UNIVERSITY OF NAIROBI

SCHOOL OF COMPUTING AND INFORMATICS

MULTI-AGENTS SYSTEM BASED SOFTWARE ERROR IDENTIFICATION, REPORTING AND FIXING

BY

KENNEDY KIPKEMOI KIRUI

PROJECT REPORT SUBMISSION FOR THE PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF MASTER OF SCIENCE IN COMPUTER SCIENCE

AUGUST 2013

DECLARATION

This project as presented in this report is my original work and has not been presented for any other institutional award.

Sign: _____ Date: _____

Kennedy Kipkemoi Kirui

P58/63885/2011

This project has been submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science at the University of Nairobi with my approval as the university supervisor.

Sign: _____ Date: _____

Dr. Elisha T. Opiyo Omulo

School of Computing and Informatics

University of Nairobi

ABSTRACT

Software systems are being used in various sectors of the economy. Software is never perfect and would always have a defect that need to be passed to the developers to be investigated and then fixed. A numbers of software vendors have incorporated a feature in their software that automatically generates an error report and sends it to the developers. This is not the case for majority of other software that relies solely on the users to report the error. This poses a great challenge on the functionality of the system as some of errors are never reported, while others are reported with incomplete information. In this study, we designed a prototype for a multi agents-based error identification and reporting system that automatically identify errors as they occur on a software and reports them. A multi-agent architecture is presented, involving a team of agents that identify and report the errors. Furthermore the agents automatically download fixes that are already available, and apply them. We tested the system with a number of known errors and all the errors were reported successfully. Further we tested the error fixing component with a number of fixes and all of them were applied successfully. We did an online survey to evaluate the user response to the system and the expected impact of the system to an organization. The data showed that the positive impacts outweigh the negative and the user response was good.

ACKNOWLEDGEMENTS

I would like express my sincere gratitude to the Almighty God for thus far he has helped me. Secondly, a special gratitude goes to my supervisor, Dr. Elisha Opiyo for his invaluable support and guidance in this study. I also appreciate the entire panelists for their comments, advices and support that helped in improving my project. Last but not least, much appreciation goes to my family, for their unyielding support and encouragement during this project.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

1.1 Background

Software systems are being used in various sectors of the economy. Most softwares are never perfect and would always have defects that need to be passed to the developers to be investigated and then fixed. A numbers of software vendors have incorporated a feature in their software that automatically generates an error report and sends it to the developers whenever an error occurs. This is not the case for majority of other software that relies solely on the users to report the error. This poses a great challenge on the functionality of the system as some of errors are never reported, while others are reported with incomplete information. One such software which does not have an automatic error reporting mechanism is IQcare, an EMR developed by FG Company.

IQCare is a data capture and reporting system with patient management tools designed to measure patient outcomes [1]. It can be implemented in a single stand-alone configuration for smaller hospitals and in networked configuration with multiple simultaneous data entrants for higher volume hospitals. The application is highly configurable and provides sophisticated ad hoc and pre-defined reporting capabilities. To date, IQCare has been implemented in over 70 hospitals in Uganda, Kenya, Nigeria and Tanzania [1].

In Kenya, hospitals are located across the country. Each hospital has a data manager and one or more data entry clerks. Data entry clerks are involved in entering data into the EMR while data manager oversees the overall data entry process and the use of the EMR, reporting any issues to FG support staff.

1.2 Problem statement

Error reporting for IQCare system is done manually. Whenever a hospital encounters an error, the data manager needs to send an email to FG support staff, giving the details of the error to aid in the investigation. Once a fix is found, a FG staff will travel to the hospital to fix the issue. If the reported error is not hospital-specific, then the fix will need to be applied in all the other hospitals.

With such a setup, there are three major challenges. The first issue is incomplete error reports. This is a major issue as outlined by [33], [25] and [34]. Whenever a hospital encounters an error, they need to send an error report to FG support staff. These errors are missing important information such as what the user was doing before the system error occurred.

The second one is unreported bugs. A number of errors that occur when using the system are never reported to the FG support staff. System users do not report some of the errors, a times thinking they are the cause of the errors [35]. Some of the unreported errors lead to critical issues such as loss of data. Often, such cases are only reported when they are critical.

Lastly, there is a problem during the deployment of fixes to the hospitals. After finding a fix for a bug at a hospital, the support staff needs to travel to the facility to apply the fix. This might delay for a few days for hospitals that are far. It is even more challenging for those fixes that need to be deployed to multiple hospitals.

1.3. Goal of the study

The main goal of this study is to develop a system that will run alongside the main system, tracking the use of the system, identifying errors and reporting them whenever they occur. The system will also automatically apply the fix, once it is ready.

1.4 Objectives

The following are the objectives of this project:

(a) Project objectives

1) Formulate the system requirements
2) Develop a Multi-agents based prototype system using an appropriate technology, which address the problem of error reporting and fix deployment.

(b) Research objectives

1. Evaluate the developed system by assessing the user reactions to the prototype
2. Assess the expected impact of the system to the organization

1.5 Justification of the study

Software vendors are spending a lot of resources in providing support for their softwares. At the same time, fixing of issues raised by clients take long to be fixed. Agents can be employed to carry out continuous monitoring of software and report any errors that occur. Furthermore, the agents will automatically apply the fixes as soon as they are made available by developers. This ensures real-time reporting as well as well as faster response by developers.

1.6 Expected contributions of the study

The following are the expected contributions of a successful implementation of the system:

1) It provides an efficient way to get client issues, while reducing or even eliminating the to-and-fro between clients and developers.
2) Developers will get the insight into which bugs are the most severe or frequent, allowing them to prioritize bug fixes based on facts, not guesswork.
3) It enables faster response to system issues from the hospitals and delivery of fixes, thus meeting, or even surpassing their expectations
4) It ensures the fixing of errors/bugs before they get complicated.
5) Deploying fixes that cut across multiple hospitals will be done more easily
6) It will lower customer support costs as the constant travel to sites will be minimized

1.7 Assumptions and limitations

These are the assumptions and limitation made concerning this study:

1) We will not be in a position to implement the system in the real life scenario therefore we will have to implement it on a simplified environment which will reflect the real life scenario.

2) The errors that will be reported by the system are those that affect the smooth running of the system, such as error pop ups, software crashing, unhandled exceptions etc.

3) IQCare EMR is currently running in Windows platform only, and therefore the solution that we will develop might not work in other platforms

CHAPTER 2: LITEREATURE REVIEW

2.1 Multi-Agent system

2.1.1 An agent

An agent is anything that can be viewed as perceiving its environment through sensors and acting upon that environment through actuators [2]. [2] further describe an agent as a special software component that has autonomy that provides an interoperable interface to an arbitrary system and/or behaves like a human agent, working for some clients in pursuit of its own agenda [3].

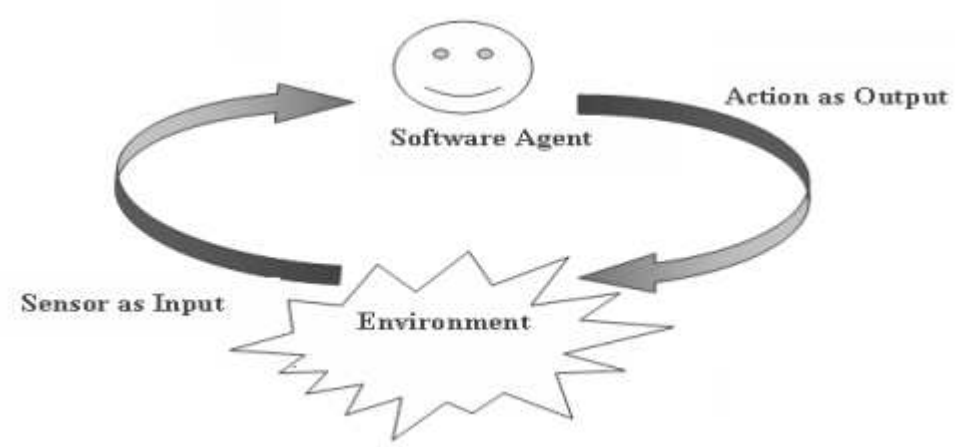


Figure 1: Agent in its environment

2.1.2 Multi-agent system

This is a term used to describe a group of intelligent agents that interact with each other in an environment. The agents are able to operate effectively and interact with each other productively [4]. The environment, in which the agents reside, provides the computational infrastructure for the agents to communicate/interact with one another.

2.1.3 Characteristics of Multi-agent systems

The characteristic of MASs are that (1) each agent has incomplete information or capabilities for solving the problem and thus, has a limited viewpoint; (2) there is no system global control; (3) data are decentralized; and (4) computation is asynchronous.

2.1.4 Strengths of MAS

Multi-agent system has a number of capabilities which include the following, explained by Katia [5]:

1) Ability to solve problems that are too large for a centralized agent to solve because of resource limitations or the sheer risk of having one centralized system that could be a performance bottleneck or could fail at critical times.
2) Ability to allow for the interconnection and interoperation of multiple existing legacy systems.
3) Ability to provide solutions to problems that can naturally be regarded as a society of autonomous interacting components agents. For example, in meeting scheduling a scheduling agent that manages the calendar of its user can be regarded as autonomous and interacting with other similar agents that manage calendars of different users
4) Ability to provide solutions that efficiently use information sources that are spatially distributed. Examples of such domains include sensor networks
5) Ability to provide solutions in situations where expertise is distributed e.g. health care, and manufacturing.
6) It enhances performance along the dimensions of computational efficiency, reliability, extensibility, robustness, maintainability, responsiveness, flexibility and reuse.

2.1.5 Communication and coordination

One of the key components of multi-agent systems is communication. Agents need to be able to communicate with users, with system resources, and with each other if they need to cooperate, collaborate, negotiate and so on. In particular, agents interact with each other by using some special communication languages called agent communication languages, that rely on speech act theory[6] and that provide a separation between the communicative acts and the content language. The first agent communication language with a broad uptake was KQML [6].

2.1.5.1 KQML

Knowledge Query and Manipulation Language is a protocol for exchanging information and knowledge among agents. The elegance of KQML is that all information for understanding

the message is included in the communication itself [7]. It allows message content to be represented in a first-order logic-like language called KIF [8]. The basic protocol is defined by the following structure:

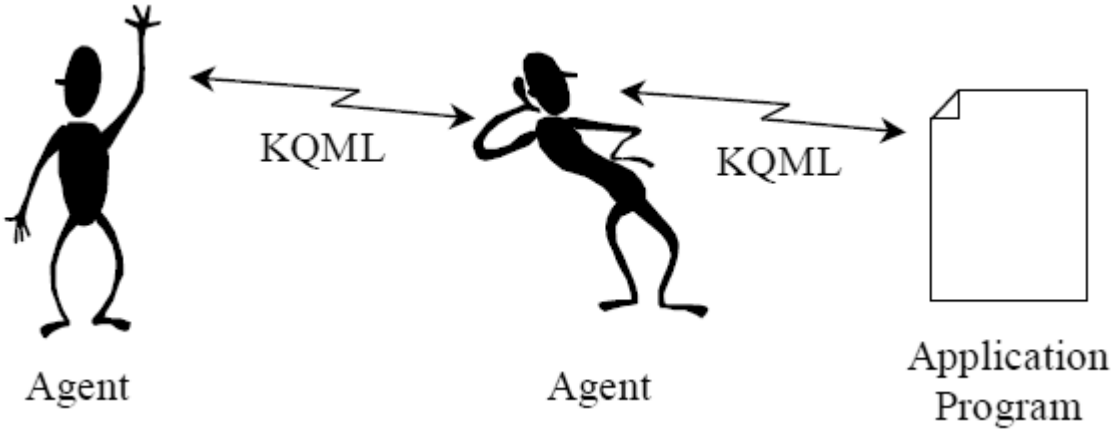


Figure 2: Simple structure of KQML

2.1.5.2 Coordination

Coordination is a process in which agents engage to help ensure that a community of individual agents acts in a coherent manner [9]. [9] gives the following reasons why multiple agents need to be coordinated: (1) agents' goals may cause conflicts among agents' actions, (2) agents' goals may be interdependent, (3) agents may have different capabilities and different knowledge, and (4) agents' goals may be more rapidly achieved if different agents work on each of them.

Coordination among agents can be handled with a variety of approaches including organizational structuring, contracting, multi-agent planning and negotiation.

Organizational structuring provides a framework for activity and interaction through the definition of roles, communication paths and authority relationships [10]. An important coordination technique for task and resource allocation among agents and determining organizational structure is the contract net protocol [10]. This approach is based on a decentralized market structure where agents can take on two roles, a manager and contractor. The basic premise of this form of coordination is that if an agent cannot solve an assigned problem using local resources/expertise, it will decompose the problem into sub-problems and try to find other willing agents with the necessary resources/expertise to solve these sub-problems.

The problem of assigning the sub-problems is solved by a contracting mechanism consisting of: (1) contract announcement by the manager agent, (2) submission of bids by contracting agents in response to the announcement, and (3) the evaluation of the submitted bids by the contractor, which leads to awarding a sub-problem contract to the contractor(s) with the most appropriate bids (see Figure 3).

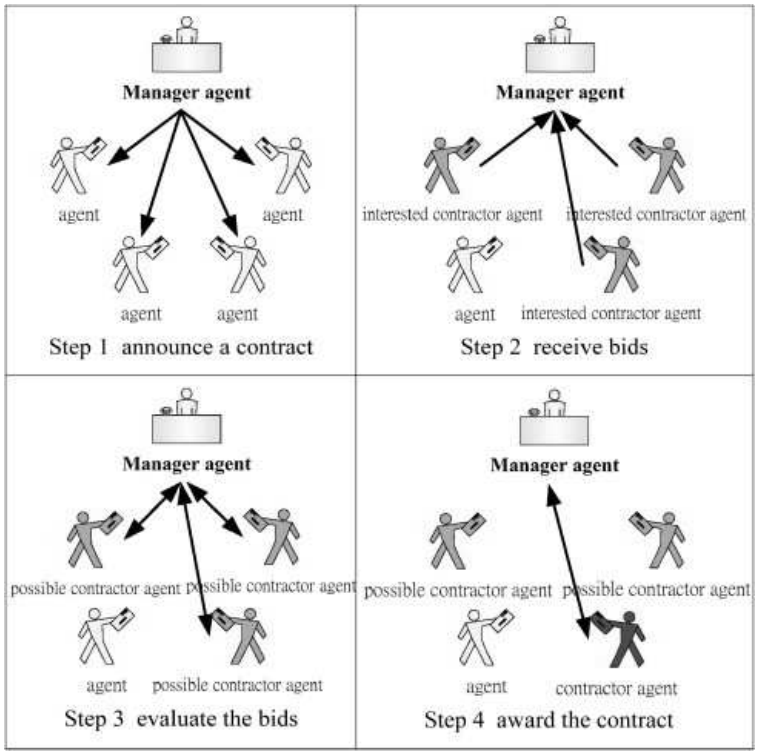


Figure 3: Contract net protocol

2.2 Agent Platforms

2.2.1 JADE

JADE was implemented to provide programmers with the following ready-to-use and easy-to-customize core functionalities [6]. JADE provides a fully distributed system inhabited by agents, each running as a separate thread, potentially on different remote machines, and capable of transparently communicating with one another.

A JADE platform is composed of agent containers that can be distributed over the network. Agents live in containers which are the Java process that provides the JADE run-time and all the services needed for hosting and executing agents [11]. There is a special container, called the main container, which represents the bootstrap point of a platform: it is the first container

to be launched and all other containers must join to a main container by registering with it. The diagram shows a JADE architecture.

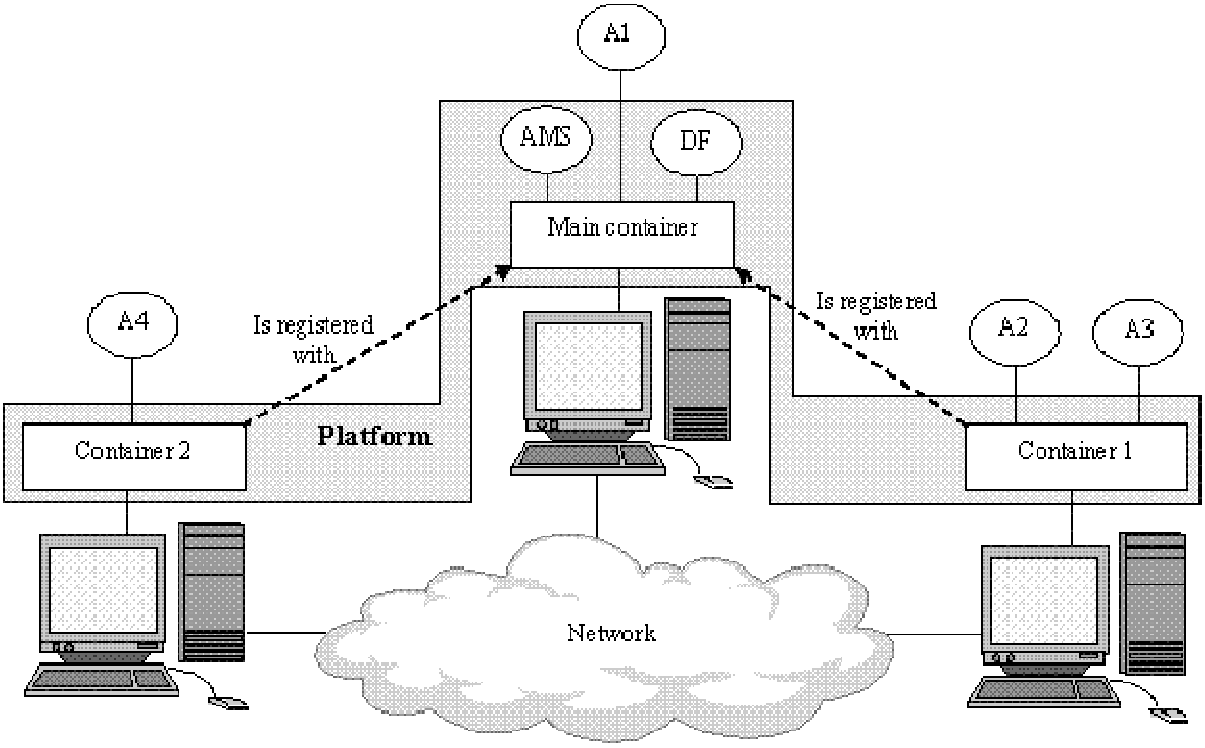


Figure 4: Jade architecture

2.2.2 .NET framework

Some features in .NET framework provide for the development of agents through the combination of units of data and code that can even be moved across machines in a network. This can be achieved through remoting [7]. .NET Remoting is a Microsoft application programming interface (API) for interprocess communication [8].

Microsoft .NET Remoting provides a framework that allows objects to interact with each other across application domains. Remoting was designed in such a way that it hides the most difficult aspects like managing connections, marshaling data, and reading and writing XML and SOAP. The framework provides a number of services, including object activation and object lifetime support, as well as communication channels which are responsible for transporting messages to and from remote applications. [9]

.NET Remoting allows an application to make an object available across remoting boundaries, which include different appdomains, processes or even different computers connected by a network. The .NET Remoting runtime hosts the listener for requests to the object in the appdomain of the server application. At the client end, any requests to the

remotable object are proxied by the .NET Remoting runtime over Channel objects that encapsulate the actual transport mode, including TCP streams, HTTP streams and named pipes. As a result, by instantiating proper Channel objects, a .NET Remoting application can be made to support different communication protocols without recompiling the application [8].

2.3 Agent Development methodologies

2.3.1 GAIA Methodology

This is a methodology that is specifically tailored for the analysis and design of agent-based systems. Its main purpose is to provide the designers with a modeling framework and several associated techniques to design agent-oriented systems.

It has two stages, namely, Analysis and Design. Analysis involves building the conceptual model of the target system. Design transforms the abstract constructs to concrete entities which have direct mapping to implementation code.

GAIA methodology assumes the availability of a requirement specification.

2.3.2 Prometheus Methodology

In this methodology, the agent development process is divided into three phases namely system specification, architectural design and then the detailed design [36].

a) System Specification: At this phase, the system is specified using goals and use-case scenarios. The system's interface to its environment is described in terms of percepts, actions and external data. This phase also involve describing the functionalities of the proposed system.

b) Architectural Design: In this phase, agent types are identified and the overall structure of the system is captured in a system overview diagram.

c) Detailed Design: This is the phase where an agent's internals are defined and developed in terms of capabilities, data, events and plans.

2.3.3 MESSAGE methodology

It comprises [37]:

a) A meta-model extending the UML meta-model and therefore adding new meta-concepts (such as Agent, Goal and Task) to those already considered in UML (e. g. Class, Actor).
b) A set of views focusing each one on specific aspects of the analysis and design models while hiding the complexity of the model as a whole.
c) A number of guidelines and heuristic rules helping the developer in building the analysis and design model and in using them to actually implement the system under development.
d) A proper notation that allows easily and intuitively representing the above views in a graphical (i.e. through diagrams) and textual (i.e. through schemas) Way.

2.3.4 Multi-agent System Engineering Methodology (MaSE)

The MaSE methodology is similar to traditional software engineering methodologies is but specialized for use in the distributed agent paradigm. It has two phases: Analysis and design. The MaSE Analysis phase consists of three steps: Capturing Goals, Applying Use Cases, and Refining Roles. The Design phase has four steps: Creating Agent Classes, Constructing Conversations, Assembling Agent Classes, and System Design.
In this study we are using MaSE methodology for the design and analysis of the system.

2.4 Bug identification and Reporting

2.4.1 Bugs overview

2.4.1.1 Bug (software bug)

Ion *et al* [11] describes a bug as an error, flaw, mistake, "undocumented feature", failure, or fault in a computer program that prevents it from behaving as intended, for example producing an incorrect result. An error occurs when software cannot complete a requested action as a result of some problem with its input, configuration, or environment.

2.4.1.2 Classification of software bugs

Software bugs can be classified according to severity or according to type. Classification according to severity gives the following classes[14]:

1) Catastrophic: These are defects that cause disastrous consequences for the system in question e.g. critical loss of data, critical loss of system availability, critical loss of security, critical loss of safety, etc.
2) Severe: These are defects that cause very serious consequences for the system in question e.g. function is severely broken, cannot be used and there is no workaround.
3) Major: These are defects that cause significant consequences for the system in question. It is a defect that needs to be fixed but there is a workaround e.g. function badly broken but workaround exists
4) Minor: These are defects that cause small or negligible consequences for the system in question. They are easy to recover from or work around them e.g. misleading error messages or displaying output in a font or format other than what the customer desired.
5) No Effect: These are trivial defects that can cause no negative consequences for the system in question. Such defects normally produce no erroneous outputs e.g. simple typos in documentation or bad layout or misspelling on screen.

Padmini C. [13] classifies bugs according to type as follows:

1) User Interface Errors: These are caused by missing or wrong functions, thus the system does not do what the user expects. It is characterized by missing information, misleading, confusing information, wrong content in Help text, inappropriate error messages, and performance issues - Poor responsiveness.
2) Error Handling errors: These are defects caused by inadequate protection against corrupted data, tests of user input, version control etc.
3) Boundary related errors: These are errors caused by exceeding boundaries in loop, space, time, memory, mishandling of cases outside boundary etc.
4) Calculation errors: These are errors caused by bad logic, bad arithmetic, outdated constants, incorrect conversion from one data representation to another, wrong formula, incorrect approximation etc.
5) Initial and Later states not set: There errors are caused by failure to set data item to zero, to initialize a loop-control variable, to re-initialize a pointer, to clear a string or flag, or incorrect initialization.
6) Control flow errors: These errors are caused by stack underflow/overflow, failure to block or un-block interrupts, comparison yielding wrong result, missing or wrong default value, or wrong data-types.

