page.

UC11. Login
TUCBW a customer support representative clicks the "Sign In" button on the home page.
TUCEW a customer support representative gets redirected to the customer support chat page.

UC12. Logout
TUCBW a customer support representative clicks the "Sigh Out" button on the customer support chat page.
TUCEW a customer support representative gets redirected to the home page.

UC13. Register
TUCBW a customer clicks on the "Register" button on the home page.
TUCEW a customer gets the message "You have been Registered" and gets redirected to the home page.

U14. Register
TUCBW a customer support representative clicks on the "Register" button on the home page.
TUCEW a customer support representative recieves a notification with the message "You have been Approved."

U15. Deregister
TUCBW a customer clicks on the "Deregister" button on customer chat page.
TUCEW a customer recieves "Customer has been successfully deregistered" message and is redirected to the home page.

U16. Register User
TUCBW an Admin clicks on the "Register User" button on the Botic Admin page.
TUCEW the Admin receives the message "User Has Been Registered."

U17. Deregister User
TUCBW an Admin clicks on the "Deregister User" button on the Botic Admin page.
TUCEW the Admin recieves the message "User Has Been Deregistered."

U18. Update Password
TUCBW the customer clicks the "Update Password" link on the customer chat page.
TUCEW the customer recieving the message "Password Updated" and being redirected to the home page.

U19. Update Password
TUCBW the customer support representative clicks the "Update Password" link on the customer support chat page.
TUCEW the customer support representative recieves the message "Password Updated" and being redirected to the home page.

UC20. Update Password
TUCBW the administrator clicking the "Update Password" link on the Botic Admin page.
TUCEW the administrator recieving the message "Password Updated" and being redirected to the home page.

UC21. Login
TUCBW the administrator clicking the "Sign In" button on the home page.
TUCEW the administrator being redirected to the Botic Admin page.

UC22. Logout
TUCBW the administrator clicking the "Sign Out" button on the Botic Admin page. TUCEW the administrator being redirected to the home page.

2.2.3 Visualizing Use Case Contexts

Here a "User" encompasses: Administrator, Customer and Customer Support Representative.