7) Errors in Handling or Interpreting Data: There are caused by un-terminated null strings, or overwriting a file after an error exit or user abort.
8) Race Conditions: This error occurs as a result or racing for resources, for instance a task starts before its prerequisites are met, or messages don't arrive in the order sent.
9) Load Conditions: These errors occur when the required resources are not available e.g. memory space. These errors occur when priority tasks cannot put off less priority tasks, or programs do not return unused memory.
10) Hardware errors: Occur as a result of wrong device, device unavailable, under-utilizing device intelligence, misunderstood status or return code, wrong operation or instruction codes etc.
11) Testing Errors: These are errors that occur as a result of some failure during testing such as failure to check for unresolved problems just before release, failure to verify fixes, failure to provide summary report etc.

2.4.2 Error identification

Bugs that occur in a system need to be identified. One way is by looking at the system event logs. Windows logs store events from legacy applications and events that apply to the entire system[15]. There are three categories of logs: Application, security and system logs[15]. Application logs contain events logged by applications or programs. For example, a database program might record a file error in the application log.

Other errors can be identified on the user interface. These include error messages on the user interface of the running program, which can pop up from the system. Such error can be identified by monitoring the text on the open windows, noting any key words that suggest that an error has occurred in the system. Monitoring of screen text can be done in the following ways:

1) Programmatically, there libraries that can be used to get application captions, URLs and other active text on the screen.
2) Use of Optical Character Recognition tools

2.4.2.1 Optical character recognition

Optical Character Recognition (OCR) lets you convert images with text into text documents using automated computer algorithms [16]. To use this means, a screen shot of the error needs to be done, then the error details can be extracted using the OCR.

Example of OCR

Microsoft Office Document Imaging [17]

An OCR component called Microsoft Office Document Imaging comes with Microsoft office 2007. When Microsoft Office is installed, you can add the OCR libraries to your project. Supported image formats are TIFF, multi-page TIFF, and BMP.

Tesseract OCR Engine

It is thought of as one of the most accurate open source OCR engines available [19]. Combined with the Leptonica Image Processing Library it can read a wide variety of image formats and convert them to text in over 60 languages. It was one of the top 3 engines in the 1995 UNLV Accuracy test [19]. It supports output text formatting, hOCRpositional information and page layout analysis. Support for a number of new image formats was added using the Leptonica library. It supports a number of languages.

It can recognize Arabic, English, Bulgarian, Catalan, Czech, Chinese, Danish, German, Greek, Finnish, French, Hebrew, Croatian, Hungarian, Indonesian, Italian, Japanese, Korean, Latvian, Lithuanian, Dutch, Norwegian, Polish, Portuguese, Romanian, Russian, Slovak, Slovenian, Spanish, Serbian, Swedish, Tagalog, Thai, Turkish, Ukrainian and Vietnamese. Tesseract can be trained to work in other languages too [20].

Tess4J

Tess4J is a Java JNA wrapper for Tesseract OCR API [21]. The library provides optical character recognition (OCR) support for[21]:

1) TIFF, JPEG, GIF, PNG, and BMP image formats
2) Multi-page TIFF images
3) PDF document format

Tess4J is being developed and tested with Java 32-bit on Windows and Linux [21].

The Tesseract OCR DLL file, language data for English, and sample images are bundled with the program. Language data packs for Tesseract should be decompressed and placed into the tessdatafolder[21].

2.4.3 Bug (error) reporting

Bug reporting process is a way to elicit feedback from end users on defects and failures that affect them. Systems like Bugzilla or Jira are frequently used to aid this bug reporting process [22]. A high-quality error report allows a user to understand and correct the problem. Unfortunately, the quality of error reports has been decreasing as software becomes more complex and layered [23].

End-users take the cryptic error messages given to them by programs and struggle to fix their problems using search engines and support websites, before finally submitting to developers. Developers cannot fix the errors when they receive an ambiguous or otherwise insufficient error indicator from a black-box software component. Therefore there is need to write a good bug report.

2.4.3.1 How to write a good bug report

Hilton[24] gives the following hints on how to write a good bug report:

1. Be specific: Use the exact same words as the application. If you see something, write it as is. If you click something, write its exact name. For menus: Follow the sequence of menus separated by the '/' character, for example "File / Save As…" For screens, Look at the top of the window and type exactly what is there. For buttons or tabs, Copy and paste the exact text shown. For links: Copy and paste the whole URL including the "http://".
2. Don't ignore error dialogs: Read the error dialog messages as they are very helpful.
3. Describe what was happening before: To reproduce it we need to reproduce the whole workflow, which means we need you to tell us what you were doing before the bug appeared and what the software was doing too.
4. Report the first error you see: Oftentimes, people get so used to an error that they become tuned to ignoring it. So when a new error occurs, they report that as the "first" error they saw. Not true. If a part of a system has failed, the next error may be a result of the first failure and not a real error in itself. We need to know if you ignored a crash before you got this error.
5. Attach or Copy and Paste: Copy and paste whatever you can into the bug report, attach as many screens and files as you can. The more information we have, the more likely we'll find the issue and fix it.
6. Workarounds are Bugs: If you cannot get something done using the expected process, but can with a workaround, you have a bug. Report it. Workarounds cause huge

problems later on so it is best to get the expected process fixed than rely on the workaround.

2.4.3.2 Components of a good bug report

Among the problems experienced by developers, incomplete information is the most commonly encountered [25]. Common problems experienced by developers when going through bug reports from clients include errors in steps to reproduce; bug duplicates; and incorrect version numbers, observed and expected behavior, incomplete information [25] among others. Another issue that developers often seemed challenged by is the fluency in language of the reporter. Most of these problems are likely to lead developers astray when fixing bugs. Therefore there is need to give a good bug report to the developer

A good bug report will need to include the following components [26]:

1) Problem Report Title: The title needs to be clear, concise, succinct and informative. It should include: (1) Build or version of the software or OS on which the problem occurred (2) Verb describing the action that occurred (3) Explanation of the situation that was happening at the time that the problem occurred and (4) In case of a crash or hang, include the symbol name
2) Steps to Reproduce: Describe the step-by-step process to reproduce the bug, including any non-default preferences/installation and the system configuration information. Be very specific and be sure to provide details, opposed to high-level actions.

2.4.3.3 Duplication in bug reporting

The same defect and failure could affect many users. These users could simultaneously or in parallel submit reports describing the same defect. These reports are termed as duplicate bug reports. Bug triagers should not assign these reports to different developers; this would be a waste of effort and a potential of causing conflicting changes being made to a system.

2.4.3.4 Addressing problem of duplicate bugs

To address the problem of duplicate bug reports, in the research community there have been two threads of work[22]:

1. Given a new bug report, return other bug reports that are similar to it.
2. Given a new bug report, classify it as either a duplicate bug report or not.

2.4.4 File sharing

Apart from error reports being send, there might be need to send files such as screen shots, log files or database backups to the developer. Some of these files might be very large, and thus cannot be sent by e-mail. There are a number of file sharing services that can be used to distribute files via links[27]. Some such as MediaFire, RapidShare, ShareFileandYouSendIt are dedicated to sending and hosting large files in a corporate context, while the others such as Box, Dropbox, Google Drive, Minus, SkyDrive and SugarSync are more general, personal-use file-storage services that have mass distribution as an adjunct feature[27].

All of the services mentioned above allow download links to be generated from uploaded files, which makes it easy to distribute them to a mailing list or other group.There are libraries that can be included in a program to enable automatic file sharing and generation of download links.

2.4.4.1 Examples of file sharing services

Dropbox [27]

Dropbox is a file hosting service that offers cloud storage, file synchronization, and client software [42]. It allows users to create a special folder on each of their computers, which Dropbox then synchronizes so that it appears to be the same folder with the same contents regardless of which computer is used to view it. Files placed in this folder also are accessible through a website and mobile phone applications [42].

It was among the first services to offer seamless upload and storage via its client software. All you need to do to sync files to Dropbox is put them in Dropbox's designated folder on a system with the client app, and the sync happens silently in the background [27].

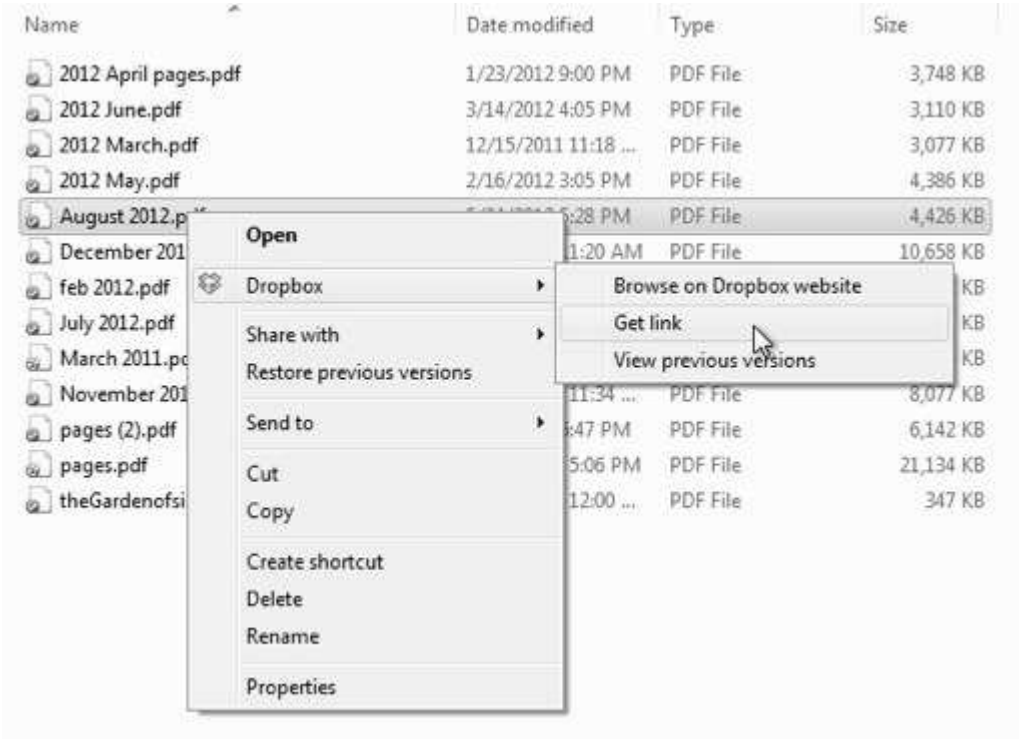


Figure 5: How to get a public link from a file in dropbox

SugarSync

In SugarSync, you can designate existing file folders in your computer to be synced to the cloud and to any other computers you designate. SugarSync also creates a "Magic Briefcase" folder in the Documents folder such that anything placed there is automatically synced across all devices registered to the user's account [27].

A "Web Archive" folder, on the other hand, stores files from devices but does not sync them automatically if the originals are changed. This makes the Web Archive a useful place for files intended mainly to be distributed to others, so they're not replicated unnecessarily [27].

The desktop client also includes a file manager application that lets one to see what files are synced into the cloud and across their devices, all in one place [27].

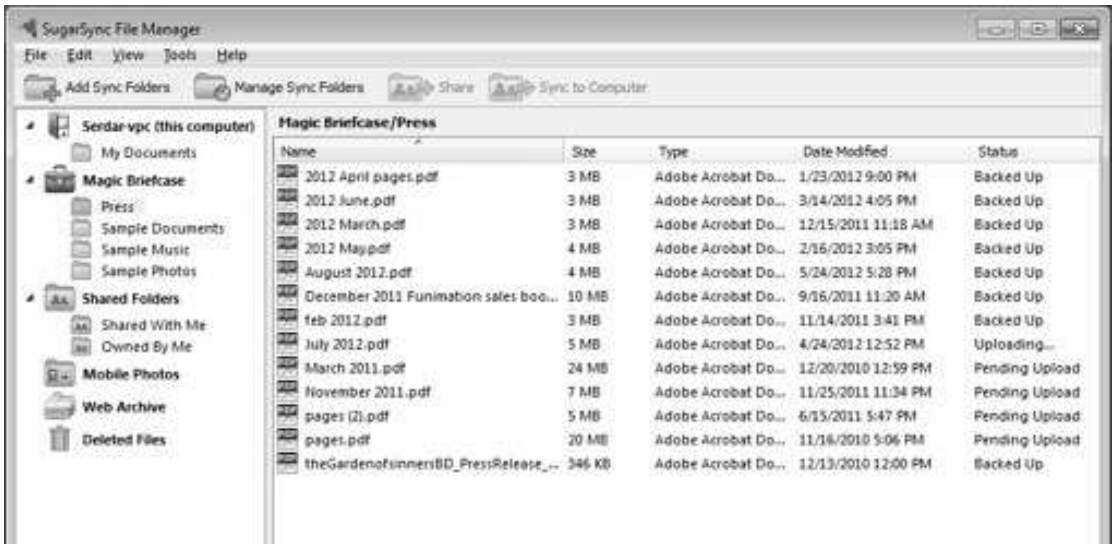


Figure 6: Sugar sync file manager

2.5 Bug tracking

Bug tracking is the process of finding defects in a product by inspection, testing, or recording feedback from customers, and making new versions of the product that fix the defects [28]. In software Engineering, when the numbers of defects gets quite large, and the defects need to be tracked over extended periods of time, use of a defect tracking system can make the management task much easier.

2.5.1 Bug tracking system

A bug tracking system is a software application that is designed to help keep track of reported software bugs in software development efforts [29]. It may be regarded as a type of issue tracking system.

Many bug tracking systems, such as those used by most open source software projects, allow users to enter bug reports directly. Having a bug tracking system is extremely valuable in software development, and they are used extensively by companies developing software products.

2.5.2 Components of a bug tracking system

A major component of a bug tracking system is a database that records facts about known bugs. Facts may include the time a bug was reported, its severity, the erroneous program

behavior, and details on how to reproduce the bug; as well as the identity of the person who reported it and any programmers who may be working on fixing it [30].

A bug tracking system should allow administrators to configure permissions based on status, move the bug to another status, or delete the bug. The system should also allow administrators to configure the bug statuses and to what status a bug in a particular status can be moved. Some systems will e-mail interested parties, such as the submitter and assigned programmers, when new records are added or the status changes.

2.5.3 Examples of bug tracking systems

There are a number of both desktop and web-based bug tracking software tools. The following bug tracking software, among others are described in [31]:

1) Census Bug Tracking Software

Census is a highly scalable web-based issue tracking tool that can track bugs, defects, change requests, support calls, test cases, timesheets, and much more. Features include full customization capabilities, Visual SourceSafe integration, automatic e-mail notifications, user/group/field-level security, role-based workflow rules, custom Web views for different groups of users, built-in reporting, attachments, and change history tracking

2) AceProject

AceProject offers free web-based project management, bug tracking and timesheet software.

3) Bugzilla

This is an open source application, web-based, general-purpose bug-tracker tool originally developed and used by the Mozilla project.

4) Mantis

It is a free web-based bug-tracking system, written in PHP scripting language, which works with MySQL, MSSQL, and PostgreSQL databases and a webserver. It is also distributed with most Linux distributions.

2.6 Fix deployment technologies

The following are ways of applying fixes to existing software applications:

1) Automatic updates

An example is windows updates which is a service provided by Microsoft that provides updates for the Microsoft Windows operating system and its installed components, including Internet Explorer.

2) Manual patching

This is done by manually copying and replacing the DLL, EXE and any other files on the installation folder.

3) ClickOnce Technology

ClickOnce Technology enables applications to check for newer versions as they become available and automatically replace any updated files. Applications can be configured to check for updates on startup or after startup.

4) Use of patch application tools

There are several tools to aid in the patch application process, such as RTPatch, JUpdater, StableUpdate or Visual Patch.

2.7 Related Work

2.7.1 RedGate Automated Error Reporting system

This is an error reporting system that customers use to report error when they occur on RedGate products [41]. It allows automatic sending of an error report when any of their products crashes. A typical error report comprises a full stack trace and details about the exception context, including values of all the local variables.

Another feature is that it allows the customization of the exception dialog to provide additional information that can be packaged with the exception report. Customizations could include a log file or a screenshot taken at the time of the error, or asking end users for contact information so you can notify them when a fix is released [41].

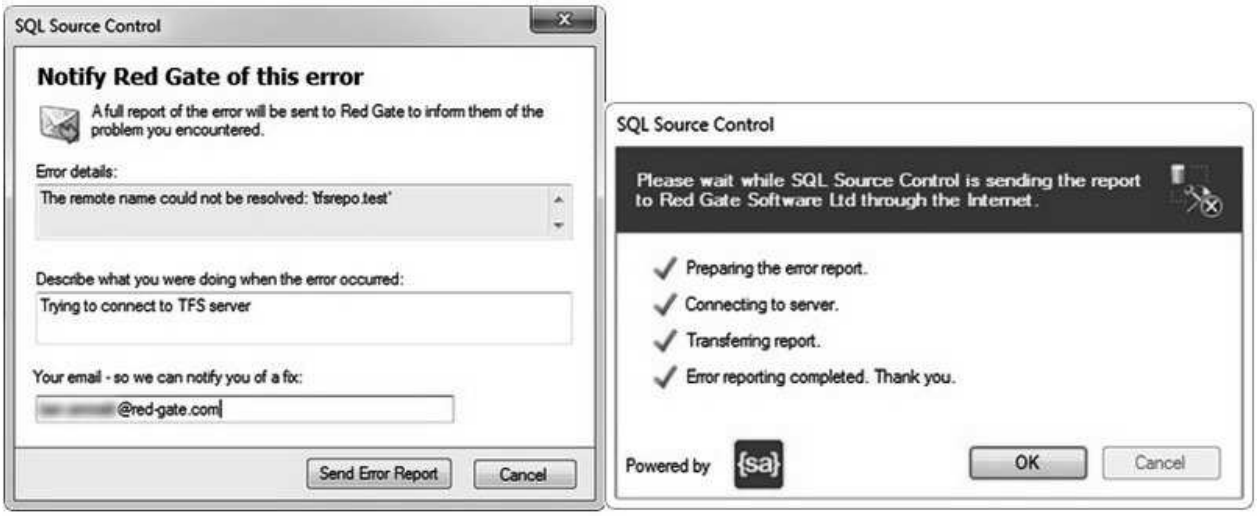


Figure 7: How error is reported for Red Gate's SQL Source Control

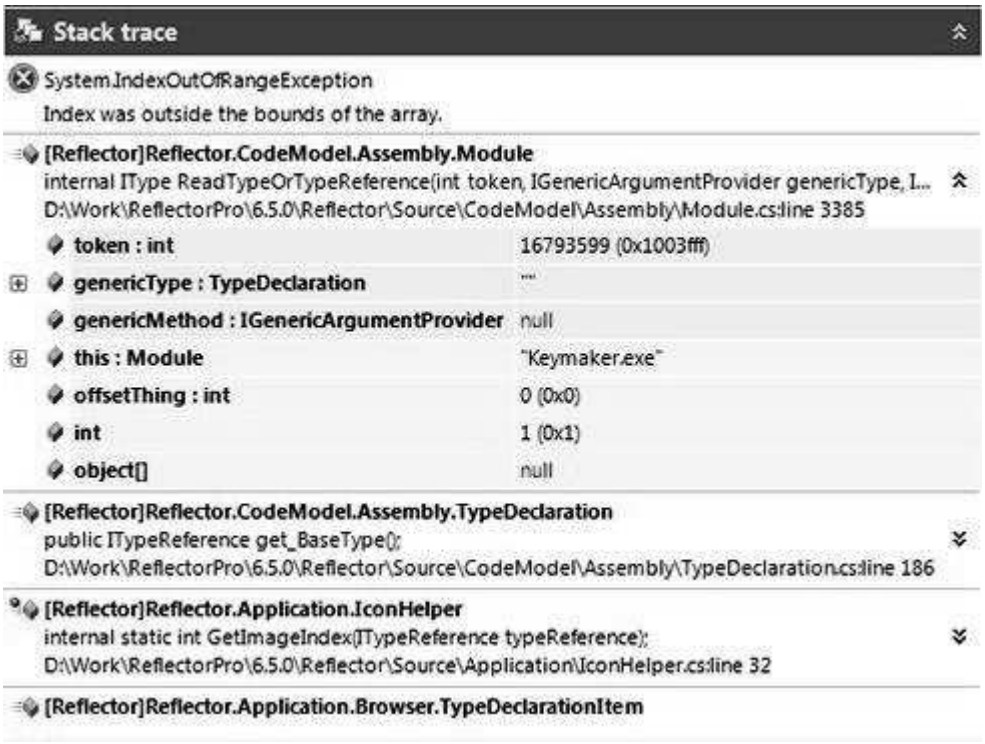


Figure 8: Stack trace of Red Gate's Error reporting system

2.7.2 Other automated error reporting tools

There are other similar or related tools that automatically report system errors. These include the following:

Windows Error Reporting [39]

It is a crash reporting service that prompts users to send crash reports to Microsoft for online analysis. The information goes to a central database run by Microsoft. It consists of diagnostic information that helps the company or development team responsible for the crash to debug and resolve the issue if they choose to do so.

Talkback [39]

Talkback was the crash reporter used by Mozilla software up to version 1.8.1 to report crashes of its products to a centralized server for aggregation or case-by-case analysis. If a Mozilla product (e.g. Mozilla Firefox) were to crash with Talkback enabled, the Talkback agent would appear, prompting the user to provide optional information regarding the crash.

ABRT - Automated Bug-Reporting Tool [39]

ABRT intercepts core dumps from applications and after a user confirmation sends bug report to various bug tracking systems, such as Fedora Bugzilla.

CrashRpt [39]

It is a light-weight open source error handling framework for applications created in Microsoft Visual C++ and running under Windows. It intercepts unhandled exceptions, creates a crash mini-dump file, builds a crash descriptor in XML format, presents an interface to allow user to review the crash report, and finally it compresses and sends the crash report to the software support team.

It also provides a server-side command line tool for crash report analysis named crprober. The tool is able to read all received crash reports from a directory and generate a summary file in text format for each crash report. It also groups similar crash reports making it easier to determine the most popular problems. The crprober tool does not provide any graphical interface, so it is rather cryptic and difficult to use.

2.7.3 Gaps and limitations of the solutions

The following are the gaps and limitations that will be addressed by the proposed solution:

1) Submitting of error fails if there is no active internet connection.
2) The solutions above do not include error fixing

3) The current solutions mainly handle cases of software crashing. There are errors that do not necessarily cause the software to crash, but need to be reported
4) There is need for a generic solution that is independent from the main system, and can be configured to track the error of any software application. The current solutions do not provide such.

2.8 Proposed architecture

For this study, we propose an architecture in which situated agents in the client and server computers will continuously monitor the system. The agents will use system logs, Screenshot outputs and unexpected system behavior to identify errors. On identifying an error, an agent creates an error report, and then sends it to the developer. On the developer's end, the error is logged in a bug tracking system. Once a fix is ready, it will be uploaded in a central public server, where the agents can download it and patch it on their machines.