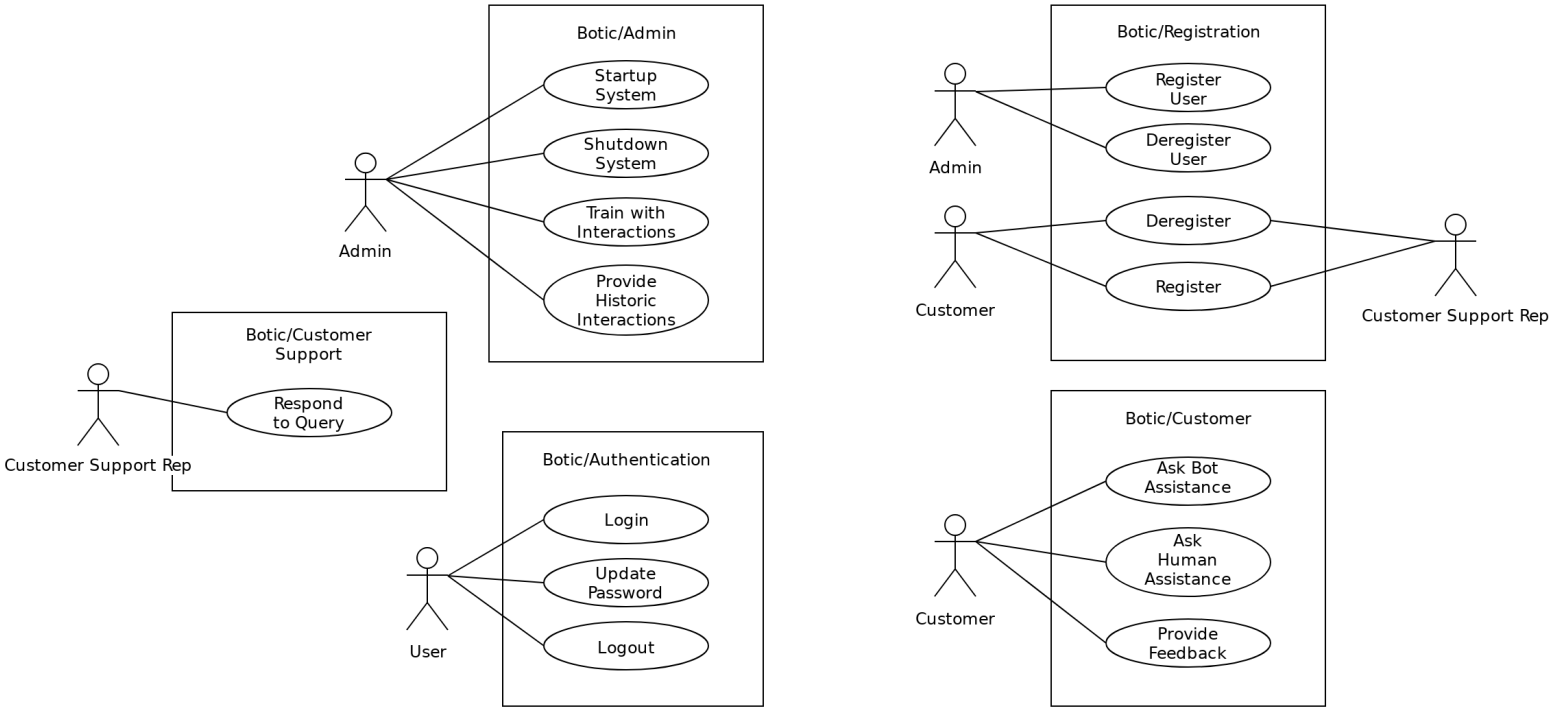

Figure 1: Use Case Contexts

2.2.4 Reviewing Use Case Specifications

All items in the use case review checklist on page 190 in book[1] have been checked, all pass.

2.2.5 Allocating the Use Cases to Iterations

The agile method of effort estimation has been used here. Highest point is 5 and the lowest is 1. Let us assume for now, that the team is capable of handling the amount of work in each use case.

Use Case	Priority	Effort (Agile)	Depend On	Iteration-1 1wks 05/29/19 - 06/03/19	Iteration-2 3wks 06/06/19 - 06/24/19	Iteration-3 2wks 07/08/19 - 07/19/19	Iteration-4 3wks 07/22/19 - 08/09/19	Iteration-5 2wks 08/12/19 - 08/23/19	IT-6 2.5wks 08/26/ - 09/11/
UC1	5	4	None				4		
UC2	5	4	UC1				4		
UC3	5	5	UC9	1	1	1	1	1	
UC4	5	3	UC6				1	2	
UC5	5	3	UC4				1	2	
UC6	4	2	UC3				2		
UC7	4	5	UC8			2	2	1	
UC8	4	4	None				2	2	
UC9	5	2	None	1	1				
UC10	5	1	UC9	1					
UC11	4	2	None	1	1				
UC12	4	1	UC11	1					
UC13	4	2	None		2				
UC14	4	2	None		2				
UC15	4	2	UC13			2			
UC16	4	2	None			2			
UC17	3	2	UC16			2			
UC18	4	2	UC13			2			
UC19	4	2	UC11			2			
Total Effort		50		5	7	13	17	8	

2.3 Allocating Use Cases and Subsystems to Iterations

2.4 Producing an Architecture Design

"The design process needs to be performed recursively down the hierarchy until the leaf node components are relatively easy to design and implement." - [1]

In producing the architecture design of the system we will follow the architectural design process outlined in [1]'s chapter 6.3 as well as additional considerations brought forth by our Software Engineering classes and recommended material in SWEBOK (Software Engineering Body of Knowledge) version and [2]. Agile principles will be followed, thus we intend to cover just enough of what is needed. Here we will cover: Design objectives; System Type; Architectural Style; Subsystem functions, interfaces, and interaction behaviour; and lastly, we will review the architectural design.

2.4.1 Design Objectives

The system quality attributes outline the essense of the business objectives and priorities of the system. Stated in order, they are: Availability, Security, Reliability, Performance, Scalability and Maintainability. Our design objectives are to achieve these quality attributes, in priority, in serving the responsibilities of the system.