The proposed architecture will make use of limited internet connection at the facilities, whereby the error is captured and stored in the machine for submission when a connection to the internet is available. It will use file sharing mechanisms such as dropbox to ensure that the errors, screenshots and other files are uploaded without failure.

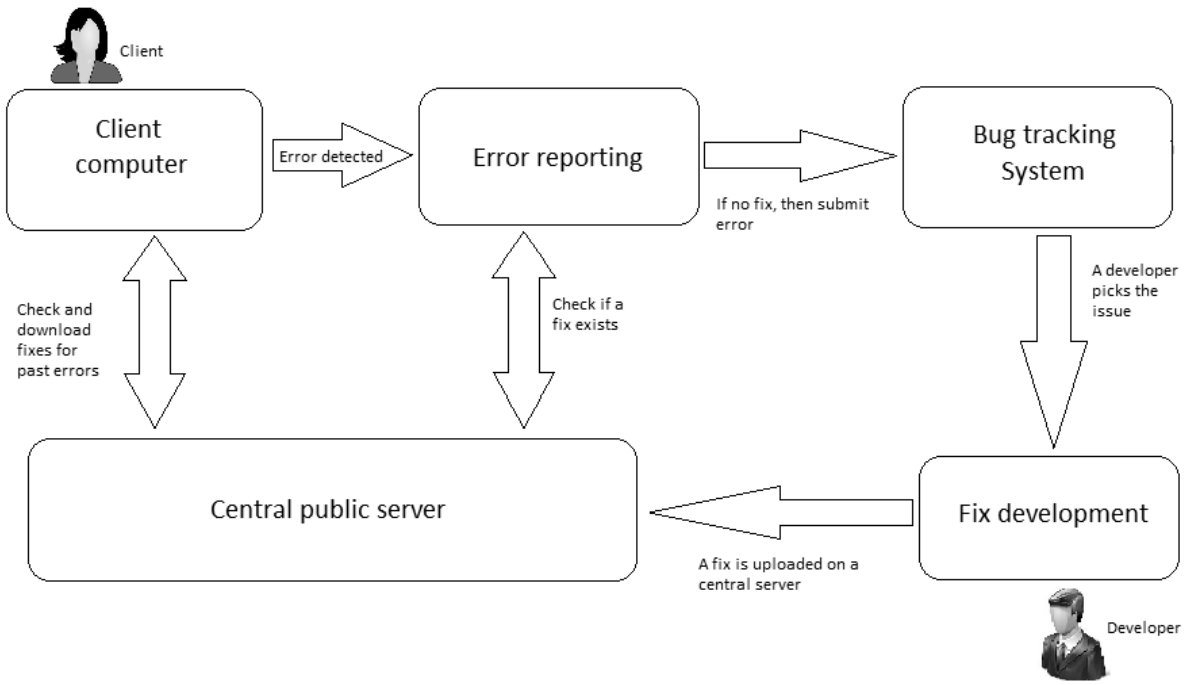


Figure 9: A diagrammatic representation of the proposed architecture

Although the proposed solution is made for IQCare system, it will be a generic solution that can be configured to be used for monitoring any other related system.

CHAPTER 3: METHODOLOGY

3.1 System Design

3.1.1 Overview

We are using MaSE methodology for system design. There are two steps involved: Analysis and design.

3.1.2 Analysis phase

The analysis phase involves the following steps:

1) *Capturing goals*: Here we take an initial system specification and transform it into a structured set of system goals. Then the goals are analyzed and structured into a Goal Hierarchy diagram.
2) *Applying Use Cases*: Here we capture a set of use cases from the initial system context and create a set of Sequence diagrams to help the system analyst identify an initial set of roles and communications paths within the system.
3) *Refining roles*: Here we transform the structured goals and Sequence diagrams into roles. After roles are created, tasks are associated with each role that describes the behavior that the role must exhibit to successfully achieve its goals.

3.1.3 Design phase

The design phase involves the following steps:

1. *Creating agent classes*: Here we create agent classes from the roles defined in the Analysis phase. The end product of this phase is an Agent Class diagram, which depicts the overall agent system organization consisting of agent classes and the conversations between them.
2. *Constructing conversations*: Here we construct conversations by extracting the messages and states defined for each communication path in Concurrent Task Models, adding additional messages and states for added robustness.
3. *Assembling agents:* Here we create the internals of the agent classes. This is accomplished via two sub-steps: defining the agent architecture and defining the components that make up the architecture. The outcome is an Agent Architecture diagram.

4. *System design*: Here we take the agent classes defined previously and instantiates actual agents. We use a Deployment Diagram to show the numbers, types, and locations of agents within a system.

3.1.4 Tools required

To accomplish the goal of this study, the following tools will be needed:

1) Windows OS (At least XP SP2)
2) Visual Studio 2010
3) Relational Database Management Systems (SQL 2008 Express)
4) Microsoft Office 2007
5) Dropbox

3.1.5 Justification of MaSE methodology

A major strength of MaSE is the ability to track changes throughout the process. Every object created during the analysis and design phases can be traced forward or backward through the different steps to other related objects. For instance, a goal derived in the Capturing Goals step can be traced to a specific role, task, and agent class.

Also MaSE supports the development from requirements analysis to implementation

3.1.6 Limitations of MaSE methodology

MaSE methodology has the following weaknesses [40]:

1) Existing gap between analysis and design phases: Therefore one should go back to the first step and analyse the role requirements to gather necessary information for decision making about the appropriate agent architecture.
2) Lack of knowledge modeling in the methodology: modeling the internal knowledge of agents such as rules or plans is ignored and is not addressed.
3) Weak documentation: In the MaSE, documentation of many aspects of an agent-based system is implicit. For example, roles are just documented in sequence diagrams and by their related tasks in the analysis phase. Since a good documentation is needed for maintenance of a system, the methodology should guide software engineer to produce necessary documents and artifacts for better maintenance of the developed system. Explicit documentation for roles helps the methodology to achieve this goal.

4) Problem in modeling interactions: MaSE use UML sequence diagram for modeling role interactions so it cannot model some specific characteristics of agents such as concurrent threads of interactions among roles.

MaSE methodology can be improved and extended by adding a "Role Schema" and a "Knowledge Modeling" step in the analysis phase. Improvement can also be done by the use of AgML instead of UML.

3.2 Research Methodology and design

3.2.1 Overview

Research is done to achieve the following two purposes:

1. To evaluate the expected impact of the system on the organization.
2. To evaluates the developed system by assessing the user reactions to the prototype.

3.2.2 Data sources

The primary source of data is a survey research where the software developers and IT support staff fill an online survey form. Willing participants are asked to participate in the survey.

3.2.3 Data collection procedure

1) Online Survey

The participants were given a link to the online survey form. There were eleven questions to answer. The first ten questions were multi-choice questions with five choices as follows:

1. Strongly disagree
2. Disagree
3. Neither agree not disagree
4. Agree
5. Strongly agree

Users answered the questions by putting a check mark (√) on the selected choice. The last question was an open question where the user was expected to describe their expectations of the proposed system. A copy of the online survey form is included in the appendix section.

The advantage for using the online survey is that a lot of checks can be put in place to ensure that the participants input the correct details. More so, as the participants fill the forms, all data is automatically put in a spreadsheet documents, this reducing the process of compiling the data.

3.2.4 Data analysis

There are two types of data being collected: Qualitative and quantitative data. The data analysis is performed according to research questions. Descriptive statistics is used to answer descriptive questions. Data is collected and put in a spreadsheet (MS Excel).The data is the tabulated and analyzed in form of graphs and charts.

The output of the analyzed data is used to make conclusion on the expected impact of the proposed system.

CHAPTER 4: SYSTEM ANALYSIS AND DESIGN

The analysis and design was guided by the MaSE methodology.

4.1 System specification

4.1.1 Overview

Currently, the reporting of errors is done manually across hospitals in Kenya. The proposed system will automatically identify report and fix errors. The error identification will involve the checking of event logs for errors as well as monitoring screen text. A non-fatal error usually involves a pop up message dialog on the screen showing the error that has occurred. A fatal error might not show a pop up message, but the application will crash and therefore, an entry will be made on the event logs.

The process of error reporting will entail the generation of an error report, based on the error report specification described in chapter 2. The error report will then be send to the developer.

On receiving the error report, he/she will develop a fix and then upload on a public server, where in turn the client application will download, and the install the fix on the client computer.

The process of identification of errors, reporting and fixing is done by agents, forming a multi-agent system.

4.1.2 Inputs and outputs

4.1.2.1 Inputs

This system is tracking an existing system; therefore the name of that system will be input. The system will be tracking errors; it therefore needs the definition of what errors are. A list of past errors that are in the current bug tracking system will be input to the system.

4.2.2.2 Outputs

The system will output a list of identified errors and their status. The status will be either “identified”, “reported”, “fix ready” or “fixed”.

4.1.3 Data management

The data used by the system will be stored in a database. The system will save all the identified errors in database. Likewise, the status of each error will be saved in the database.

The list of past errors described in section 4.1.2.1 above will be saved in database.

4.1.4 System failure

On the event that this prototype system fails, a record will be put in the event log. In addition, the output from the system will indicate the point of failure. The output shows the progress of the identified errors, and if the progresses of the errors stagnate at some point, then it would be an indication of system failure.

4.2 System analysis

The purpose of the MaSE analysis phase is to produce a set of roles whose tasks describe what the system has to do to meet its overall requirements. A role describes an entity that performs some function within the system. In MaSE, each role is responsible for achieving or helping to achieve specific system goals or sub-goals [37].

System analysis involves the following three steps:

1. Identifying goals from user requirements and structuring them into a Goal Hierarchy Diagram
2. Identifying use cases and creating sequence diagrams to help identify an initial set of roles and communications paths
3. Transforming the goals into a set of roles

4.2.1 Identifying goals

This is the first step in the analysis phase, which takes an initial system specification and transforms it into a structured set of system goals.

a) Capturing goals

This process begins by extracting scenarios from the initial specification and describing the goal of that scenario. The following are the scenarios from our initial specification:

1. The system is responsible for identifying, reporting and fixing of errors

2. Error identification will involve monitoring of the target system as it runs, and detecting errors on the screen as they occur. A knowledge base of errors will be used to inform the system on the type of errors to report.
3. An Error reporting component will generate an error report, ensuring that all the components of a good error report are captured.
4. An error will be reported by sending an error file to the developer.
5. If need be, the developer can place a request for the database, and the system will automatically get and send it to the developer
6. The error fixing component of the system will check regularly for any new fixes released by the developer.
7. If a new fix if available, the component will download it, and then install/apply it on the target system.

Goals are then derived from the scenarios. The following are the derived goals:

1. Identify screen errors
2. Identify errors on windows logs
3. Upload captured screenshots
4. Generate an error report
5. Send error report to developer
6. Send database to developer
7. Check if new fixes exist
8. Download new fixes
9. Apply/install the fixes

b) Structure the goals

The goals are put in hierarchies depending on the importance, level or detail. This results in a goal hierarchy diagram.

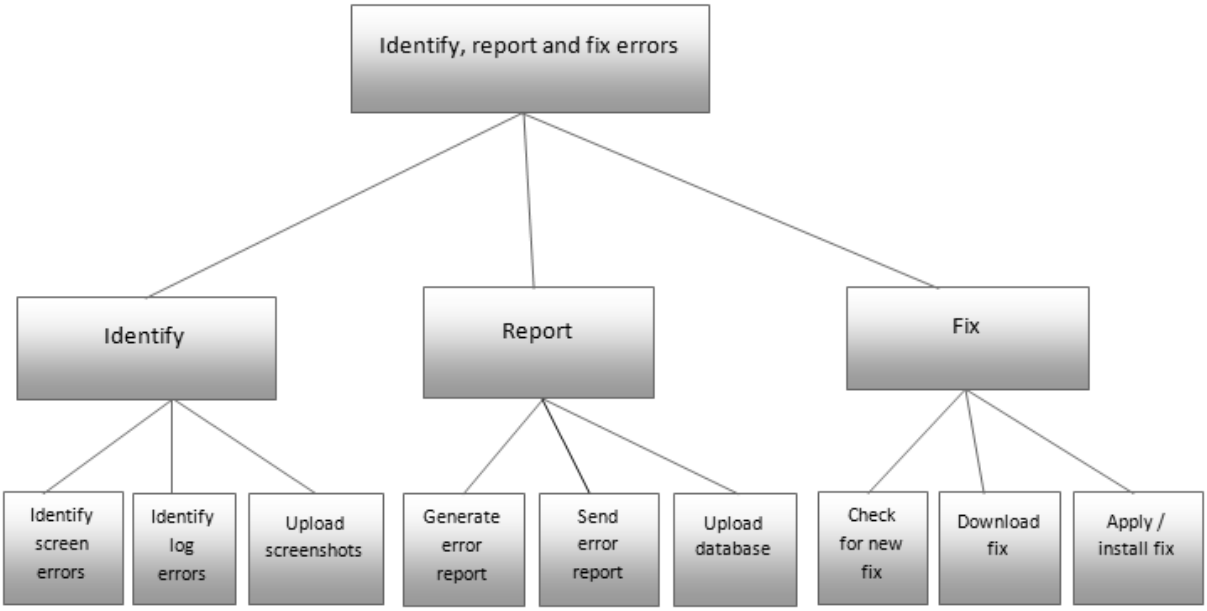


Figure 10: Goal hierarchy diagram

4.2.2 Applying Use Cases

a) Creating use cases

Use cases define a sequence of events that can occur in the system. They are examples of how the user thinks the system should behave. Although part of the Applying Use Cases step, creating use cases may actually elicit more information or clarify existing information about system goals.

The goal of creating use cases is to identify paths of communication, not to define all possible combinations of events and data in the system.

Actors

The main actors are the agents that are responsible for the identification, reporting and fixing of software errors. The other actor is the systems developer, who is the one responsible for developing fixes for the identified errors.

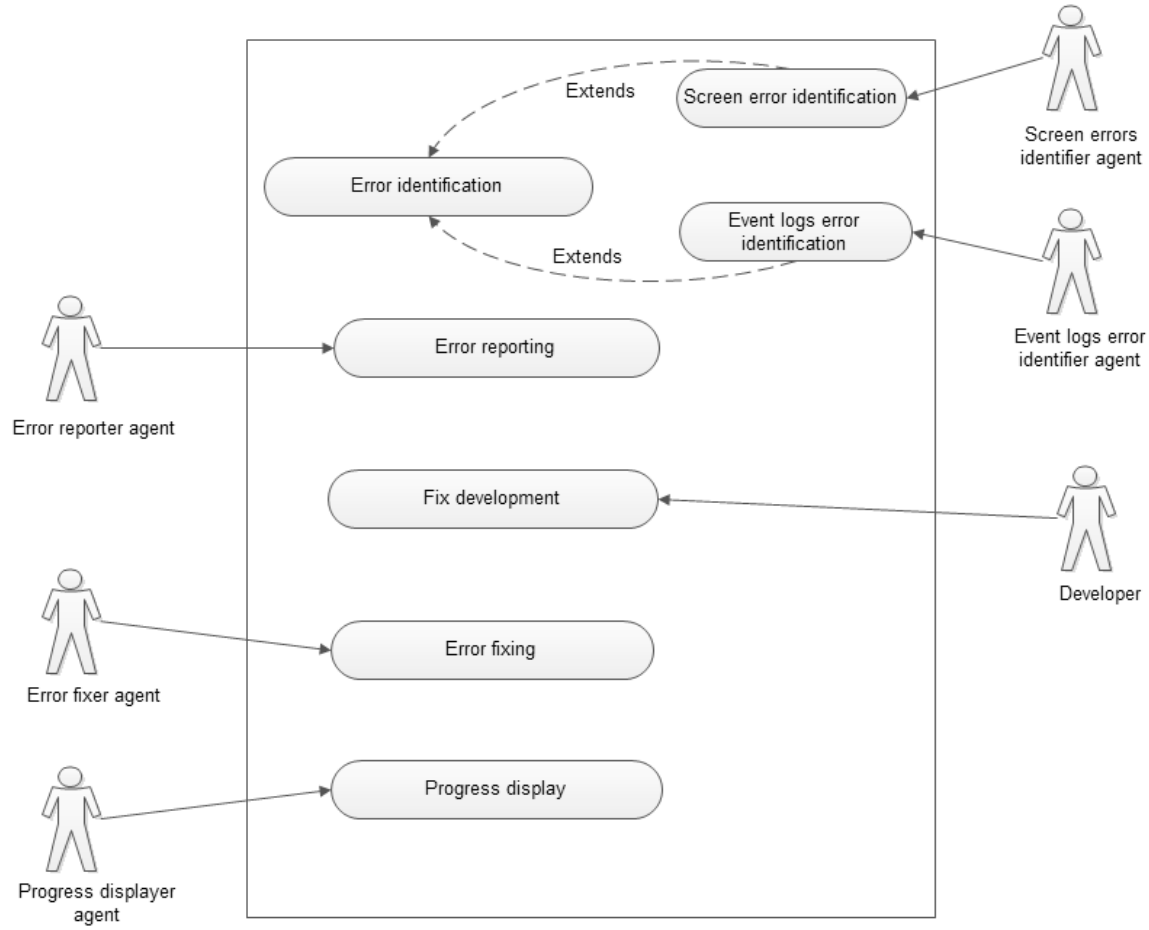


Figure 11: Use cases diagram

4.2.3 Refining roles

The objective of this step is to transform the structured goals and sequence diagrams into roles and their associated tasks. The tasks are generally derived from the goals for which a task is responsible. These are captured in a role model diagram as shown below.

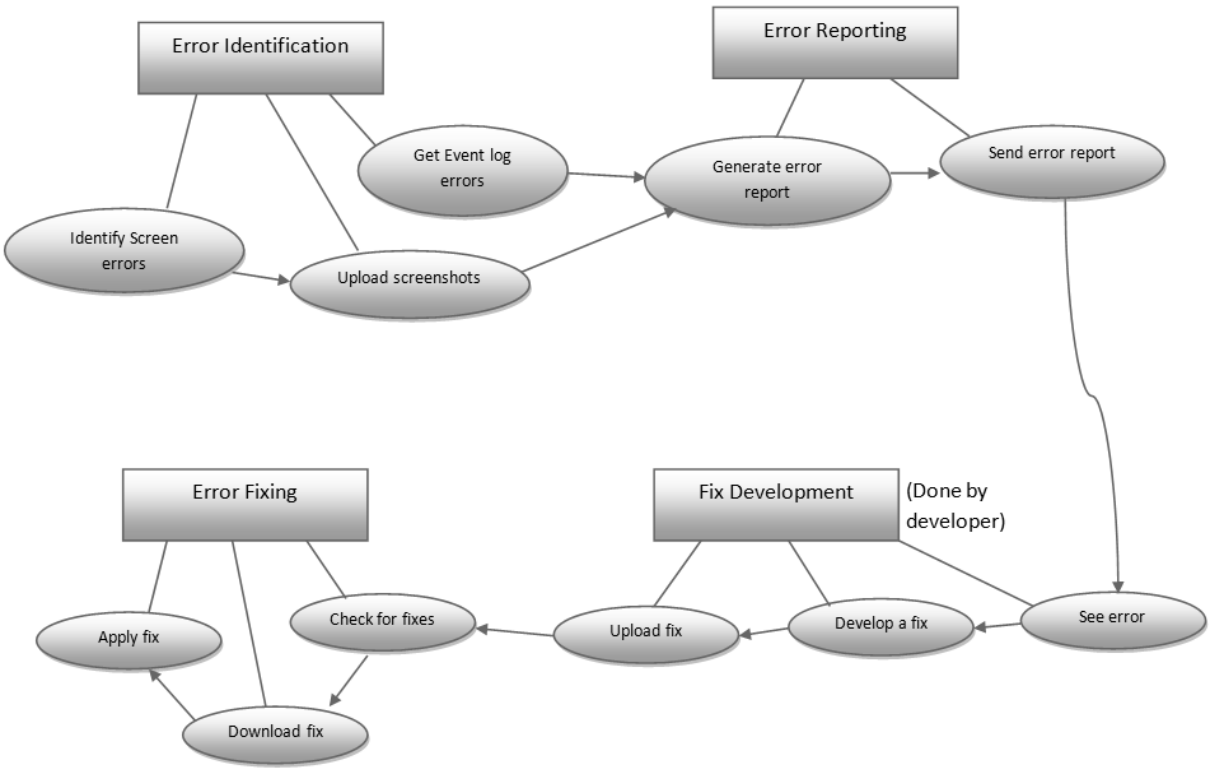


Figure 12: Role Model Diagram

There are roles that may have concurrent executing tasks that define the required roles behavior. In this system, the Identification of screen errors and the identification of Event log errors are done concurrently. This is represented in a concurrent model diagram shown below.

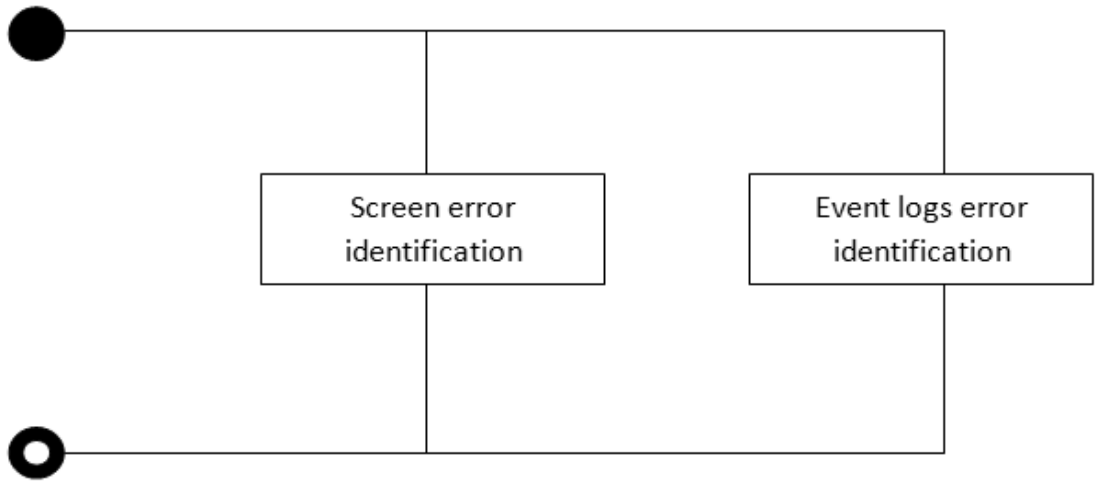


Figure 13: Concurrent model diagram

4.3 System Design

4.3.1 Overall architecture

Situated agents in the client computers will continuously monitor the system. The agents use system logs and screenshots to identify errors. On identifying an error, an agent creates an error report, and then uploads it using Dropbox. The error report will be downloaded on the developer's computer and logged in a bug tracking system. Once a fix is ready, it will be uploaded in Dropbox, where the agents can pick it and apply/install it on the client's machines.

The system will make use of limited internet connection at the hospital, whereby the error is captured and stored in the machine for submission when a connection to the internet is available.

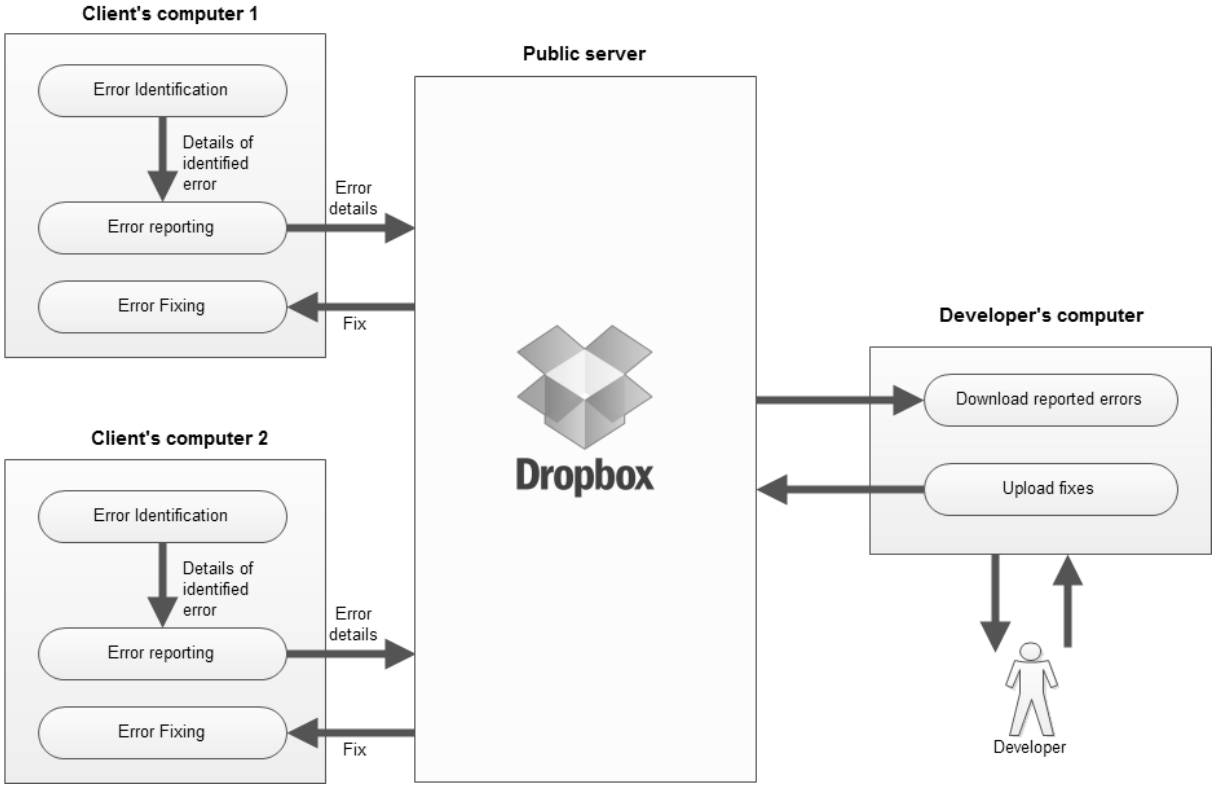


Figure 14: Overall architecture

The process of screen error identification involves the capturing of screen shot, and then text is extracted from the image using Microsoft's Office Document Imaging libraries. An algorithm is then used to match the text to that of already known errors. If it matches then,

that means an error has occurred, and therefore it will be reported. The following applications therefore will need to be put in place to ensure the successful running of the system:

1. Installing Office 2007, ensuring Microsoft Office Document Imaging is installed.
2. Installation of Dropbox and logging in using a valid Dropbox account

4.3.2 Flow design

4.3.2.1 Overall flow of the system

The identification of screen and event logs errors is done concurrently. The errors are then reported. Once a fix is developed, error fixing will be done.

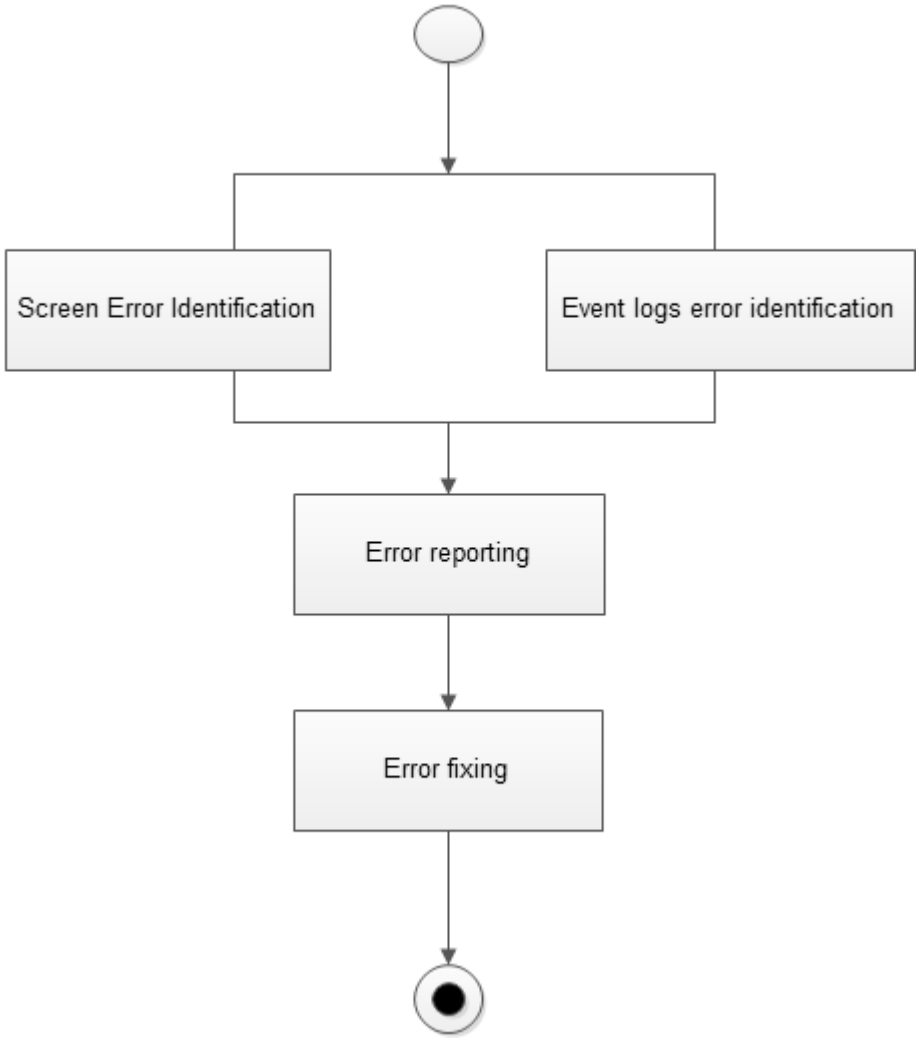


Figure 15: Overall flow of the system

4.3.2.2 Screen error identification flow diagram

The identification of screen errors entails ensuring that the currently active application is the target software, which is IQCare. The flow of the screen error identification is shown in the diagram below.

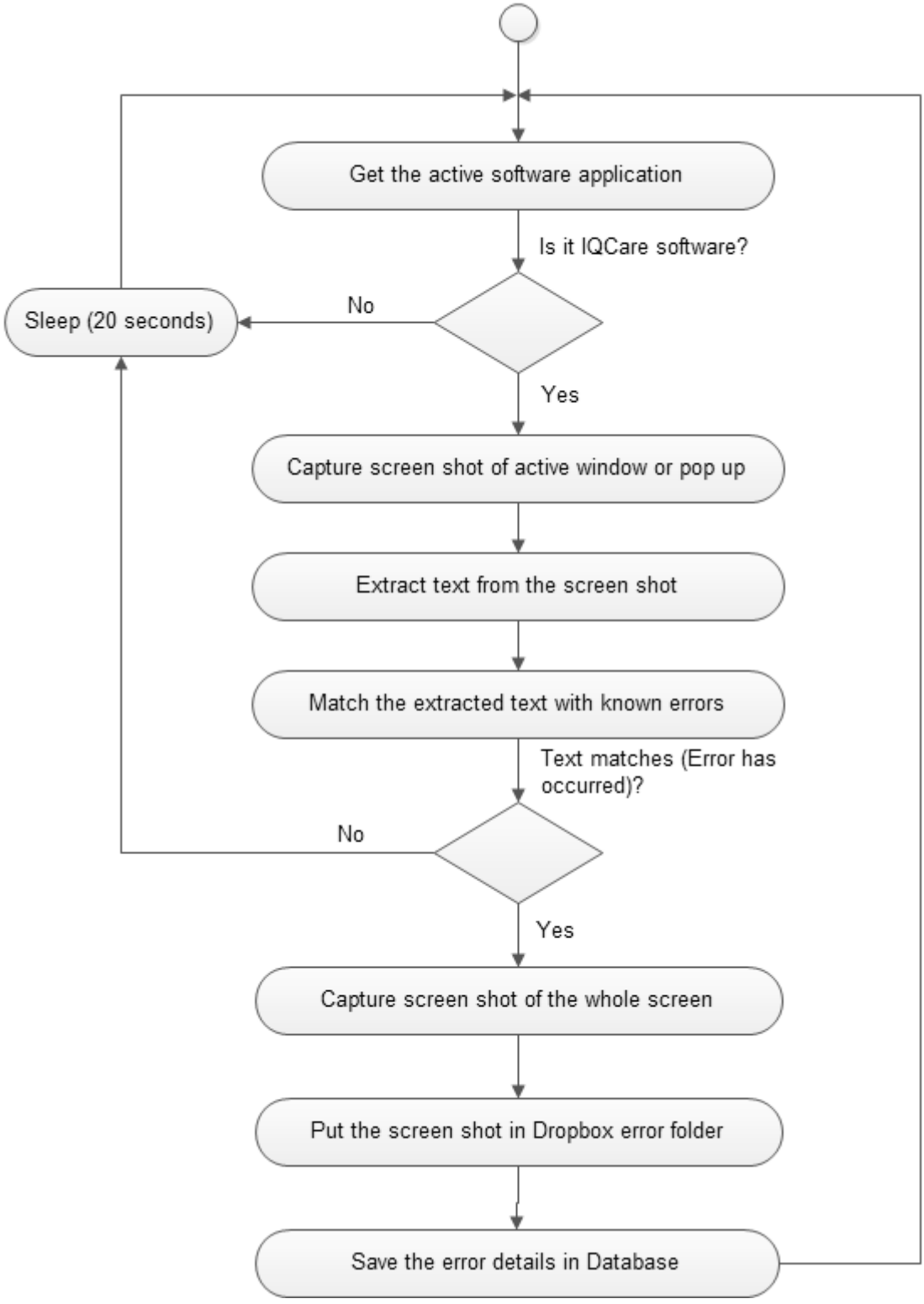


Figure 16: Screen error identification flow diagram

4.3.2.3 Event logs error identification flow diagram

Errors logged in the windows event logs are extracted in the process shown below:

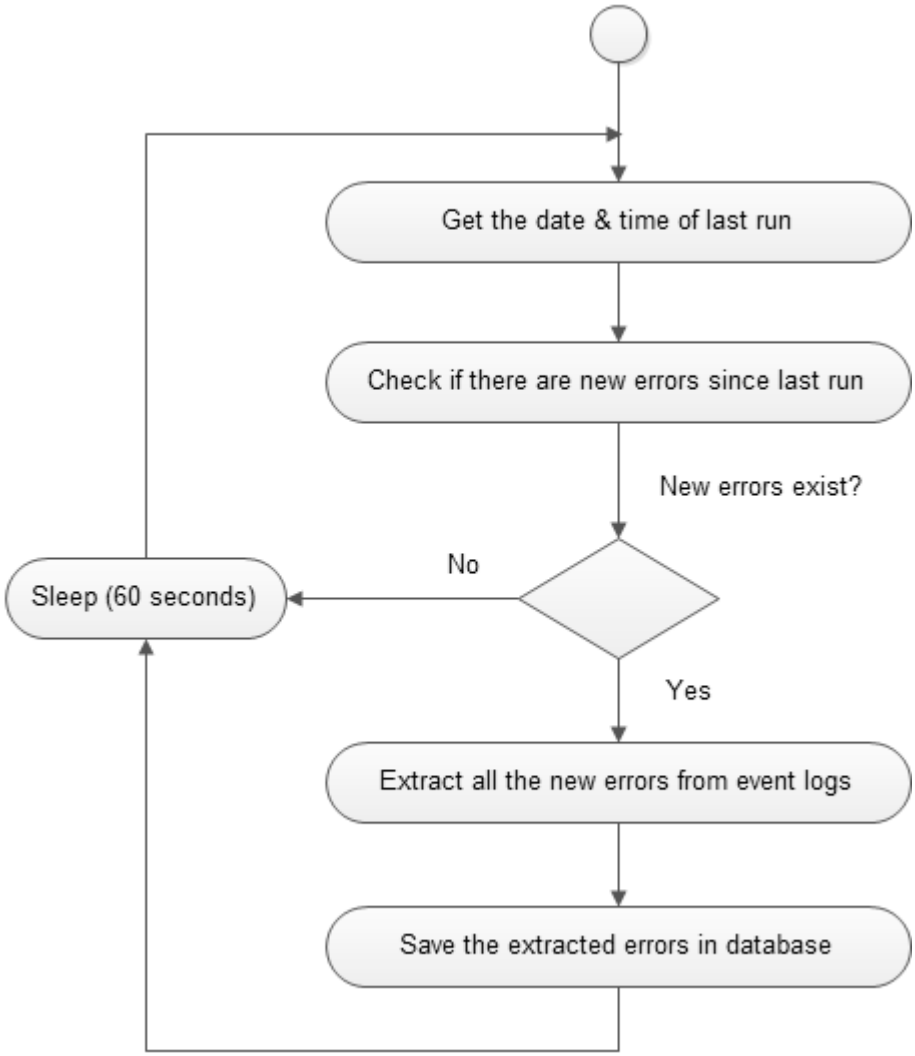


Figure 17: Event logs error identification flow diagram

4.3.2.4 Error reporting flow diagram

Once errors have been identified, they will be reported by placing them in a Dropbox folder. The synchronization mechanism of Dropbox will ensure that the errors will be accessible in the developer's computer.

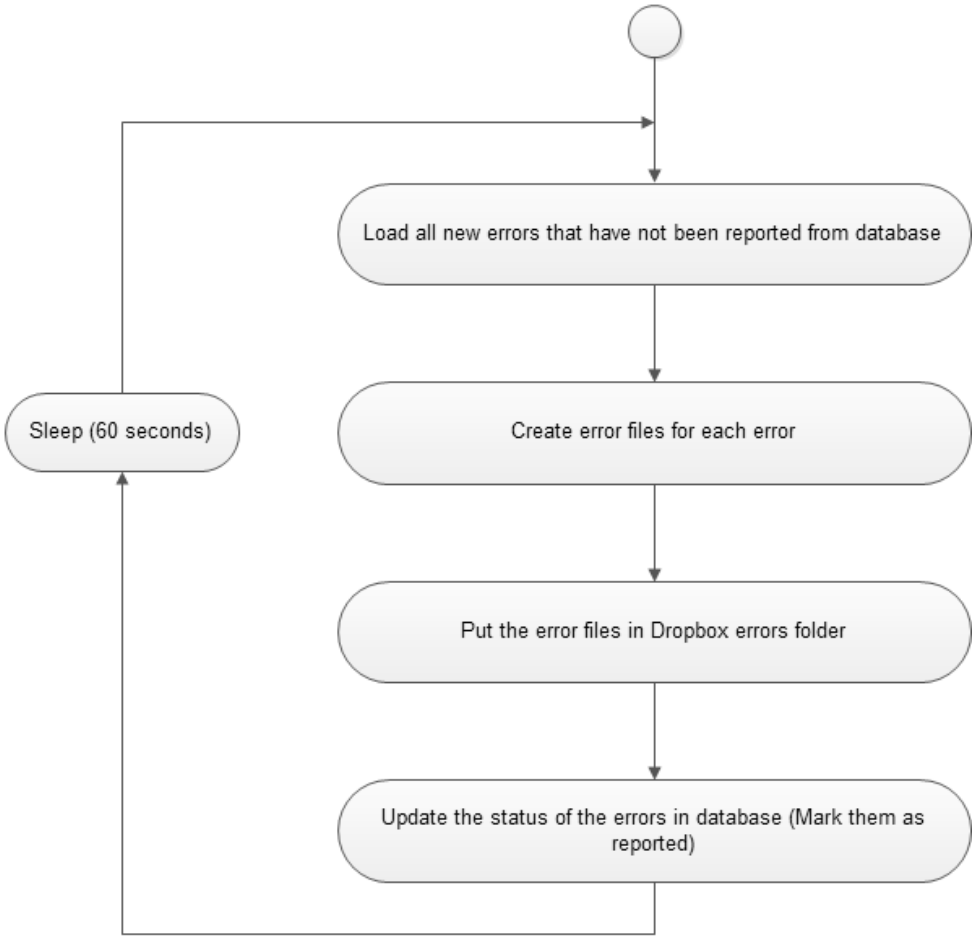


Figure 18: Error reporting flow diagram

4.3.2.5 Error fixing flow diagram

After the developer has developed and released a fix, the ErrorFixer agent will get and install the fix as described in the flow diagram below.

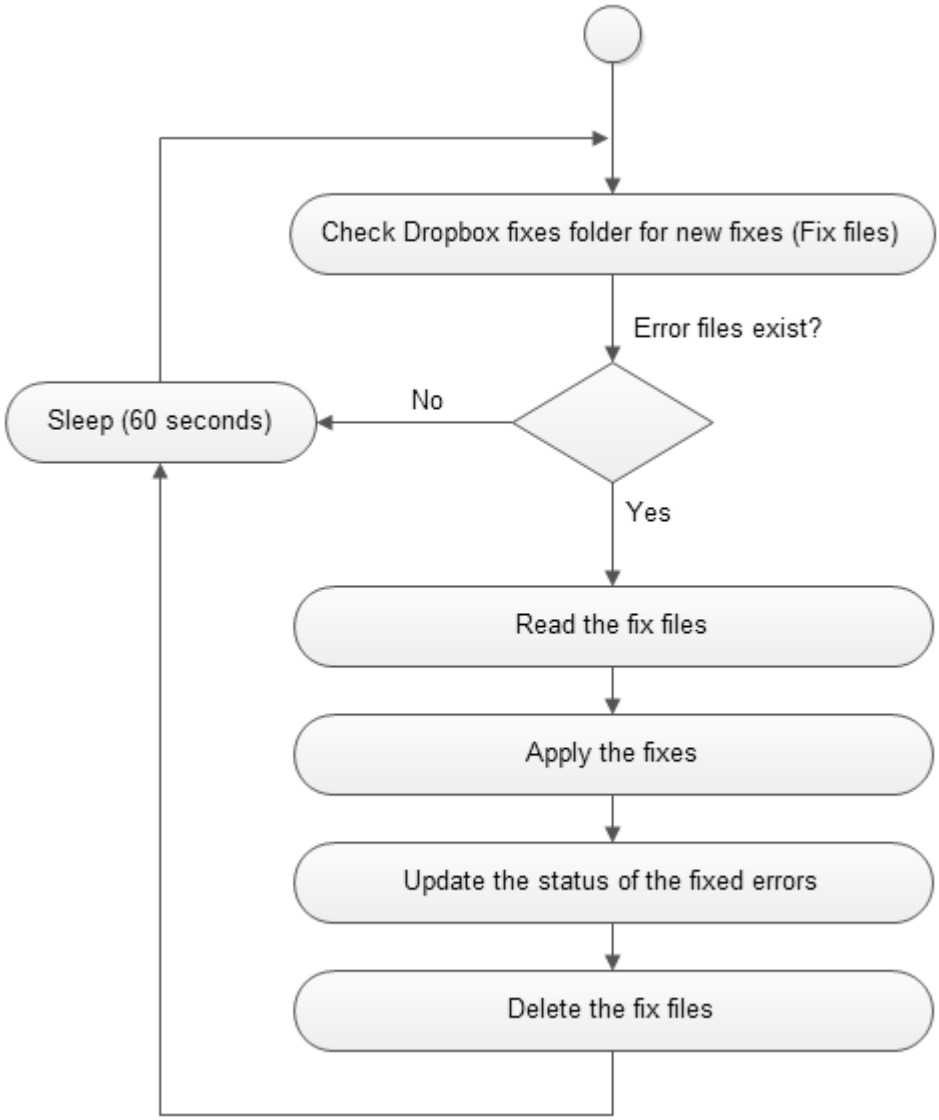


Figure 19: Error fixing flow diagram

4.3.3 Create agent classes

Agent classes are created from the roles defined in the Analysis phase. The end product of this phase is an Agent Class Diagram, which depicts the overall agent system organization consisting of agent classes and the conversations between them.

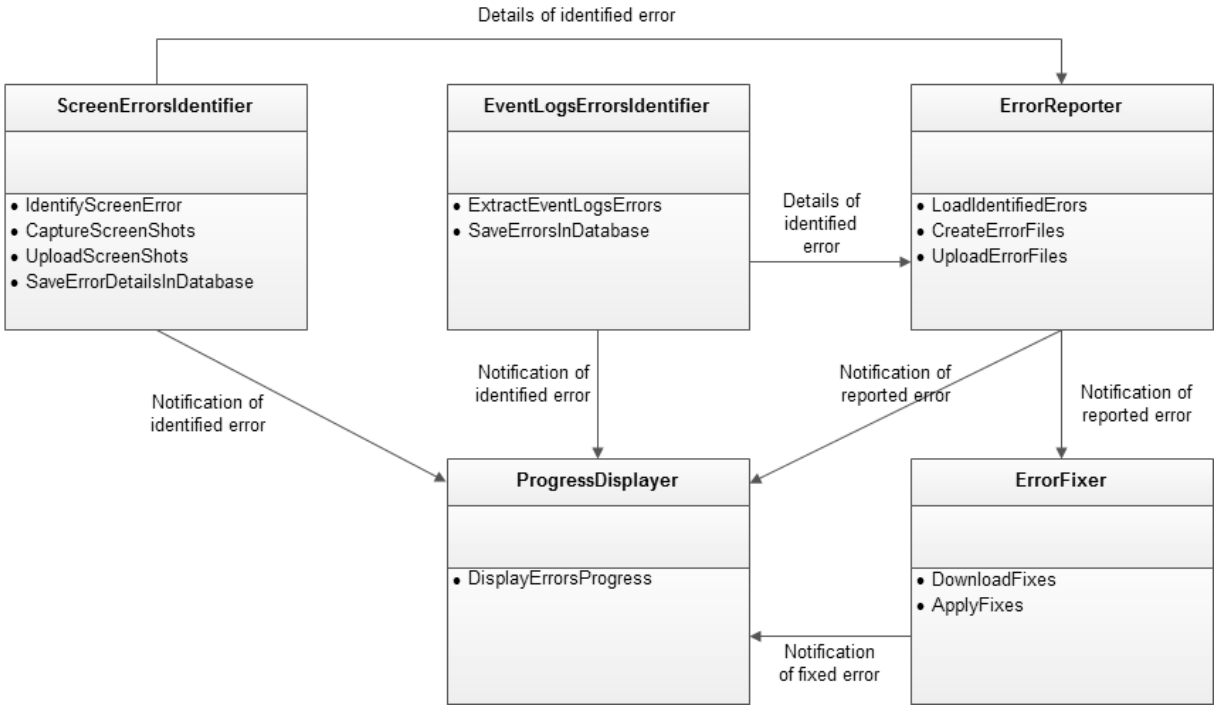


Figure 20: Agent classes diagram

4.3.4 Constructing conversations

Here, the conversations between agents are defined. The following conversations occur between agents:

a) Communication diagram for error identification

On identification of an error, the ErrorIdentifier agent will send a message to the ErrorReporter and ProgressDisplayer agents. The communication is done informally by updating the status of an error on the database.