In order to achieve these quality attributes architectural tactics, which are "techniques an architect can use to achieve the required quality attributes"[2], will be used. These tactics will then be used to form the architectural design in a manner resembling "first principles."

2.4.1.1 Availability

Looking at the quality attributes involving availability and following the design checklist for availability as specified by [2], we realise that in order to fascilitate the application of tactics that improve availability, we would have to include the following into our system:

1. Logging of faults
2. Notifying appropriate entities of the faults
3. Disable the source of events causing the fault
4. Be temporarily available if need be
5. Fix or mask the fault/failure
6. or Operate in a degraded mode

In order for these to be effected, faults need to be detected, the system needs to be able to recover from faults, and it is appropriate to have a measure of preventing faults before they happen.

In order to log faults, the fault detection mechanism or tactic need to be able to store the detected 'faulty' state. The memento design pattern would clearly be useful here. We can even go as far as to have types of faults being associated with the types of mechanisms set out to identify each. For these, we may apply the Abstract Factory Pattern in order to produce specific fault logs, and store them in our fault logging database through the memento pattern. Event's can be stored in this way, so that we may be able to have a trace of operations that led to an error.

Once that is stored in the caretaker, which in and of itself is a state change, we notify the module responsible for notifying the relevant parties of faults in the system. In order to effect these responsibilities, we would have to assign functionality that stores fault logs, and notifies users. These will be done in reference to use cases, because availability is an attribute that affects users - which are the actors in the use cases. The fault detection mechanisms (architectural tactics) depend on the particular operations of the modules, thus each of the modules or subsystems stand to have their own fault detection mechanisms (availability architectural patterns). This logic applies to tactics responsible for recovering from faults, and preventing faults.

The table below provides the subsystems yeilded from functional clusters for high cohesion, and their system types as well as our chosen fault detection tactic, recovery tactic and fault prevention tactic. These are only subsystems that are necessary, and ordered according to priority. The architectural tactics are solution specific, therefore we expect to add many more of them as the development of the project continues; software engineering is a wicked problem afterall:

Subsystem	Subsystem Type	Fault Detection	Recovery from Faults	Fault Prevention
Chatbot Subsystem	Transformational Subsystem	ping/echo, Retry	Exception Handling	Removal from Service

Please note that the Chatbot Subsystem consists of the Message Scraper, Classifier and the Response Generator subsystems; thus all the mentioned tactics actually apply to each of this subsystem's components.

Two modules will be produced here: one for log management, and another for notification. These require that data be persisted.

2.4.1.2 Security

Looking at the quality requirements involving security in following the design checklist specified by [1], we find that it is useful to use physical security as a model for software security as indicated by [2]. This results in the four categories of focus for this quality attribute: detect, resist, react, and recover.

In order to support the design and the analysis of security in our system we will pay attention to the following, and ensure that these responsibilities are assigned and met:

1. Identify the actor
2. Authenticate the actor
3. Authorize actors
4. Grant or deny access to data or services
5. Record attempts to access or modify data or services
6. Encrypt data
7. Recognize reduced availability for resources or services and inform appropriate personnel and restrict access
8. Recover from an attack
9. Verify checksums and hash values

With regards to our data we shall ensure that the following is observed:

1. Seperation of data of different sensitivities
2. Ensure different access to data of different sensitivities
3. Ensure that access to sensitive data is logged and that the log files is suitably protected
4. Ensure that data is suitably encrypted and that keys are separated from the encrypted data
5. Ensure that data can be restored if it is inappropriately modified

This attribute modifies or enhances modules introduced in the availability requirement analysis above, this includes the assignment of new responsibilities in particular, certain logs have to be stored securely– those logging access to sensitive information. It also makes use of the notification module to let the appropriate personnel know that their system is being attacked, when these attacks are detected.