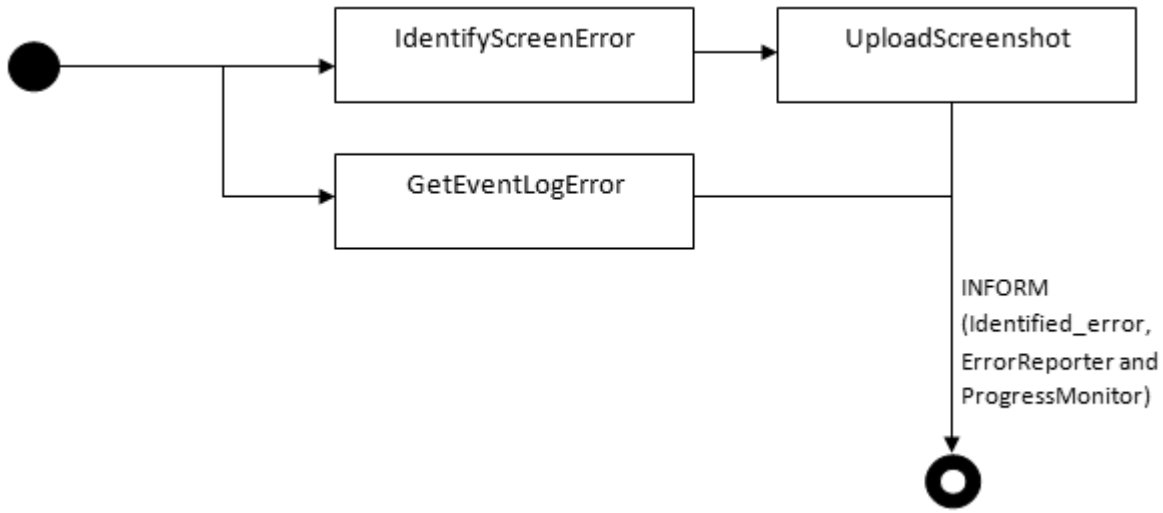


Figure 21: Communication diagram for error identification

b) Communication diagram for error reporting

The ErrorReporter agent will receive an information message from the ErrorIdentifier agent. It will then generate an error report and then send it. After sending the report, it will inform the ErrorFixer and ProgressDisplayer agents of the reported error.

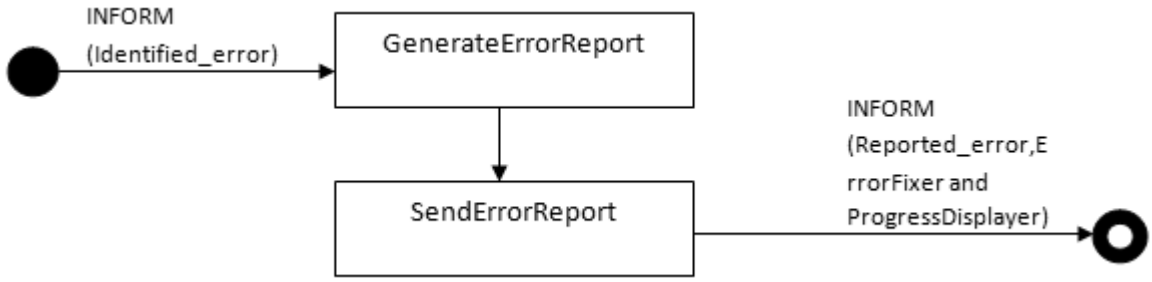


Figure 22: Communication diagram for error reporting

c) Communication diagram for error fixing

Upon receiving a message from the ErrorReporter agent, the ErrorFixer will check for new fixes, download and then apply/Install the fix. It will then send a message to the ProgressDisplayer agent, with the details of the fixed error.

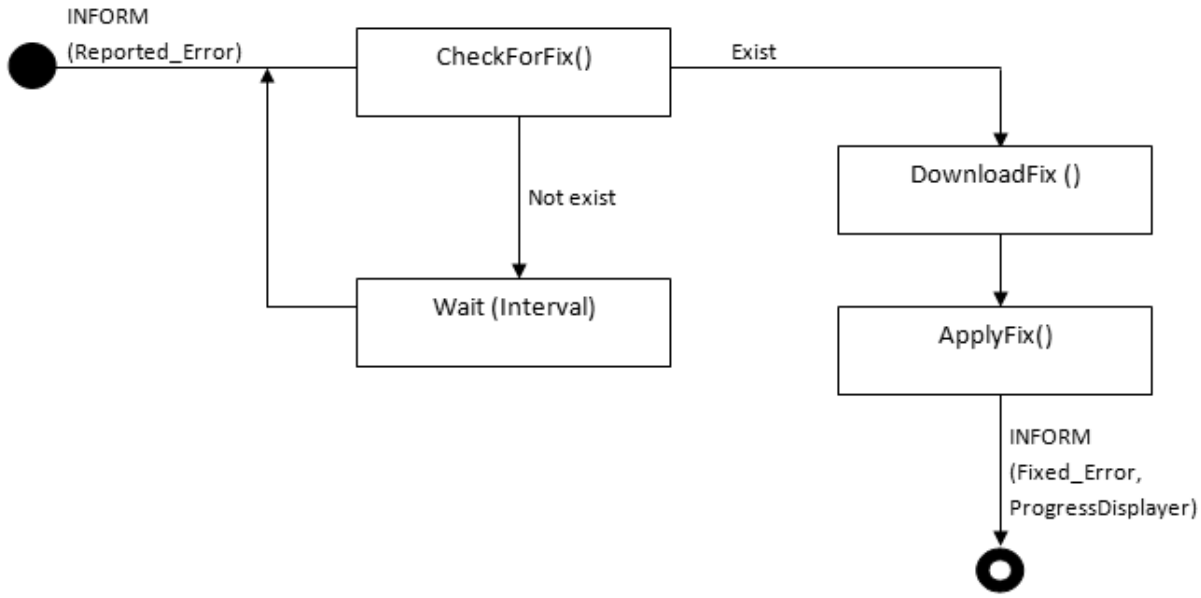


Figure 23: Communication diagram for error fixing

d) Communication diagram for Progress display

The agent will be receiving messages from the ErrorIdentifier, ErrorReporter and ErrorFixer agents. Upon receiving a message from the ErrorIdentifier, it adds it to the list of identified errors. It will use the messages from the Error Reporter and ErrorFixer to update the status of the error accordingly.

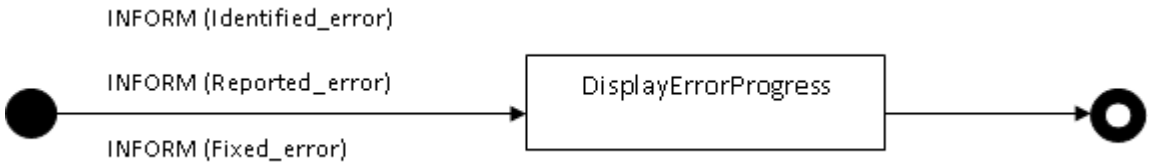


Figure 24: Communication diagram for progress display

4.3.5 Creating a sequence diagram

A Sequence Diagram depicts the sequence of events that are transmitted between the agent classes. The communications described above are summed up in a sequence diagram.

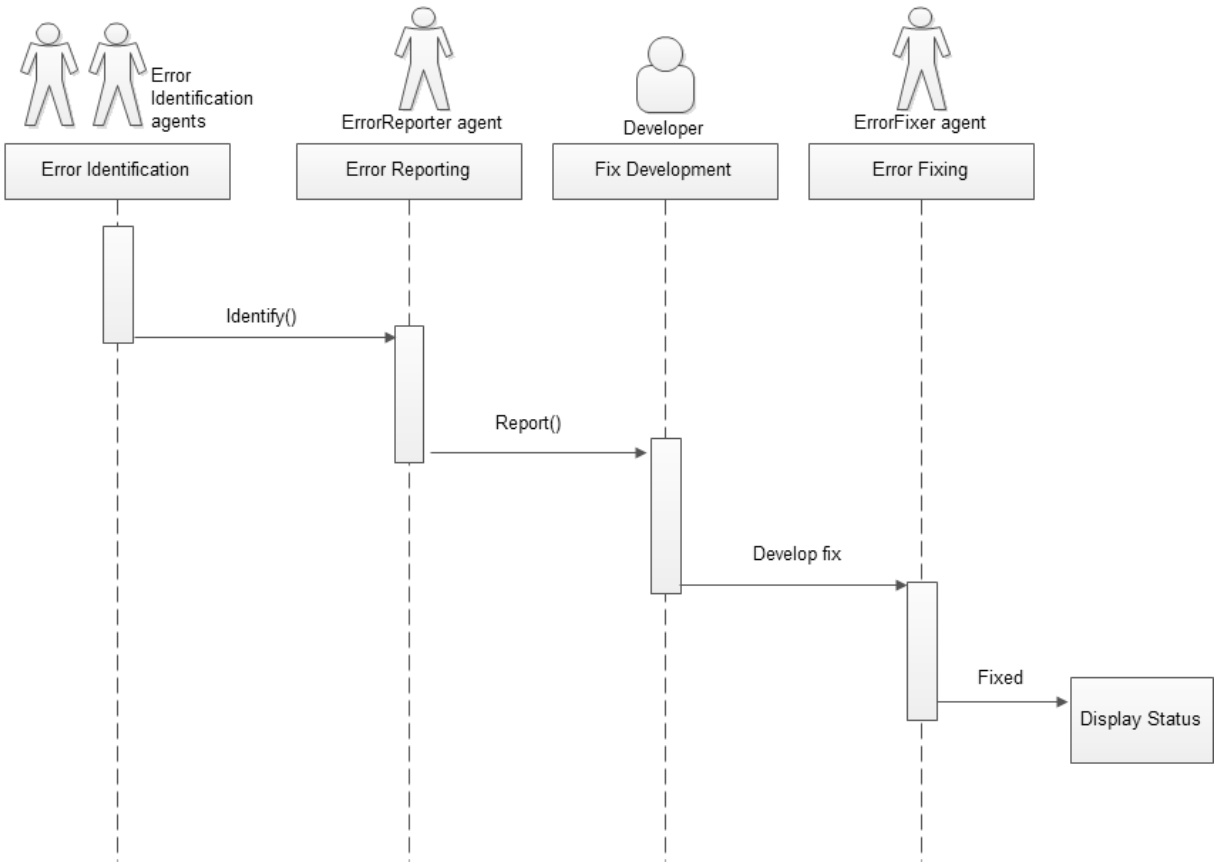


Figure 25: Sequence diagram

4.3.6 Assembling agents

The thick dotted lines represent outer agent connectors. They define connection with external resources such as other agents, sensors, databases and data stores.

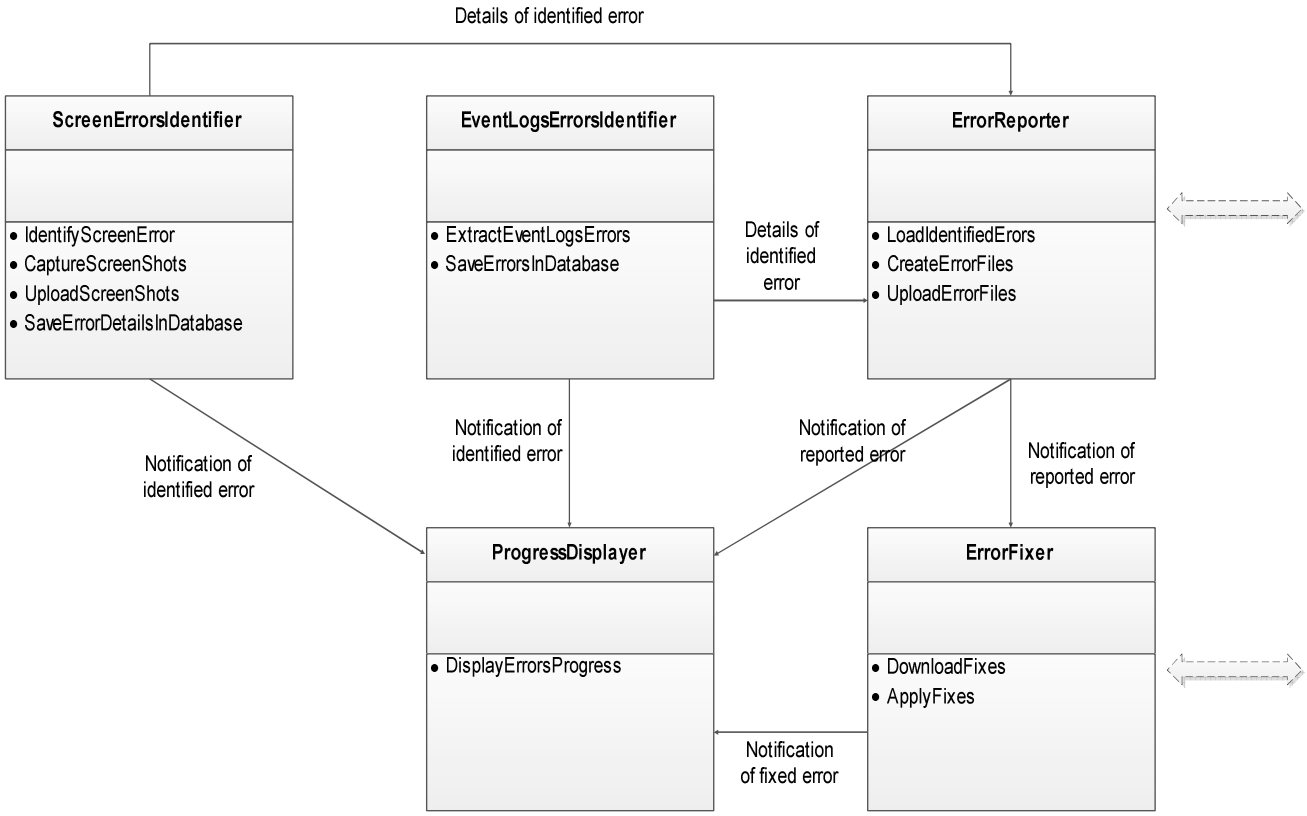


Figure 26: Agent architecture

4.3.7 Instantiating the agents

Here we take the agent classes and instantiate actual agents. The instantiation is represented in a deployment diagram shown below.

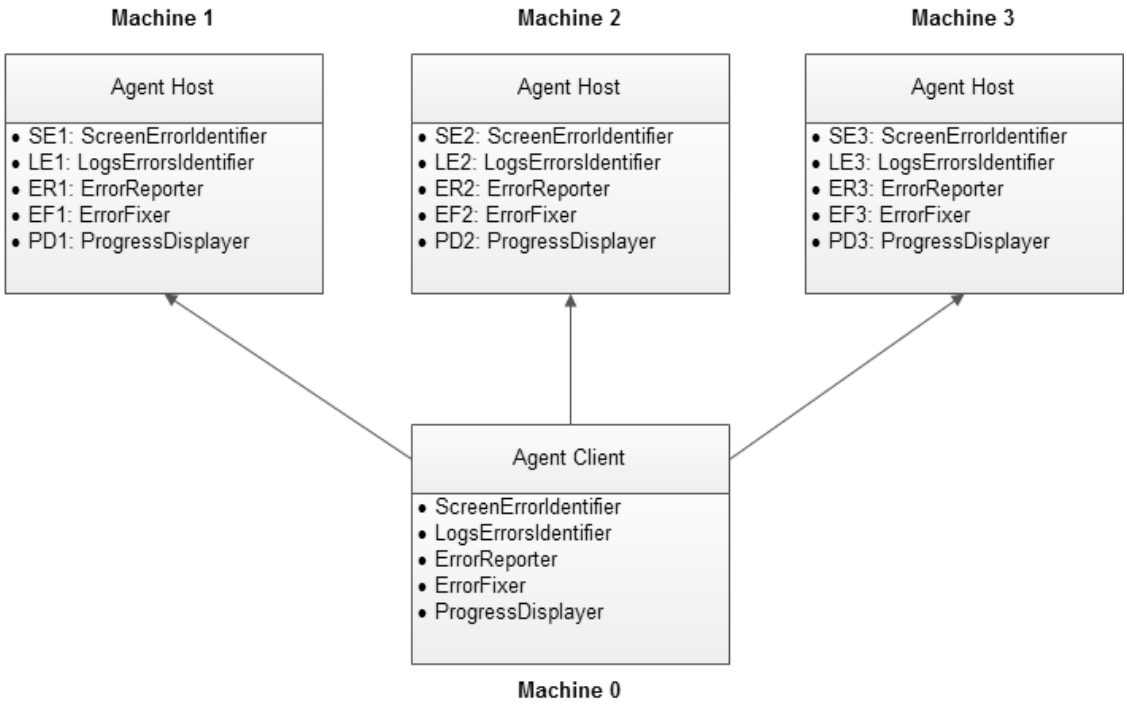


Figure 27: Deployment diagram

4.3.8 Database design

The database stores the details of the identified error, the status of whether it is reported or not and the status of whether it is fixed or not. Moreover, it will also store the details of the fix applied on the target software.

A reported error might have one or more fixes, and therefore the relationship between the identified error and the fix will be one-to-many relationship. Two tables will be created, for storing the errors and the fixes respectively, as shown in the following database diagram.

After determining the data to be stored, and applying the normalization rules, I have come up with two tables described below:

Errors table

Name	Data type	Length	Description
Id	Int	4	Primary key
ErrorName	Text	100	Name of the error
Category	Text	20	Category of the error, which is Event logs or ScreenErrors
ErrorText	Text	Max	The details of the caqptured error
AttachmentLink	Text	300	A web link of uploaded screen shots
MachineName	Text	20	Name of the computer where the error occurred
MsgSendToReporter	Boolean	1	A status to indicate whether the error has been passed to reporter module or not
Reported	Boolean	1	A status to indicate whether error has been reported or not
MsgSendToFixer	Boolean	1	A status to indicate whether the fixing component has been notified
DateOfError	DateTime	8	date and time when the error occurred

Fixes table

Column	Data type	Size	Description
id	Integer	4	Primary key
ErrorID	Integer	4	Foreign key links to Errors table
FixType	Text	20	Type of fix. This is either manual patch, database script or executable patch
FixDestination	Text	300	This is used for manual patches. It is the folder where the files are copied to
DateReleased	date time	8	Date when the fix was uploaded by developer

Applied	Boolean	1	A status to indicate whether the fix has been applied or not
DateApplied	Date time	8	Date and time when the fix was applied

The “ErrorID” column of the fixes table links to the “id” column of the errors table.

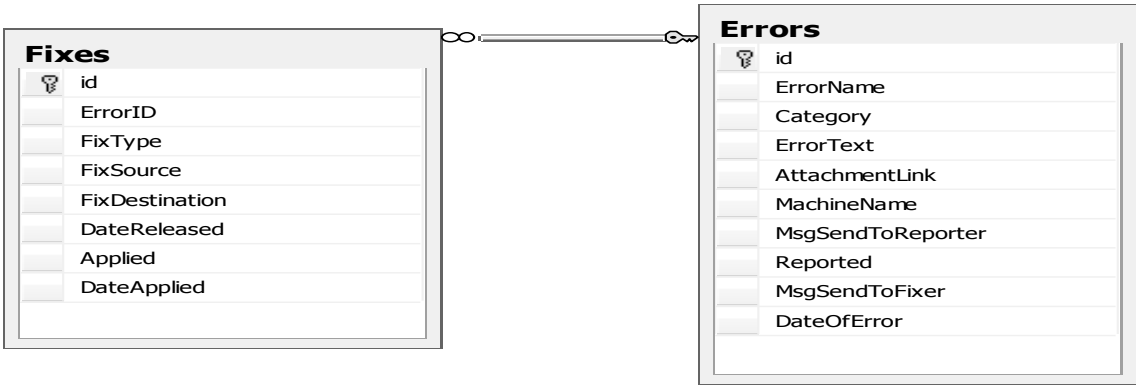


Figure 28: Database structure

4.3.9 Developer’s Module (Bug Tracking System)

This is an application that is running on the software developer’s computer. Any reported error will be displayed by this application. Once an error is reported, it will be downloaded and then displayed. The developer will then develop a fix, and upload it using this module.