The table below includes the architectural tactics we will employ with respect to each of the four categories (detection, resistance, reaction, and recovery) we defined earlier, the subsystems and their subsystem types. These are in order of priority:

Subsystem	Subsystem Type	Detection	Resistance	Reaction	Recovery
User Interface Subsystem	Interactive Subsystem	Detect service denial, Detect message delay	Identify actors, Authenticate actors, Authorize actors, Encrypt data, Separate entities	Revoke access, Lock computer, Inform Actors	Maintain Audit Trail
Chabtot Subsystem	Transformational Subsystem	Verify Message Integrity, Detect service denial	Identify actors, Authorize actors, Encrypt data	Revoke access, Inform Actors	Maintain Audit Trail
Chatbot Trainer Subsystem	Transformational Subsystem		Authorize actors, Encrypt data	Inform Actors	Maintain Audit Trail, Data Model checklist above*
Database Subsystem	Database subsystem		Authorize actors, Encrypt data, Separate entities	Inform Actors	Maintain Audit Trail, Data Model checklist above*

Note: we mention the UI subsystem, but what we really mean are the operations performable by the roles of Administrator and Customer Representative. The UI is how they interact with the system.

These are not meant to be exhaustive; they are meant to be revised according to future needs and adapted with feedback.

2.4.1.3 Reliability

All of these concerns are encapsulated in the availability analysis above as well as the security analysis above. The requirements have nonetheless been associated with the revelant use cases.

2.4.1.4 Performance

Here we will look at design decisions that affect the performance of the system. Here we will consider two contributors to the response time: processing time and blocked time[2]. In these our architectural tactics will be concerned with the control of resource demand as well as the management of resources.

A design checklist proposed by [2] suggests useful design considerations for the sake of ensuring performance; other than just following this checklist as in, on pages 142 to 144, we will note especially that under resource management, it is advised that we manage and monitor important resources under normal and overloaded operation; resources such as process and thread models– a detail that involves the availability quality attribute.

The table below includes the architectural tactics that we will employ with respect to the control of resource demand and the management of resources in our system. This will only be for the performance critical components, and ordered by importance. The list contains the following columns: subsystem, subsystem type, control resource demand, and manage resources. This is not an exhaustive list, but alas software engineering is a wicked problem:

Subsystem	Subsystem Type	Control Resource Demand	Manage Resources
Chatbot Subsystem	Transformational Subsystem	Prioritize Events, Increase resource efficiency, Reduce overhead	Increase resources*, Introduce concurrency, Load balancer, Schedule resources
Message Scraper	Transformation Subsystem	Prioritize Events, Increase resource efficiency	Increase resource*, Introduce concurrency, Load balancer, schedule resources

Note: With regards to "Increase resources" we mean to say that a recommended deployment environment ought to be a cloud platform such as PaaS (Platform as a Service) which would dynamically increase or reduce resources based on demand and bill the customer accordingly.

2.4.1.5 Scalability

Looking into design decisions affecting scalability, we have to consider the impact of adding or of removing resources, and these measures will reflect on associated availability as well as the load that will be assigned to existing and new resources[2]. Here, two kinds of scalability will be dealt with: horizontal scalability (adding more resources to individual units), vertical scalability (adding more resources to individual nodes).

2.4.1.6 Maintainability

Maintainability may encapsulate some aspects of modifiability as well as testability and availability. All three of these will be observed as well as that which is required for maintanence management from [1].

In as far as modifiability is concerned, our architectural tactics in this regard, concern all of our system and its subsystems; they are reducing the size of modules through spliting modules, increasing cohesion by increasing semantic coherence, reducing coupling by encapsulation, using an intermediary, restrict dependencies, refactoring and abstract common services.

Looking at architectural tactics for testability we are mostly going to rely on language specific testing tools, especially those that can allow us to follow state, pause and playback tests. It is also useful to follow documentation. The high priority subsystems, in terms of availability and functional responsibilities are high priorities for testing.

Availability has already been covered above. As far as preparing the system for maintainability is concern, during development we will pay close attention to the application of software design principles as well as the application of software design patterns.

"An aspect of testing in that arena is logging of operational data produced by the system, so that when failures occur, the logged data can be analyzed in the lab to try to reproduce the faults."[2]

2.4.2 Determine System Types

From a top view of the entire system, we have observed that the system is more of an interactive system type. Our system is focused on the customer's interaction with the Chatbot, the customer support's interaction, and the administrator's interaction. This means that the interaction begins and ends with each user. The results in the ordering of all subsystems into a number of layers. This has the consequence of creating a clear and well-documented separation of concerns[2].

The layers that result are the following:

1. The User Interface Layer

2. The Chatbot Layer

3. The Training Layer

4. The Persistance Layer

*This comes from the application of a 4-Tier Architecture.

These four layers we treat as the total system's four main subsystems. Each of these layers, or subsystems, produce exhibit system types of their own. Going into each, we will apply the architectural design process until the we find that the individual leaf node elements are relatively easy to design and implement[1].

Now we have a look at the "User Interface Layer." This layer deals heavily with actor requests and responses; all actors primarily interact with this subsystem in order satisfy business processes. This is clearly an interactive subsystem.

Looking at the Chatbot layer, we observe that the layers in a transformational subsystem. The Support layer, or rather the Training layer we observe a transformational subsystem. The Perisistance Layer, however, is very clearly an object-persistence subsystem.

2.4.3 Applying Architectural Styles

2.4.3.1 Entire System

The whole system adheres to the 4-Tier architectural style. It's layers or subsystems include: The User Interface Layer, the Chatbot Layer, the Support/Training Layer, and lastly the Persistence Layer. This architectural style ensure that all the subsystems and modules can be seperate and thus developed seperately with minimal interaction between them (changes), ensuring adherance to the software design principles of high cohesion and low coupling[2]. This makes our entire system more portable and modifiable (and thus maintainable). Here is the solution:

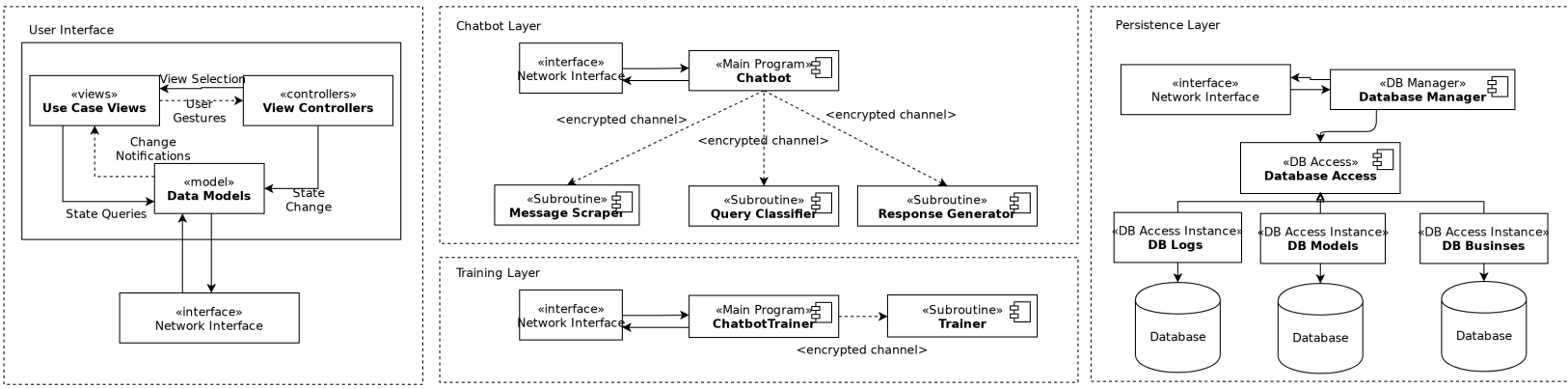

Figure 2: Architectural Stye of the System

The allowed-to-use relation here is denoted by the geometric adjacency of the layers from left to right– layers on the left use layers on the right and the layers on the right are used by the layers on the left ONLY. The connections between each layer are made to be through encrypted channels. An appropriate technology will be used to implement this during the implementation phase e.g. SSL.