The two core functionalities are:

1. Download all details of the reported errors
2. Enable the developer to upload fixes for the errors

4.3.9.1 Flow diagram for the downloading of reported errors

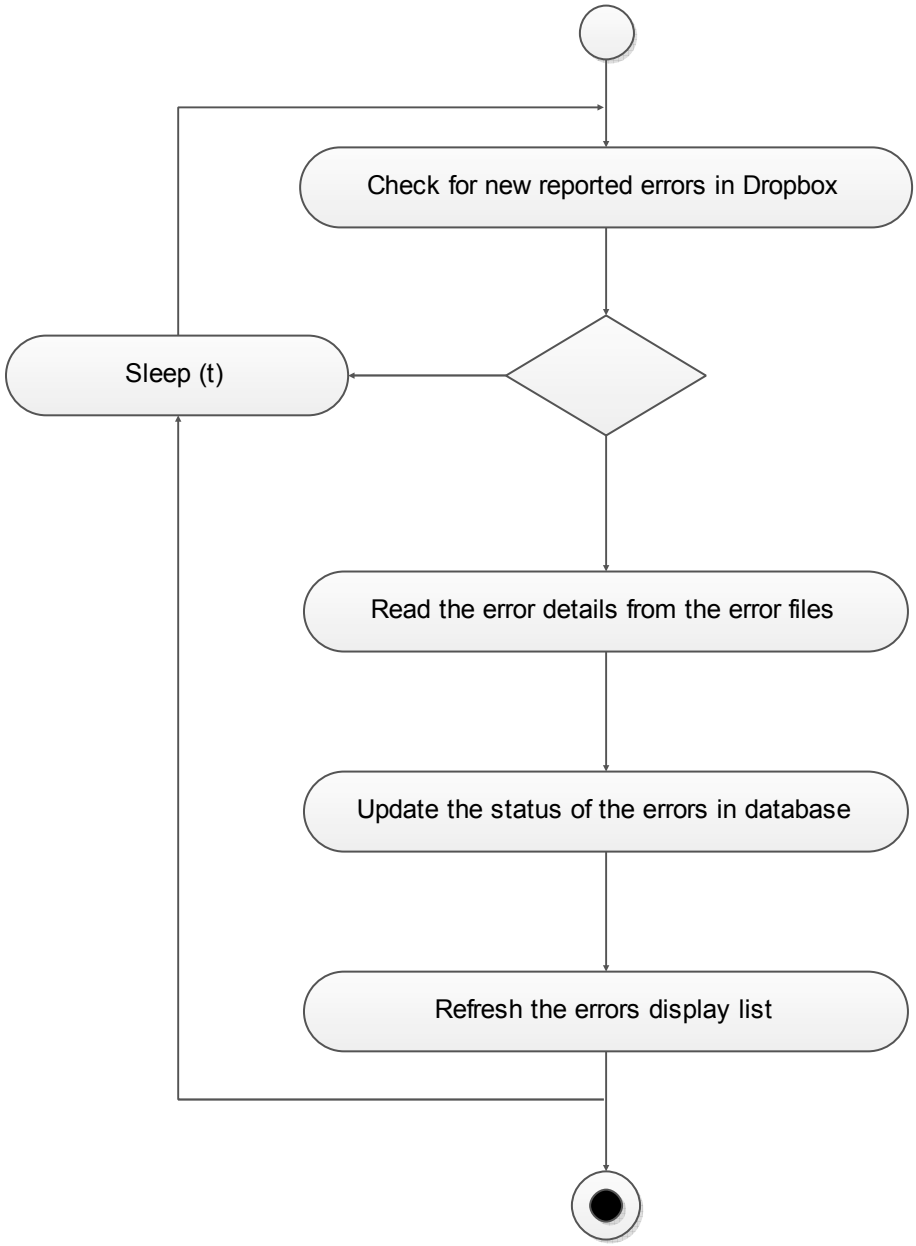


Figure 29: Download errors flow diagram

4.3.9.2 Flow diagram for the uploading of fixes for errors

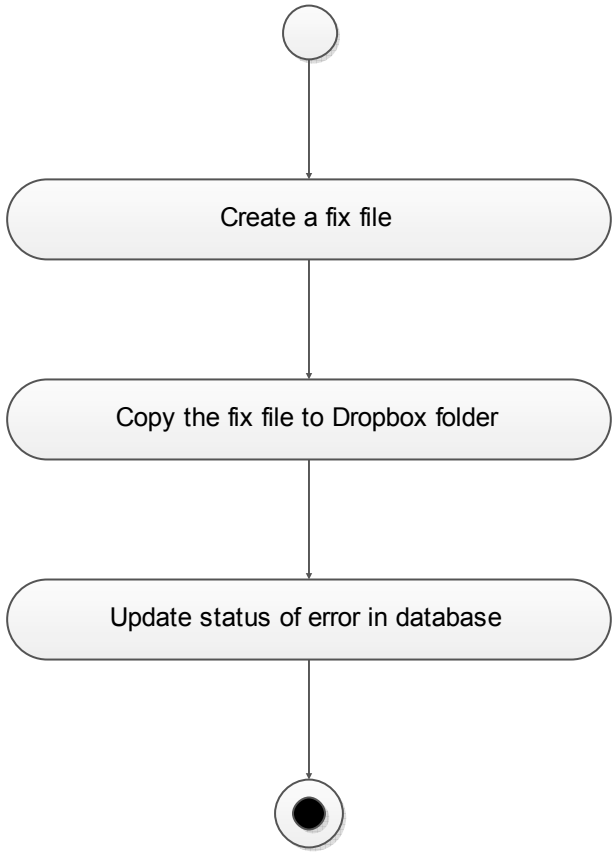


Figure 30: Upload fixes flow diagram

4.3.9.3 Database design

The data for this module is stored in a SQL Server database. The database has one table described below:

Column	Data type	Size	Description
Id	Integer	4	A unique identity
ErrorID	Integer	4	Id of the error in the client machine
ErrorName	Text	100	The name of the error
Category	Text	20	Category of the error: This is either screen errors or event log errors
ErrorText	Text	Max	The error text as captured
ReceivedFrom	Text	200	The Hospital/client where the error is from
FixUploaded	Boolean	1	The status of whether a fix has been uploaded or not
DateOfError	DateTime	8	The date and time when the error occurred

CHAPTER 5: SYSTEM IMPLEMENTATION AND TESTING

5.1 System development

The system is developed using C#.net. The agents are run in JADE platform, where some of the functionalities such as error identification and reporting are done in C#. The agents have the capability to move across machines. This is enabled by the use Dot Net framework, using a communication called remoting. Sample code have been included in the appendix section.

The development was broken down into modules. The following modules were identified:

1) Error identification

Error identification is done by reading errors in Event logs, and identifying screen errors. Identification of screen errors is done by capturing a screenshot, and then checking for errors details in the screenshot. The identification of errors is done using Optical character recognition tools. For this project Microsoft Office Document Imaging (MODI), to read all the text from the screenshot. We then check through the text for any of the error in the knowledge base.

2) Error Reporting

The errors are reported by use of dropbox software. The error reporting agent creates an error file, then copies it to a dropbox folder. This is a shared folder that will enable the application in the developer's end to access the shared error files.

3) Error fixing

This module checks for new fixes that have been uploaded by the developer. The process of uploading fixes is done using dropbox. The file synchronization feature of dropbox is used whereby; the developer places the fix in a folder in the server machine. The folder will be synchronized, thus having the same file downloaded to the client machine.

4) Progress Displaying

Developer's computer

This module is showing the progress of the reported errors.

I have also developed a module that runs on the developers end. This module is used to notify the developers of new bugs reported, and also to enable the developers to upload fixes.

5.2 Configuration

Once the system is developed, the following configurations will need to be done before running the system:

1. Putting all the known errors in database. These are the errors that have occurred in the past. The errors are obtained from the current bug tracking system
2. Setting/updating the name of the application to be tracked for errors
3. Installing Dropbox folder in the various client machines, and ensuring that Errors and fixes folder are created. The folders are created in one machine, and then shared with the users in the other machines, including the developer.

5.3 Testing and Experimentation

The system was set up in a testing environment, which was composed of two client computers representing the hospital computers and one computer representing the FG company server machine. The system was tested in different conditions to ensure that it was functioning well. The various tests are described below:

5.3.1 Testing system for screen and event logs errors identification and reporting

The five agents are started as shown in the screenshot below. The five agents are all mobile and can move across the network. An agent host needs to be installed on every computer. This is the container where the agents will reside when they move to the computer.

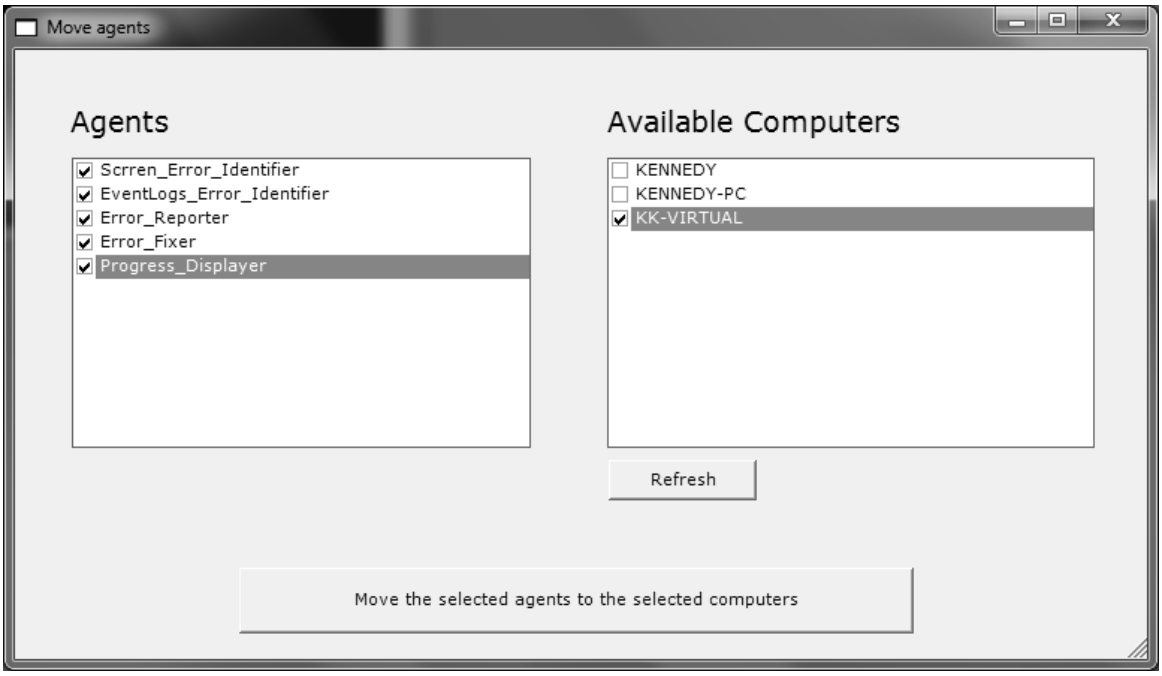


Figure 31: Main screen for starting mobile agents

On moving the agents, the status of the agents is displayed below, showing if the agents moved successfully to the target computer.

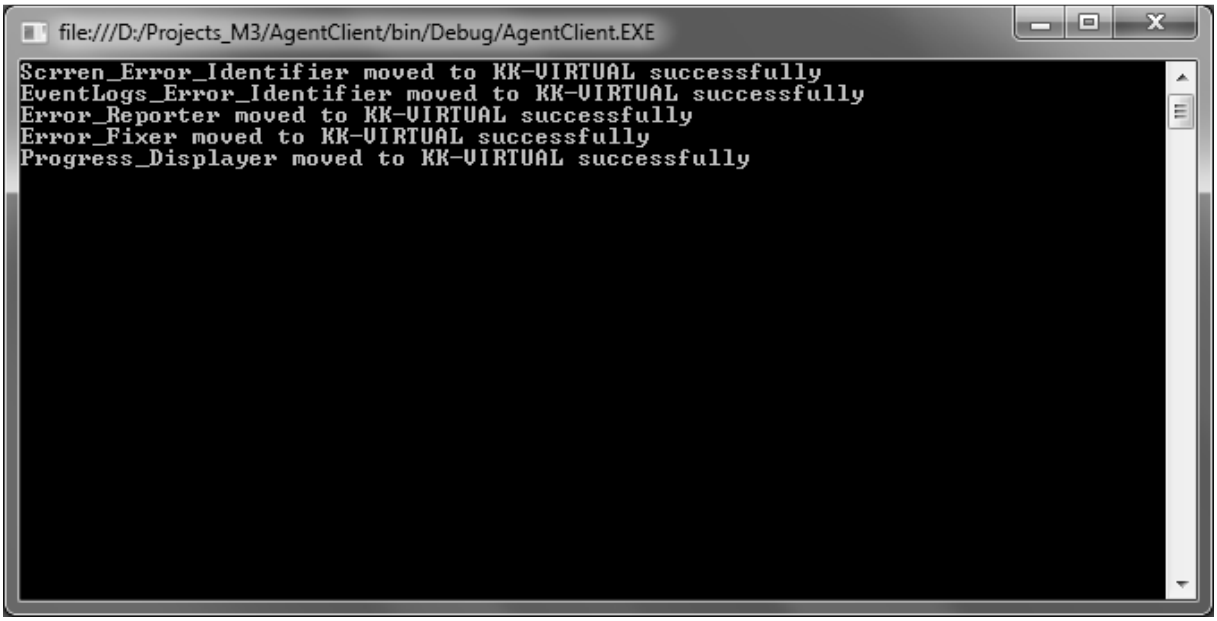


Figure 32: Screen shot showing agents moving to another computer

The agent host in the target computer will indicate when an agent has moved to the target computer as shown in the screenshot below. The agents will continue running until the agent host is closed.

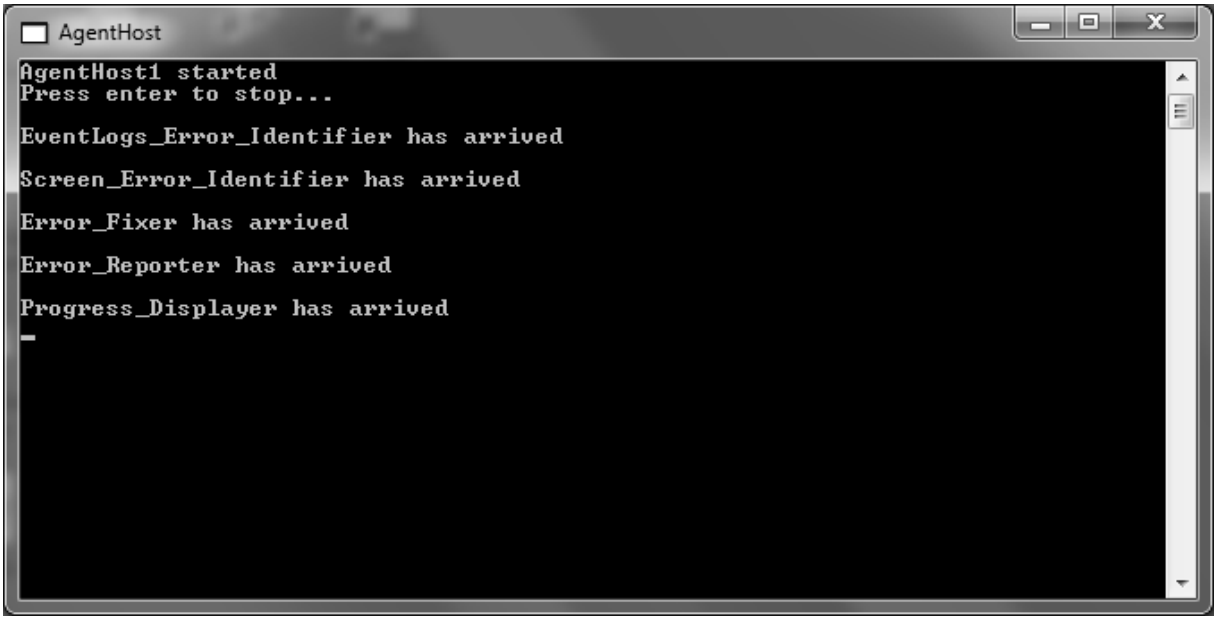


Figure 33: Screen shot showing agents moving into a computer

To test the error identification and reporting, errors were created by running IQCare and causing known errors like the one in the example below.

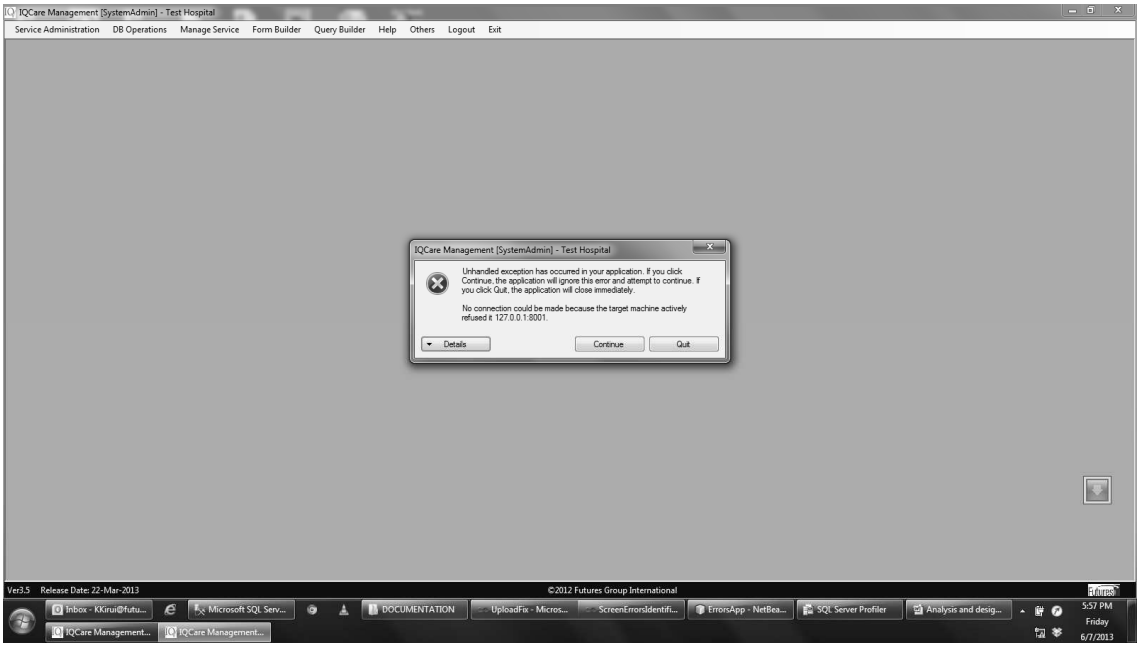


Figure 34: An example of an error occurring on IQCare System

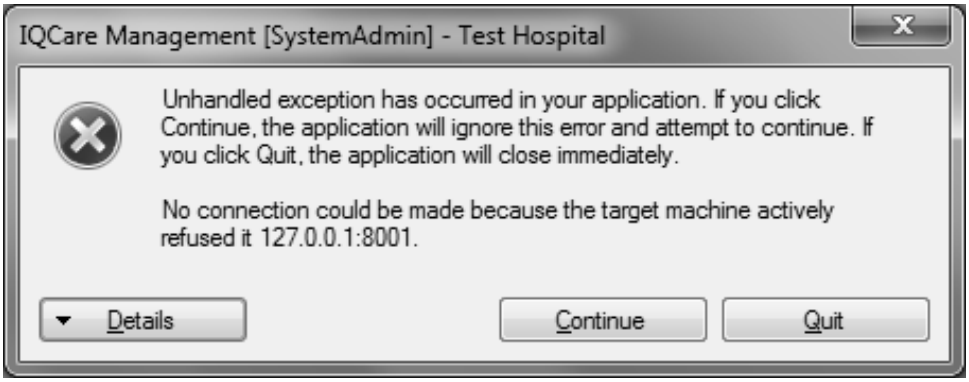
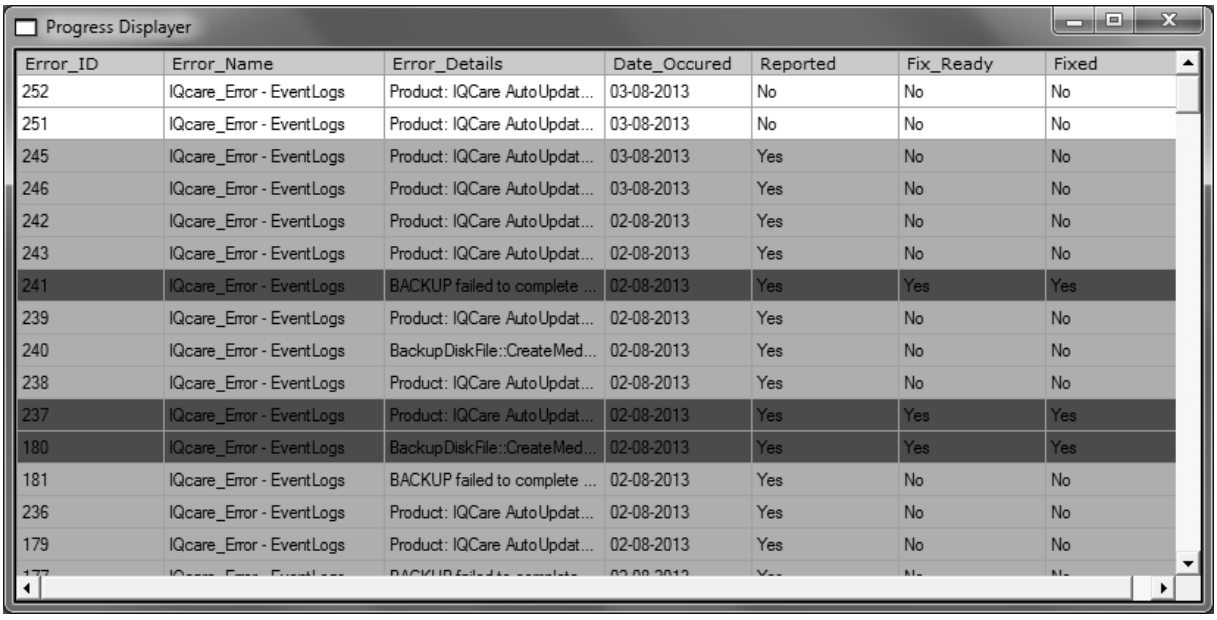


Figure 35: Details of the error occurring in IQCare system

A progress displayer displays the status of the errors. Different color codes are used to differentiate between identified, reported and fixed errors.



Error_ID	Error_Name	Error_Details	Date_Occured	Reported	Fix_Ready	Fixed
252	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	03-08-2013	No	No	No
251	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	03-08-2013	No	No	No
245	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	03-08-2013	Yes	No	No
246	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	03-08-2013	Yes	No	No
242	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	02-08-2013	Yes	No	No
243	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	02-08-2013	Yes	No	No
241	IQcare_Error - EventLogs	BACKUP failed to complete ...	02-08-2013	Yes	Yes	Yes
239	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	02-08-2013	Yes	No	No
240	IQcare_Error - EventLogs	BackupDiskFile::CreateMed...	02-08-2013	Yes	No	No
238	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	02-08-2013	Yes	No	No
237	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	02-08-2013	Yes	Yes	Yes
180	IQcare_Error - EventLogs	BackupDiskFile::CreateMed...	02-08-2013	Yes	Yes	Yes
181	IQcare_Error - EventLogs	BACKUP failed to complete ...	02-08-2013	Yes	No	No
236	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	02-08-2013	Yes	No	No
179	IQcare_Error - EventLogs	Product: IQCare AutoUpdat...	02-08-2013	Yes	No	No

Figure 36: Progress Displayer agent showing the status of various errors that have been identified

Developer's Module (Bug tracking system)

The following application is running on the developer's machine, enabling him/her to view any new reported errors. The seven errors shown in the previous diagram have all been

reported, and the developer is able to view them by clicking on 'Download new errors' button. On selecting an error, the error details are displayed on the right hand side.

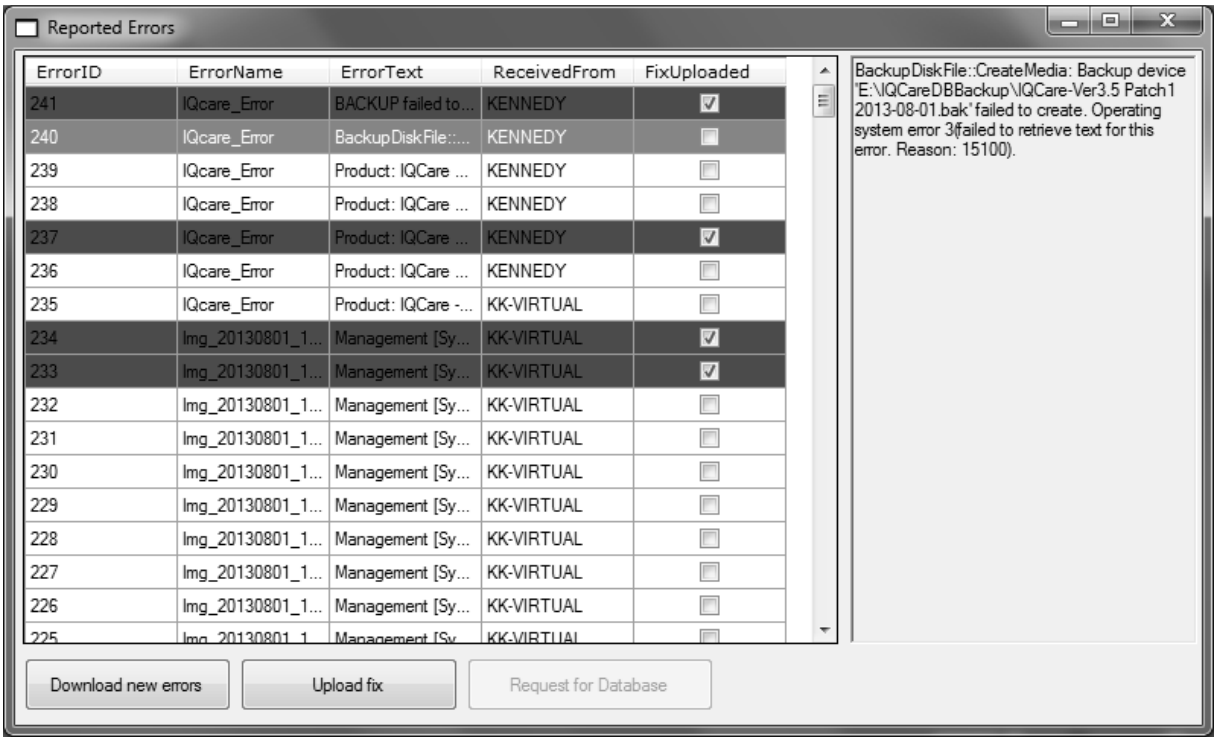


Figure 37: An application running at the developer's machine.