2.4.3.2 The User Interface Subsystem

The User Interface Layer that is also an interactive subsystem uses an MVC architectural style. Using this architectural style, we ensure that user interface interface functionality, something that tends to change frequently especially in response to knew design trends *Quote IMY theory notes, be kept seperate from the application functionality and yet be ever responsive to user input, the most important thing in an interactive system[2]. This is the solution:

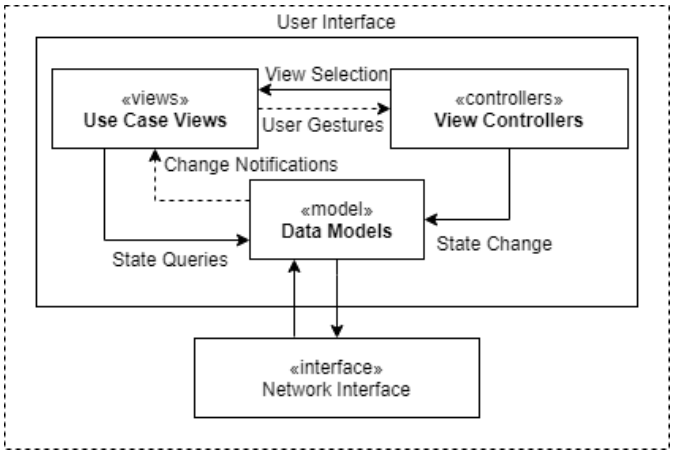

Figure 3: Architectural Style of the User Interface

2.4.3.3 The Chatbot Layer

The Chatbot Layer is a transformational subsystem, and will use a Main Program and Subroutines architectural style.

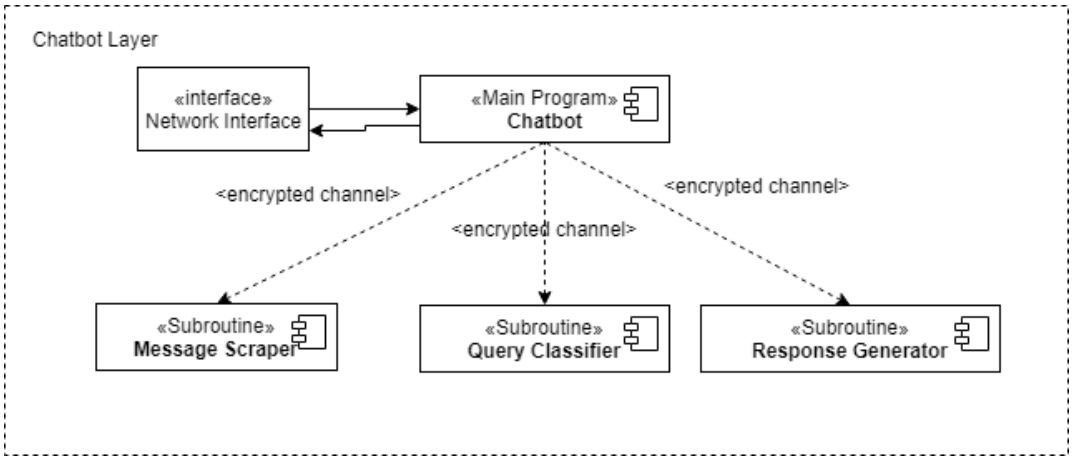

Figure 4: Architectural Style of the Chatbot Layer

2.4.3.4 The Training Layer

This layer is a transformational subsystem and will also use a Main Program and Subroutines architectural style.

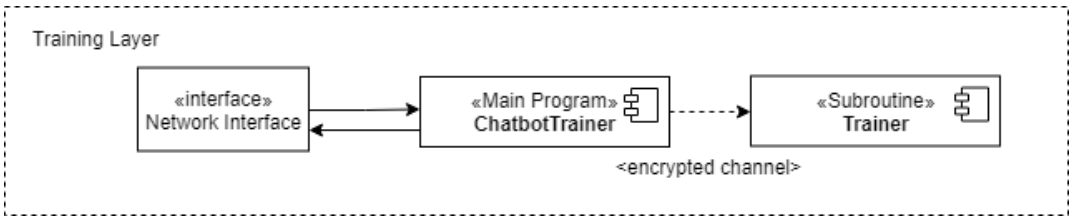

Figure 5: Architectural Style of the Training Layer

The ChatbotTrainer is a single container that has 4 instances: these are to service requests from the user interface (requests which are sent by the Administrator user), one to train the Message Scraper, one to train the Classifier, and one to train the Response Generator. It uses the Strategy Desgin Pattern to switch different training strategies/methods to suit each subsystem it services. This results in multiple instances of the same subsystem, but each using a different training method and data on a different chatbot subsystem.