We developed a fix, and then uploaded it by clicking on the "Upload fix" button as shown in the screenshot below. Upon uploading a fix, the background colors changes to differentiate them from errors that do not have fixes.

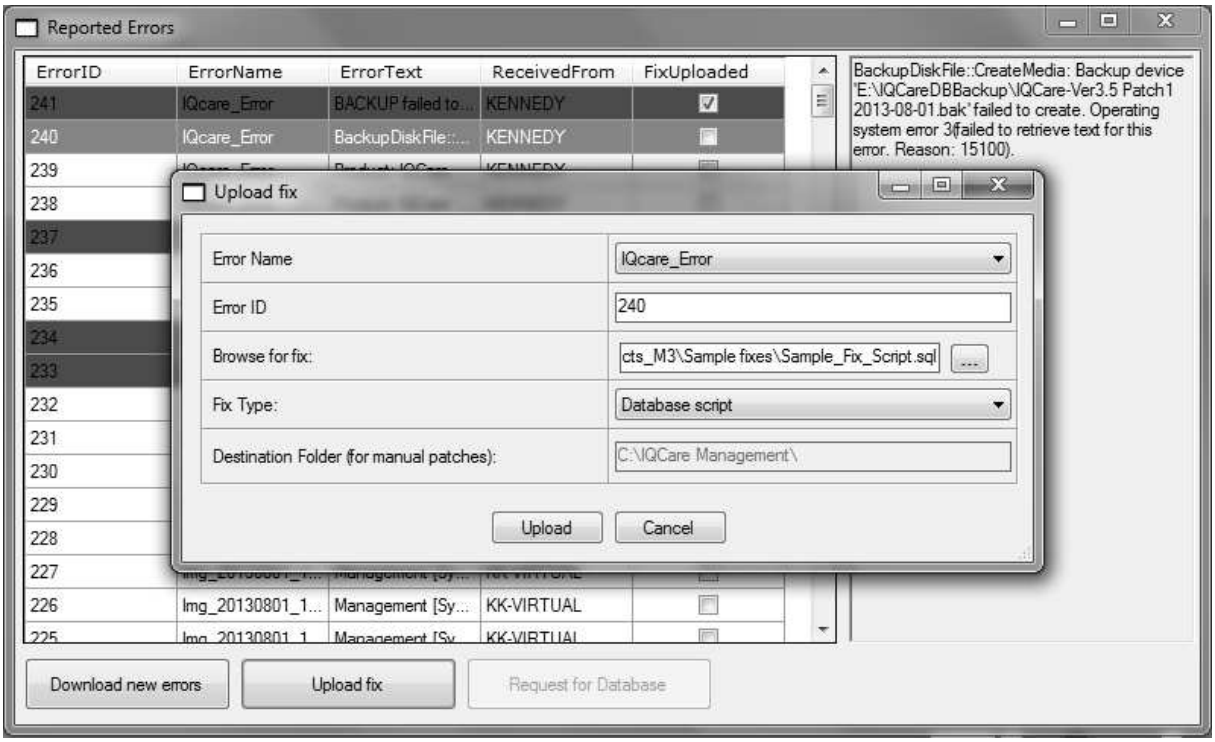


Figure 38: Process of uploading a fix by the developer

The system was tested with both screen errors and event log errors as outlined below:

(a) Screen errors

The system was tested by creating a number of known errors to ensure that the system is able to identify and report them. Ten known screen errors were caused to occur in two machines, and all of them were identified. Some of the known errors were caused to occur more than once therefore 52 errors were identified in total, and for every instance, they were reported successfully as shown in the table below. Ten fixes were uploaded, and all of them were applied successfully as shown in the table below.

Error Name	Category	Machine Name	Reported	Fix Uploaded	Fixed
Img_20130731_211539	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130731_211932	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130731_211956	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_121156	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_121205	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_125236	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_142727	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_144338	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE

Img_20130801_144339	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_144340	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_144341	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_144342	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_144343	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_144344	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_144345	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_144905	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_144906	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_144907	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_144912	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151040	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_151053	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_151054	ScreenError	KK-VIRTUAL	TRUE	TRUE	TRUE
Img_20130801_151102	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151105	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151114	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151435	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151440	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151453	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151454	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151454	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151500	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_151501	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_152027	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_152033	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_152033	ScreenError	KK-VIRTUAL	TRUE	FALSE	FALSE
Img_20130801_152051	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_152055	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_152055	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_152100	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_152108	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_152738	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_152800	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_153950	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_154002	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_154005	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_163348	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130801_163847	ScreenError	KENNEDY	TRUE	FALSE	FALSE

Img_20130803_090137	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130803_093635	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130803_093703	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130803_093727	ScreenError	KENNEDY	TRUE	FALSE	FALSE
Img_20130803_093738	ScreenError	KENNEDY	TRUE	FALSE	FALSE

Table 1: Table showing a list of screen errors

(b) Event logs errors

Twenty four (24) event log errors were identified as well as summarized in the table below:

Error Name	Category	Machine Name	Reported	Fix Uploaded	Fixed
IQcare_Error_106	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_11	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_118	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_128	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_134	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_139	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_142	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_170	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_176	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_180	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_19	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_21	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_235	EventLogs	KK-VIRTUAL	TRUE	FALSE	FALSE
IQcare_Error_240	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_241	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_252	EventLogs	KENNEDY	TRUE	FALSE	FALSE
IQcare_Error_27	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_31	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_43	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_49	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_64	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_7	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_83	EventLogs	KENNEDY	TRUE	TRUE	TRUE
IQcare_Error_88	EventLogs	KENNEDY	TRUE	FALSE	FALSE

Table 2: Table showing list of event logs errors

The table shows that all the errors were reported, and for all errors that a fix was provided, the fix was downloaded and applied.

5.3.2 Testing system for internet downtime

The other thing that we tested was error identification and reporting when the internet connection is down. This was done by creating some system errors. The Errors were identified, and once the internet connection was restored, all the errors were reported to the developer. The use of Dropbox ensured that once an internet connection was available, the files were synchronized appropriately.

5.3.3 Testing system with non errors

This involved testing system to ensure that it reports errors only. This involved creating messages on the system that do not exist in the knowledge base as well as running the system in a normal manner without any errors. In this case, the system did not report any error.

5.3.4 Testing system for correctness screen error text captured

This is a test to ensure that the captured screen text is correct. The tests show that the captured screen text was 60% correct. The algorithm used when matching the text ensured that even with 60% correctness, then error text could still be matched with the text in database. Below is an example of error text reported by the system:

Screen error text:

> *IQCare Management [SystemAdmin] - Test Hospital unhandled exception has occurred in your application. If you click Continue, the application will ignore this error and attempt to continue. If you click Quit, the application will close immediately. No connection could be made because the target machine actively refused it 127.0.0.1:8001*

Reported error text*:*

> *IQCare Management [SystemAdmini - Test Hospital - - Unhaned cep(ion has oocured your appl,cgion you clidc OContinue, the ,pIication wi qoe tWs error and tent to contn.ie. I you ckdc (kd. the application wi dose immediate. No connection coid be made because the tag machine adively refused I 1270.0.1 :8001.*

5.3.5 Testing for Efficiency and accuracy

This is a test to ensure that the identified error is the one reported to the developer as well as to test the length of time that the system takes to report the error. The system identified errors as soon as they occurred. All errors that were identified were reported. Likewise all fixes that were uploaded from the developer's end were downloaded and then applied/installed.

5.3.6 System testing for user acceptability

The prototype system was set up at the company for the users to experiment with it. We did a research find out the user reaction to the prototype, and well as to assess the expected impact of implementing the system in the origination. The results of the research are described in the chapter 6.

5.3.7 Testing summary

(a) The summary of identification and reporting of errors

Type of Error	Errors that occurred	Errors identified	Errors reported
Screen Errors	52	52	52
Event logs errors	24	24	24

Table 3: Summary of identification and reporting of errors

(b) The summary of error fixing

Type of Error	Fixes uploaded	Fixes applied
Screen Errors	10	10
Event logs errors	10	10

Table 4: Summary of error fixing

(c) The overall summary of the tests done

Test	Pass or fail	Comments
Event log errors identification	Pass	
Screen errors identification	Pass	The text matching algorithm was adjusted to match only 60% of error text

Screenshots uploading	Pass	
Error reporting	Pass	
Error fixing	Pass	
Testing system in inconsistent internet connection	Pass	
Testing system with non errors	Pass	
Testing for accuracy and efficiency	Pass	The process takes a maximum of 5 minutes
Testing for user acceptability	Pass	

Table 5: Testing summary

CHAPTER 6: EVALUATION AND RESULTS

6.1 Overview

This chapter presents the results of the data analysis. An online survey was done which involved the respondents answering eleven questions, of which ten were multi-choice question. The survey contained ten statements in which the user stated how much they agree with the statement in a scale of 0 to 4. '0' represented "disagree" whereas '4' represented 'totally agree'.

6.2 Results and discussion

We received responses from 18 participants, and the following is an average of the responses for the participants. The ten statements and their average score for the 18 respondents:

Statement	Average Score
I am enthusiastic about the new system	2.72
The new system will cut down on customer support cost	3.17
The new system will provide an efficient way to get the issues from clients	3.17
Developers will get insight of the errors which are severe and frequent	2.83
The new system will ensure faster response to system issues	3
Deployment of fixes across the hospitals will be done much more easily	3.11
I might lose my job as a result of the use of this system	2.17
This system is hard to use	1.94
For me, the system will frustrate my work	1.83
We will achieve less when using this new system that what we are achieving now	1.78

Table 6: Table showing the average score for participants in the various survey questions

The first six statements are focused on evaluating the positive impact of the system. This is represented graphically in the figure 32. The chart reveals that on average, the system score around 3 points out of 4, which is around 75%. This shows that the expected positive impact of the system is good.

The last four statements are focused on evaluating the negative impact of the system. This is represented graphically in figure 33. The average score for the negative impacts of the system is slightly less than 2, which is around 45%. This is still a high figure, which should not be ignored. If the organization is to implement the proposed system, then it will need to put measures in place to address the four issues represented in the four statements.

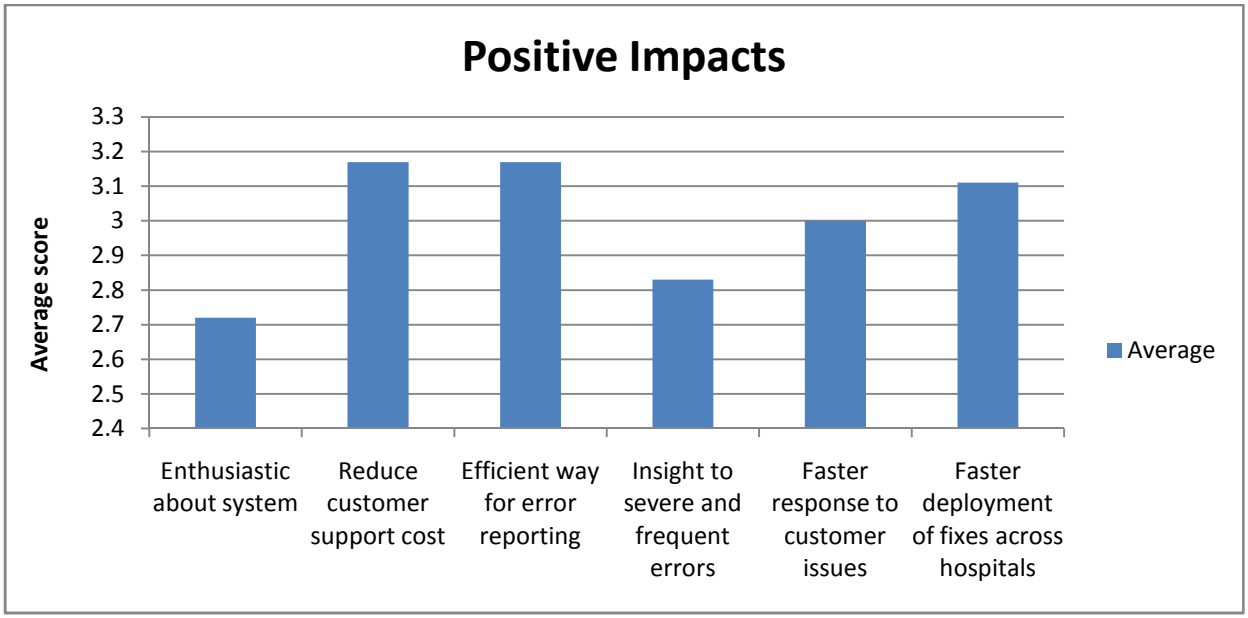


Figure 39: Figure showing the average score for the positive impact questions

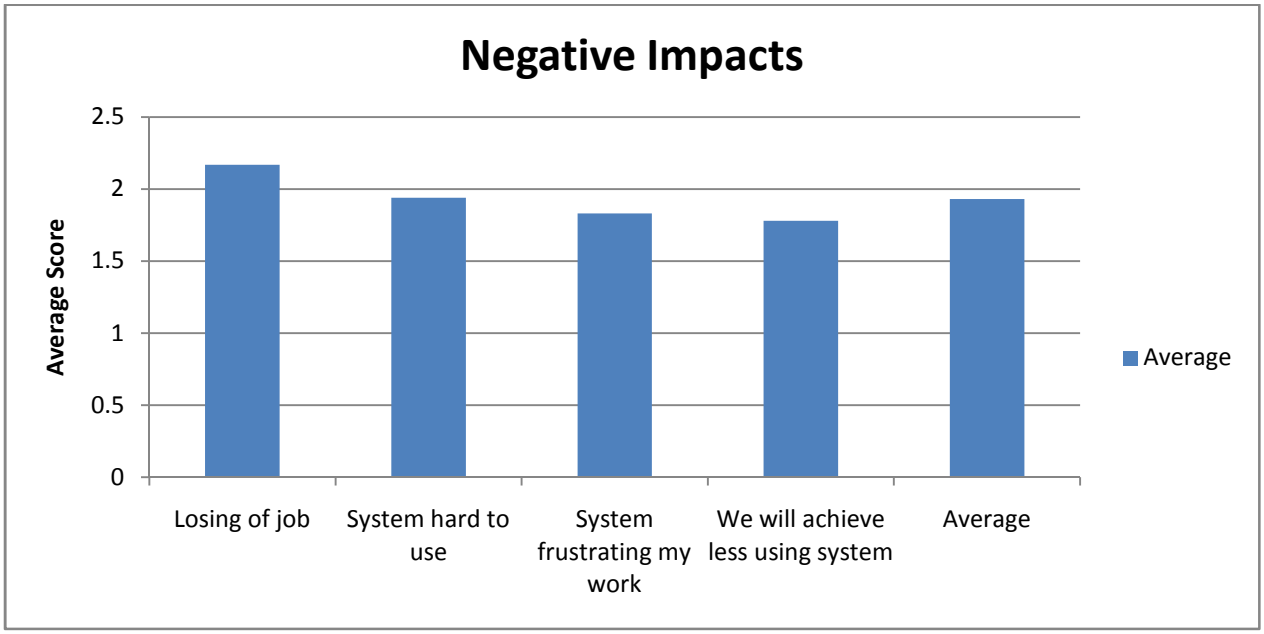


Figure 40: Average score for the negative impact questions

The following is a comparison of the negative and the positive impacts of proposed system. The positive impacts score higher than the negative impacts, and therefore this system is likely to receive a good response among a majority of the users.

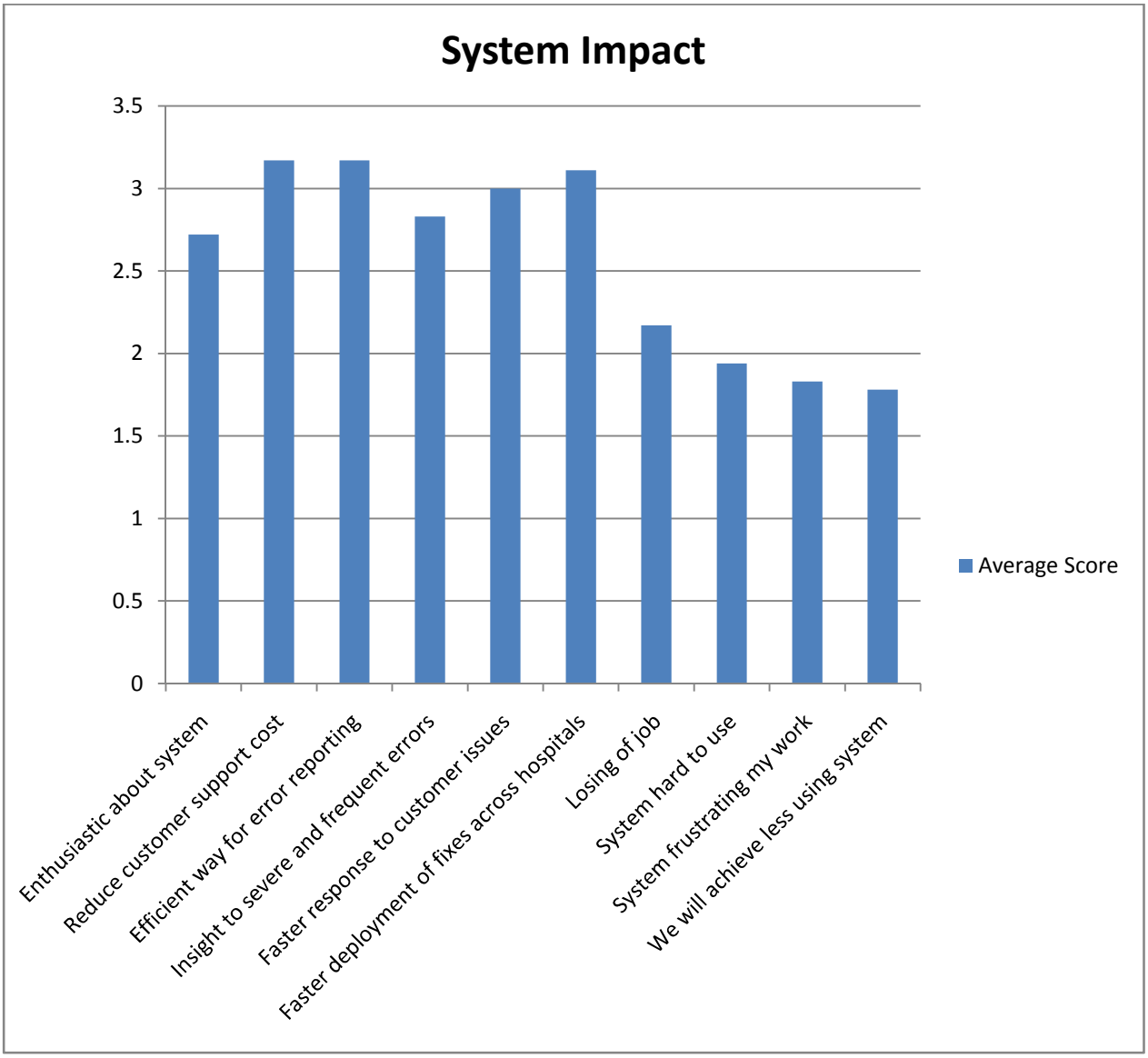


Figure 41: Comparison of the positive and the negative impacts

An analysis of the individual responses reveals that a number of users who have great expectations of the system, entered low values for the negative impacts of the system. Users who entered low values for the positives entered high values for the negatives. For instance respondent 10 (R10) recorded 4 for the negative impact, and 1.5 for the positive impact of the system. This could be an issue to do with the attitude towards the system.

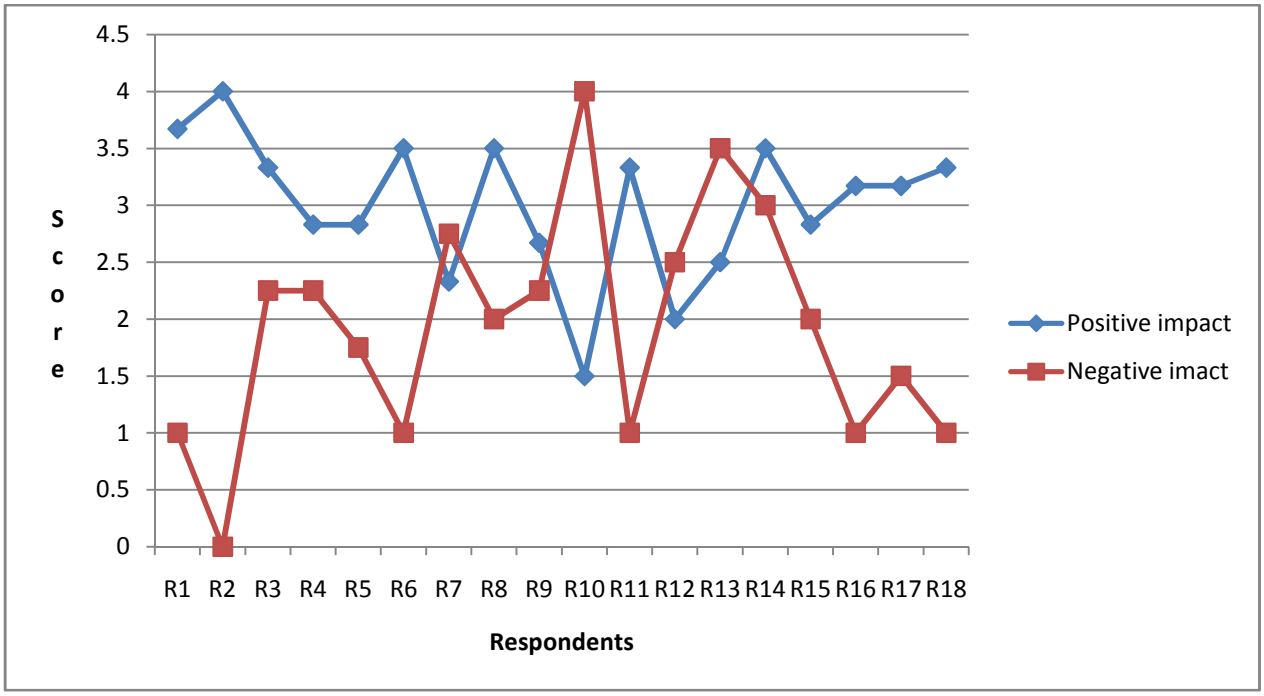


Figure 42: Positive versus negative impacts

The final question in the survey was open ended and asked "What are your thoughts about the proposed system?" Eight participants responded to this question, and here are their answers:

1. I am not sure if this system would be effective as some of our hospitals do not have constant internet connection, therefore it might result in even more delays.
2. This is a very important idea to implement. The ability to have the errors reported automatically will be a nice. Also the fact that screenshots will be captures, would ensure easier recreation of errors occuring at the hospitals
3. I like the proposed system. It would save the company quite some money
4. I am not sure this will work, this is because it will be hard to identify what is an error and what is not, but all the same I like the idea
5. The idea of automatic error reporting is a good one. If this can be implemented as proposed, then it help in knowing what system issues are being experienced in the hospitals, and being able to fix in time before they become serious issues
6. I like the system. This needs to be implemented.
7. I think this is a good system. My concern is on the resources it require. Will this affect the running of the main system?
8. I think the system is good. I am looking forward to using it.