2.4.3.5 The Persisitance Layer

This layer is a clear candidate for the Persistence Framework architectural style, and it's architectural style is just that. Here is our solution:

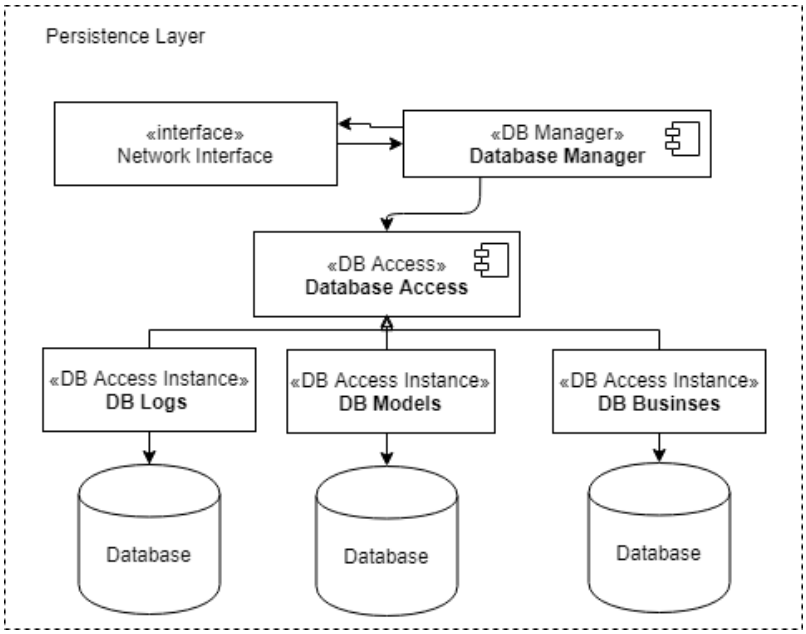

Figure 6: Architectural Style of the Persistence Layer

In accordance to the architectural tactics we have chosen to observe to achieve availability and security quality attributes and requirements, in particular the security tactic to seperate access and data of different sensitivities, we have made the decision to have three databases. The first is for logs and audit trails, the second for the data models what would be used for the chatbot subsystems, and the third for recording queries and responses– data that is concerned with the functional business processes.