Six out of the eight responses likes the proposed system, and think that it will work. This is about 75% of the respondents reacting positively towards the system.

6.3 Summary

The study reveals that reaction of the users towards the system is good, with about 75% of the participants responding positively. The expected positive impact of the system is high. There is also some expected negative impact of the system. Most of the expected negative impact is issue to to with attitude towards the system. Measures need to be put in place to ensure that the positive impact is maintained, whereas the negative impact of the system is minimised.

CHAPTER 7: CONCLUSION AND RECOMMENDATIONS

7.1 Conclusion

In this study, we have identified the technologies for error detection, reporting and fixing of errors. We have used these technologies in the development of a multi-agents based error identification, reporting and fixing system. We have tested the system under different scenarios and it works well. We have then done an evaluation of the expected impacts of the system to the organization. From the research, we established that the user response to the system is good. We have also established that both positive and negative impacts are expected if the system is implemented but the positive impacts outweigh the negative.

7.2 Limitations

The solution we have developed is limited to the identification of software errors that occur in two forms only:

a) Errors that are manifested on the screen, i.e. when an error occurs, evidences of the error are displayed on the screen.
b) Errors that are logged on the Application's Event logs.

Errors that are not are not manifested in the two ways above will not be captured by the developed system.

Secondly, the fixing of errors is limited to the following three ways only:

a) Manual patching: This involves the overwriting of the installation files in a program's installation directory
b) Running of a database update script: For database related issues, a script will be updated which will make the necessary correction on the database.
c) Installation of an update program: This is the running of an executable file which will re-install the program afresh or overwrite the files in the installation directory.

7.3 Further work

As explained above, I explored two ways of identifying errors. More work need to be done to come up with more ways of detecting and identifying software errors. Secondly, I used .NET framework as a platform for the agents. Compared to JADE, not much has been done on the

.NET framework; more work can be done on that area, to explore the potentiality in running agents in this platform.

REFERENCES

[1] IQcare, http://www.iqstrategy.net/frmiqcare.shtml, last accessed February 15, 2013

[2] Russell, S. J. and Norvig, P. (2003). Artificial Intelligence: a Modern Approach. Prentice Hall, 2nd edition

[3] Claudiu I., Andy S. and Laura S. (2011). Multi-Agent Approach for Data Analysis in a Knowledge-based System for Contact Centers, http://www.waset.org/journals/waset/v59/v59-216.pdf, last accessed February 15, 2013

[4] Michael N. and Larry M. (n.d.). Multiagent systems and societies of agents, http://www.dsi.fceia.unr.edu.ar/downloads/iia/biblio/cap2-agentes.pdf, last accessed February 15, 2013

[5] Katia P. (1998). Multiagent systems, http://www.cs.uga.edu/~maria/pads/papers/AIMag19-02-2-article.pdf, last accessed February 15, 2013

[6] Fabio L., Giovanni C. and Dominic G. (2007). Developing multiagent systems with JADE, http://www.ittelkom.ac.id/staf/kms/Advanced%20SW%20Eng%202011/ebook/developing%20multi%20agent%20systems%20with%20JADE.pdf, last accessed February 15, 2013

[7] Neely M. (2006). Write Mobile Agents In .NET To Roam And Interact On Your Network, http://msdn.microsoft.com/en-us/magazine/cc163649.aspx, last accessed February 15, 2013

[8] Wikipedia (n.d.). .NET Remoting, http://en.wikipedia.org/wiki/.NET_Remoting, last accessed February 15, 2013

[9] Hassan S. (2006). .NET remoting with an easy example, http://www.codeproject.com/Articles/14791/NET-Remoting-with-an-easy-example, last accessed February 15, 2013

[10] Jacques F., Olivier G. and Fabien M. (2004). From Agents to Organizations: an Organizational View of Multi-Agent Systems, http://www.lirmm.fr/~ferber/publications/papers/AOSE03_FerbGutMich.pdf, last accessed February 15, 2013

[11] Priyanka S., Hassan M., Mijal M. and Pranav P. (2004). Open Agent based system for strategic decisions using JADE Architecture, http://www.prjpublication.com/PrjAdmin/UploadFolder/Mijal_PPS_Pranav_hasan.pdf last accessed February 15, 2013

[12] Ion V. and Christian B. (1999). A study concerning the bug tracking applications, http://picoforge.int-evry.fr/cgi-bin/twiki/viewfile/Helios_wp3/Web/Task1Category?rev=1;filename=A_STUDY_CONCERNING_THE_BUG_TRACKING_APPLICATIONS.pdf, last accessed February 15, 2013

[13] Padmini C. (n.d.). Types of Software errors and bugs | Most Common Software bugs, http://www.softwaretestingtimes.com/2010/04/types-of-software-errors-and-bugs-most.html, last accessed February 15, 2013

[14] Investigation of software defect prediction, http://shodhganga.inflibnet.ac.in/bitstream/10603/4557/15/15_chapter%204.pdf, last accessed February 15, 2013

[15] Event logs, http://technet.microsoft.com/en-us/library/cc722404.aspx last accessed February 15, 2013

[16] About Optical Character Recognition in Google Drive, http://support.google.com/drive/bin/answer.py?hl=en&answer=176692, last accessed February 15, 2013

[17] Welker M. (2007). OCR with Microsoft Office, http://www.codeproject.com/Articles/10130/OCR-with-Microsoft-Office, last accessed February 15, 2013

[18] Language Technologies, Bangor University (2008). An overview of the Tesseract OCR (optical character recognition) engine, and its possible enhancement for use in Wales in a pre-competitive research stage, http://www.saltcymru.org/english/saltcymru_document5.pdf, last accessed February 15, 2013

[19] Google (n.d.). tesseract-ocr: An OCR Engine that was developed at HP Labs between 1985 and 1995... and now at Google, http://code.google.com/p/tesseract-ocr/, last accessed February 15, 2013

[20] OCR - Optical Character Recognition, OCR - Community Ubuntu documentation, https://help.ubuntu.com/community/OCR, last accessed February 15, 2013

[21] Tess4J, Tess4J - A Java JNA wrapper for Tesseract OCR API, http://tess4j.sourceforge.net/, last accessed February 15, 2013

[22] Yuan T., Chengnian S. and David L. (2012). Improved Duplicate Bug Report Identification, 16th European Conference on Software Maintenance and Reengineering, http://www.comp.nus.edu.sg/~suncn/papers/csmr12.pdf, last accessed February 15, 2013

[23] Jungwoo H., Christopher J., Jason V., Indrajit R., Hany E., Donald E., David L., and Emmett W. (2007). Improved error reporting for software that uses black-box components, Microsoft research, http://research.microsoft.com/apps/pubs/default.aspx?id=139859, last accessed February 15, 2013

[24] Hilton L. (2012). How to Write a Good Bug Report, Software and advice that's not of this verse, Retrieved Janauary 16, 2013, from http://noverse.com/blog/2012/06/how-to-write-a-good-bug-report/, last accessed February 15, 2013

[25] Nicolas B., Sascha J., Adrian S., Cathrin W., Rahul P. and Thomas Z. (n.d.). What Makes a Good Bug Report, Retrieved Janauary 16, 2013, from www.cs.vu.nl/~rpremraj/papers/08-fse.pdf

[26] Apple (n.d.). Bug reporting best practices, https://developer.apple.com/bugreporter/bugbestpractices.html, last accessed February 15, 2013

[27] Serdar Y. (2012). 10 file-sharing options: Dropbox, Google Drive and more, http://www.computerworld.com/s/article/9228869/10_file_sharing_options_Dropbox_Google_Drive_and_more, last accessed February 15, 2013

[28] Wikipedia (2012). Defect tracking, http://en.wikipedia.org/wiki/Defect_tracking, last accessed February 15, 2013

[29] Shreya J. (2011). Top 10 Open Source Bug Tracking Systems, http://www.toolsjournal.com/articles/item/184-top-10-open-source-bug-tracking-systems, last accessed February 15, 2013

[30] Detect tracking tools,http://softwarequalitysource.com/DefectTrackingTool.html, last accessed February 15, 2013

[31] Software Bug Tracking Tools,http://www.bug-tracking.info/software-bug-tracking-tools.php, last accessed February 15, 2013

[32] Scott A. Deloach, Mark F. Wood and H. Sparkman, Multi-agent systems engineering, from http://people.cis.ksu.edu/~sdeloach/publications/Journal/MaSE%20-%20IJSEKE.pdf, last accessed February 15, 2013

[33] Roderic, Incomplete error messages #2, http://drupal.org/node/1590212, last accessed February 15, 2013

[34] Error reporting and disclosure, R. Wolf, G. Hughes. http://www.ncbi.nlm.nih.gov/books/NBK2652/, last accessed February 15, 2013

[35] David U.(2008). Medication Error Reporting Systems: Problems and Solutions, http://www.ismp-canada.org/download/Medication%20Error%20Reporting%20Systems%20-%20Problems%20and%20Solutions.pdf, last accessed February 15, 2013

[36] W. Michael and P. Lin, (2004). "The prometheus methodology", http://www.cs.rmit.edu.au/agents/www/papers/mseas04-wp.pdf, last accessed Feb 25th, 2013

[37] G. Caire, F. leal, J. Rodriques, and P.Inovacao (2007).Message: Methodology for Engineering Systems of Software agents, http://archive.eurescom.eu/~pub-deliverables/P900-series/P907/TI2/p907ti2.pdf, last accessed February 15, 2013

[38] Scott A. DeLoach, (n.d.).Multiagent Systems Engineering: A Methodology And Language for Designing Agent Systems,http://macr.cis.ksu.edu/mase, last accessed February 15, 2013

[39] Crash reporter, http://en.wikipedia.org/wiki/Crash_reporter, last accessed Feb 25th, 2013

[40] S. Vafadar, A.Barfouroush, M. Reza and A. Shirazi, (2009). Bridging the Gaps in the MaSE Methodology, http://ceit.aut.ac.ir/~vafadar/Reference%20Files/Vafadar_ATS2003.pdf, last accessed February 15, 2013

[41] Automated Error Reporting: The Gateway to Better Quality, L. Lotfi. http://www.infoq.com/articles/Error-Reporting, last accessed February 15, 2013

[42] Wikipedia (n.d.), Dropbox (Service), http://en.wikipedia.org/wiki/Dropbox_(service), Last accessed February 15, 2012

APPENDIX

Appendix 1: Online survey form

ERROR REPORTING AND FIXING SYSTEM

Online Survey

Thank you for checking out the error reporting prototype system. Please take some time to give us your views on the system by answering the following questions. Note that this survey is anonymous so try to be as honest as possible.

The first ten questions are multi-choice requiring you to tell us how you agree with the statements in a scale of 0 to 4. "0" represents "STRONGLY DISAGREE", “1” represents “DISAGREE”, “2” represents “NEITHER AGREE NOR DISAGREE”, “3” represents “AGREE” and "4" represents "TOTALLY AGREE".

The last question is an open question where we are asking you to give us your thoughts about the system.

1. I am enthusiastic about the new system

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☉ 4

2. The new system will cut down on customer support cost

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

3. The new system will provide an efficient way to get the issues from clients

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

4. Developers will get insight of the errors which are severe and frequent

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

5. The new system will ensure faster response to system issues

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

6. Deployment of fixes across the hospitals will be done much more easily

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

7. I might lose my job as a result of the use of this system

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

8. This system is hard to use

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

9. For me, the system will frustrate my work

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

10. We will achieve less when using this new system that what we are achieving now

☐ 0 ☐ 1 ☐ 2 ☐ 3 ☐ 4

11. What are your thoughts about the prototype system?

Appendix 2: Data collected from the survey

	POSITIVE IMPACTS							**NEGATIVE IMPACTS**			
RESPONDENTS	Enthusiastic about system	Reduce customer support cost	Efficient way for error reporting	Insight to severe and frequent errors	Faster response to customer issues	Faster deployment of fixes across hospitals		Can lead to loss of job	System is hard to use	System frustrating my work	We will achieve less using the system
R1	4	4	4	4	3	3		1	1	1	1
R2	4	4	4	4	4	4		0	0	0	0
R3	3	4	4	3	3	3		3	3	3	3
R4	2	4	4	3	2	2		3	3	2	1
R5	2	3	3	3	3	3		3	2	1	1
R6	3	3	4	3	4	4		1	1	1	1

R7	3	4	2	1	2	2		3	2	3	3
R8	4	3	3	3	4	4		1	2	2	3
R9	4	2	2	2	3	3		3	2	2	2
R10	1	2	2	2	1	1		4	4	4	4
R11	4	3	3	3	4	3		1	1	1	1
R12	1	2	3	3	2	1		3	3	2	2
R13	1	2	3	3	3	3		4	4	3	3
R14	2	4	4	3	4	4		3	3	3	3
R15	2	2	3	3	3	4		2	2	2	2
R16	3	3	3	3	3	4		1	1	1	1
R17	3	4	3	2	3	4		2	1	2	1
R18	3	4	3	3	3	4		1	1	1	1

Appendix 3: Sample code

(a) String matching algorithm

```
public static int DamerauLevenshteinDistance(string source, string target)
        {
            if (String.IsNullOrEmpty(source))
            {
                if (String.IsNullOrEmpty(target))
                {
                    return 0;
                }
                else
                {
                    return target.Length;
                }
            }
            else if (String.IsNullOrEmpty(target))
            {
                return source.Length;
            }

            var score = new int[source.Length + 2, target.Length + 2];

            var INF = source.Length + target.Length;
            score[0, 0] = INF;
            for (var i = 0; i <= source.Length; i++) { score[i + 1, 1] = i; score[i + 1, 0] = INF; }
            for (var j = 0; j <= target.Length; j++) { score[1, j + 1] = j; score[0, j + 1] = INF; }

            var sd = new SortedDictionary<char, int>();
            foreach (var letter in (source + target))
            {
                if (!sd.ContainsKey(letter))
                    sd.Add(letter, 0);
            }

            for (var i = 1; i <= source.Length; i++)
            {
                var DB = 0;
                for (var j = 1; j <= target.Length; j++)
                {
                    var i1 = sd[target[j - 1]];
                    var j1 = DB;

                    if (source[i - 1] == target[j - 1])
                    {
                        score[i + 1, j + 1] = score[i, j];
```

```
                    DB = j;
                }
                else
                {
                    score[i + 1, j + 1] = Math.Min(score[i, j], Math.Min(score[i + 1, j], score[i,
j + 1])) + 1;
                }

                score[i + 1, j + 1] = Math.Min(score[i + 1, j + 1], score[i1, j1] + (i - i1 - 1) +
1 + (j - j1 - 1));
            }

            sd[source[i - 1]] = i;
        }

        return score[source.Length + 1, target.Length + 1];
    }
```

(b) Code for Screen Errors identification

```
protected override void Run()
        {
            Console.WriteLine("\nScreen_Error_Identifier has arrived");

            //create working folders
            Directory.CreateDirectory(sWorkingFolder);
            Directory.CreateDirectory(sErrorsFolder);
            Directory.CreateDirectory(sTempFolder);
            Directory.CreateDirectory(sFixesFolder);

            //Load the list of known errors
            ErrorKB = LoadErrors();

            //Remove any files left during the last execution
            DeleteAllFilesInFolder(sWorkingFolder);
            DeleteAllFilesInFolder(sTempFolder);

            //Start today's business
            for (; ; )
            {
                try
                {
                    if
(ActiveWindowTitleReader.GetActiveWindowTitle().ToString().ToLower().IndexOf(sApp.ToLo
wer()) != -1)
                    {
                        //Printscreen active window
                        string sPath = PrintScreenActiveWindow();

                        //Do a full print screen and put in Temp folder
                        PrintScreen(sPath.Replace("Working", "Temp"));

                        //Extract the image text
                        string sErrorText = ExtractImageText(sPath);

                        //Check if we are still dealing with the current screen
                        if (sPreviousErrorText != sErrorText)
                        {
                            if (CheckIfErrorHasOccurred(sErrorText) == true)
                            {
                                //Copy the file in temp folder
                                File.Copy(sPath.Replace("Working", "Temp"), sPath,
true);
```

```
                                if (sPreviousErrorKeywords != sErrorKeywords)
                                {
                                    //Save error in database
                                    string sFileName =
Path.GetFileNameWithoutExtension(sPath);
                                    int iRecordID = SaveError(0, sFileName,
"ScreenError", sErrorText, sErrorsFolder, Environment.MachineName, false,
DateTime.Now);

                                    //Copy image to Errors folder
                                    string sTempPath = sPath.Replace("Img_",
iRecordID.ToString() + "_");
                                    File.Copy(sPath, sTempPath.Replace(sWorkingFolder,
sErrorsFolder), true);
                                }

                                File.Delete(sPath.Replace("Working", "Temp"));
                            }

                            //Keep a copy of the current text
                            sPreviousErrorKeywords = sErrorKeywords;
                            sPreviousErrorText = sErrorText;
                        }

                        //Delete the file in working folder
                        File.Delete(sPath);
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("\nScreen_Error_Identifier: The following error
has occured: " + ex.Message);

                    if (ex.Message.ToLower().IndexOf("retrieving the com class
factory") != -1)
                    {
                        Console.WriteLine("\nScreen_Error_Identifier: The
Screen_error_Identifier will now run as a separate program");

Process.Start(@"C:\Dropbox\ErrorReporter\Agents\ScreenErrorIdentifier\ScreenErrorsIden
tifier.exe");
                        break;
                    }
                }
            }
        }
```

(d) Error reporting using Dropbox

```
protected override void Run()
        {
            Console.WriteLine("\nError_Reporter has arrived");

            int iRecordID = 0;
            string sErrorName = string.Empty;
            string sErrorText = string.Empty;

            for (; ; )
            {
```

```
                Thread.Sleep(20000);

                Console.WriteLine("\nError_Reporter: Checking if new errors have been
identified...");

                try
                {
                    DataTable Errors = LoadIdentifiedErrors();

                    for (int i = 0; i < Errors.Rows.Count; i++)
                    {
                        iRecordID = Convert.ToInt32(Errors.Rows[i]["id"]);
                        sErrorName = Errors.Rows[i]["ErrorName"].ToString();
                        sErrorText = Errors.Rows[i]["ErrorText"].ToString();

                        string sErrorDetails = iRecordID.ToString() + "|" + sErrorName
+ "|" + "Category not specified" + "|" + sErrorText + "|" + Environment.MachineName;

                        //Put fix in local dropbox folder
                        File.WriteAllText(sErrorsFolder + iRecordID.ToString() + "_" +
sErrorName + ".error", sErrorDetails);

                        SaveError(iRecordID, "", "", "", "", "", true, DateTime.Now);

                        Console.WriteLine("\nError_Reporter: Identified error
reported");
                    }
                }
                catch (Exception ex)
                {
                    Console.WriteLine("\nError_Reporter: The following error has
occured: " + ex.Message);
                }
            }
        }
```

(e) Code for Error fixing

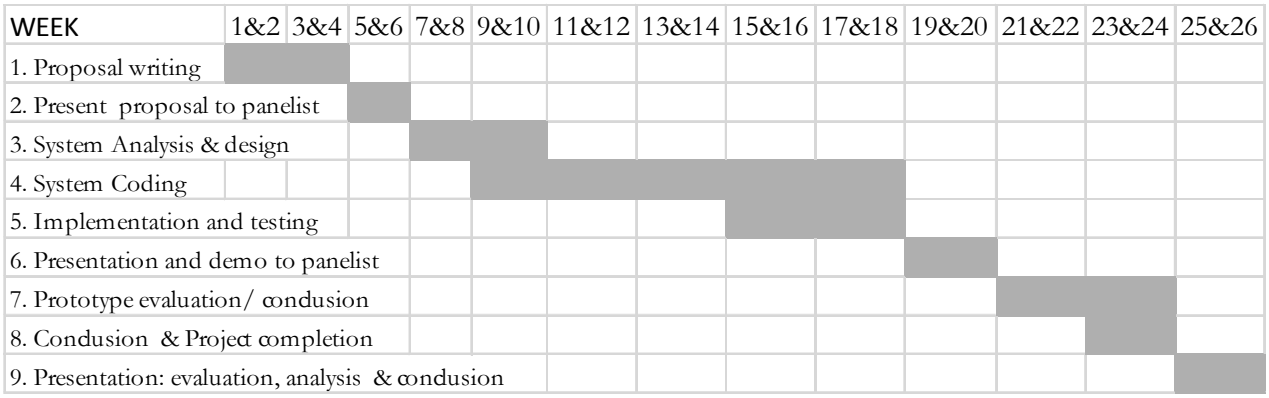
```
private static void ApplyFixes()
        {
            List<sy_LoadActiveFixesResult> oActiveFixes =
oDAL.sy_LoadActiveFixes().ToList();

            for (int i = 0; i < oActiveFixes.Count; i++)
            {
                try
                {
                    if (oActiveFixes[i].FixType == "Script")
                    {
                        //Run database script
                        RunDatabaseScript(oActiveFixes[i].FixSource);
                    }
                    else if (oActiveFixes[i].FixType == "Filepatch")
                    {
                        //Copy patch files
                        CopyPatchFiles(oActiveFixes[i].FixSource,
oActiveFixes[i].FixDestination);
                    }
                    else if (oActiveFixes[i].FixType == "Executable")
                    {
```

```
                //Execute the installation
                RunInstallMSI(oActiveFixes[i].FixSource);
            }

            oDAL.sy_UpdateFixStatus(oActiveFixes[i].id, true, DateTime.Now);
        }
        catch (Exception ex)
        {
            Console.WriteLine(ex.Message);
        }
    }
}
```

Appendix 4: Project Schedule



Appendix 5: Project Budget

Item	Units	Cost per Unit (Kshs)	Total Cost (Kshs)
Wireless Router	1	10,000	10,000
Computers	3	40,000	120,000
Modems	2	2,500	5,000
MS Office 2007	1	2,500	2,500
Dropbox Software	1	0	0
RDBMS(SQL Server express)	1	0	0
Total			**127,500**

Appendix 6: How to run the system

1. Install Microsoft office 2007, ensuring that Microsoft office Document Imaging is installed.

2. Install Dropbox software and create a Dropbox folder in Drive C of the computer
3. Install SQl Server 2008, and then run the create database script (Create database script.sql) in SQL server management studio
4. Install the Agent Host application in every computer in the network
5. Install Agent Client application in the computer where the agents will be launched
6. Start the agent Host application in all the computers in the network
7. Start the agent Client application and use it to send agents to the computers on the network