3 Iterative Phase

3.1 Phase 1:

3.1.1 Accomodating Requirements Change

3.1.2 Domain Modeling

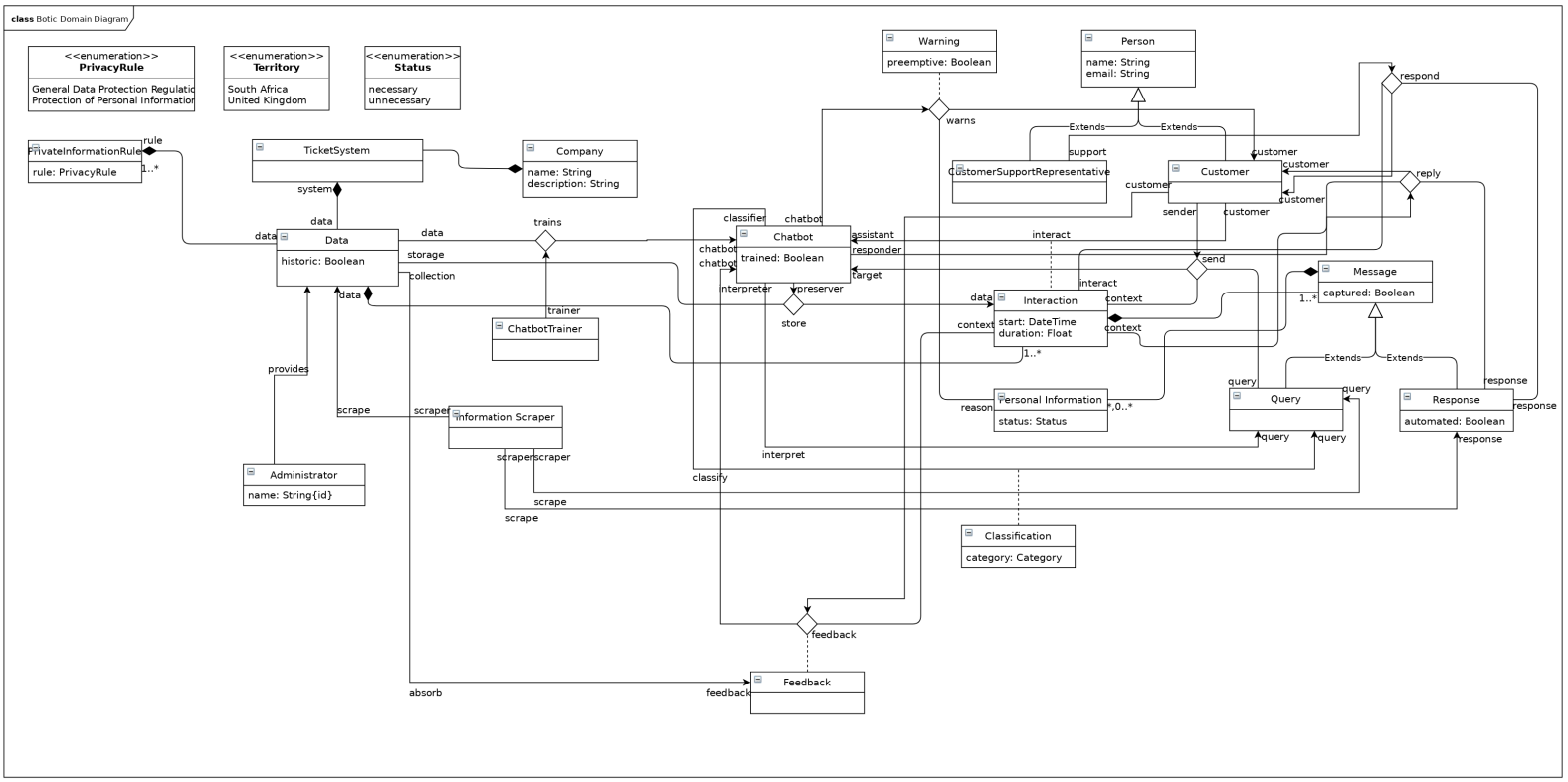

Figure 7: UML Domain Model of the Botic System

3.1.3 Actor-System Iteraction Modeling and User Interface Design

3.1.4 Behavior Modeling and Responsibility Assignment

3.1.5 Deriving Design Class Diagram

3.1.6 Test-Driven Development

3.1.7 Integration

3.1.8 Deployment

3.2 Phase 2:

- Demo 2 happened here - Phase 2 artifacts to be listed here.

3.2.1 Accomodating Requirements Change

3.2.2 Domain Modeling

3.2.3 Actor-System Iteraction Modeling and User Interface Design

3.2.4 Behavior Modeling and Responsibility Assignment

3.2.5 Deriving Design Class Diagram

3.2.6 Test-Driven Development

3.2.7 Integration

3.2.8 Deployment

3.3 Phase 3:

3.3.1 Accomodating Requirements Change

- Another meeting with clients (include proper date). - Refined the requirements according to proper rules thus we must refine the use cases. Link to the new SRS. - Updated architectural desgin. - Link the updated specific requirement and the use cases. - List of use cases to implementated (before and after) - No actionable customer feedback

P: Iteration Use Cases - Haven't produce new ones out of necessity. - Placing emphasis on the Chatbot component that is meant to use all subsystem. - Using updated Architecture

3.3.2 Domain Modeling

- Domain Model has been updated.

3.3.3 Actor-System Iteraction Modeling and User Interface Design

3.3.4 Behavior Modeling and Responsibility Assignment

3.3.5 Deriving Design Class Diagram

3.3.6 Test-Driven Development

3.3.7 Integration

3.3.8 Deployment

4 References

[1] D. C. Kung, Object-Oriented Software Engineering An Agile Unified Methodology. McGraw-Hill, 2014.

[2] R. K. Len Bass, Paul Clements, Software Architecture in Practice. Addison-Wesley, 2013.