INVESTIGATING END USER ERRORS IN OIL AND GAS CRITICAL CONTROL SYSTEMS

LAYTH NABEEL ALRAWI

JUNE 2019

INVESTIGATING END USER ERRORS IN OIL AND GAS CRITICAL CONTROL SYSTEMS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES OF
ÇANKAYA UNIVERSITY

BY
LAYTH NABEEL ALRAWI

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE
IN
THE DEPARTMENT OF COMPUTER ENGINEERING
INFORMATION TECHNOLOGY PROGRAM

JUNE 2019

Title of the Thesis: **Investigating End User Errors in Oil and Gas Critical Control Systems**

Submitted By **Layth Nabeel ALRAWI**

Approval of the Graduate School of Natural and Applied Sciences, Çankaya University.

[illegible]

Prof. Dr. Can ÇOĞUN
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

[illegible]

Prof. Dr. Erdoğan DOĞDU
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

[illegible]

Assoc. Prof. Dr. Ö. Tolga PUSATLI
Supervisor

Examination Date: 26.June.2019

Examining Committee Members

Assoc. Prof. Dr. Murat KOYUNCU	Atılım Univ.	[illegible]
Assoc. Prof. Dr. Hakan MARAŞ	Çankaya Univ.	[illegible]
Assoc. Prof. Dr. Ö. Tolga PUSATLI	Çankaya Univ.	[illegible]

STATEMENT OF NON-PLAGIARISM PAGE

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last Name: Layth Nabeel ALRAWI

Signature: [illegible]

Date: 26.July.2019

ABSTRACT

INVESTIGATING END USER ERRORS IN OIL AND GAS CRITICAL CONTROL SYSTEMS

ALRAWI, Layth Nabeel

M.Sc., Department of Computer Engineering
Information Technology Program

Supervisor: Assoc. Prof. Dr. Ö. Tolga PUSATLI

JUNE 2019, 84 pages

System availability and efficiency are critical aspects in the oil and gas sector; as any fault affecting those systems may cause operations to shut down; which will negatively impact operation resources as well as costs, human resources and time. Therefore, it became important to investigate the reasons of such errors. In this study, software errors and maintenance are studied. End user errors are targeted after finding that is the number of these errors is projected to increase. The factors that affect end user behavior in oil and gas systems are also investigated and the relation between system availability and end user behavior are evaluated.

An investigation has been performed following the descriptive methodology in order to gain insights into the human error factor encountered by various international oil and gas companies around the Middle East and North Africa. This was conducted by distributing a questionnaire to 120 employees of the companies in this study; 81 had responded. The questionnaire contained questions related to software/hardware errors and errors due to the end user.

In short, the study shows that there is a relation between end user behavior and system availability and efficiency. Factors including training, experience, education, work shifts, system interface and I/O devices were identified in the study as factors affecting end user behavior. Moreover, the study contributes new knowledge by

identifying a new factor that leads to system unavailability, namely memory sticks. This thesis presents a valuable knowledge that explains how errors occur and the reasons for their occurrence.
Major limitations of this research include company policies, legal issues and information resources.

Keywords: Software errors, Faults, Failure, Oil and gas, Human error, Human mistake, End user error, Copy-paste error, Torque turn system, Tubular running services, Casing Running tools, Critical control systems, casing and tubing.

ÖZ

PETROL VE GAZ KRİTİK SİSTEMLERİNDE SON KULLANICI HATALARININ ARAŞTIRILMASI

ALRAWI, Layth Nabeel

Yüksek Lisans, Bilgisayar Mühendisliği Anabilim Dalı
Bilgi Teknolojileri Programı

Tez Yöneticisi: Doç. Dr. Ö. Tolga PUSATLI

Haziran 2019, 84 sayfa

Sistem kullanılabilirliği ve verimliliği petrol ve gaz endüstrisinin kritik parçalarıdır. Bu sistemleri etkileyecek herhangibir hata, operasyonları durdurabilir ki bu da para, insan ve zaman kaynaklarını boşa harcatır. Bu nedenle, bu hataların sebeplerinin incelenmesi önemdir. Bu çalışmada, yazılım hataları ve bakım işlenmiştir. Bu hataların arttığının görülmesinden sonra da son kullanıcılar incelenmeye başlanmıştır. Petrol ve gaz sistemlerindeki son kullanıcı davranışlarını etkileyen faktörler de araştırılmış ve sistem kullnılabilirliği ve son kullanıcı davranışları ilişkileri değerlendirilmiştir.

Ortadoğu ve Kuzey Afkika'da çeşitli uluslararası şirketlerde rastlanan insan hatası faktörünü anlamak için betimsel araştırma yöntemi kullanılmıştır. Burada, 81'nin geri dönüş yaptığı 120 çalışanla anket çalışması yapılmıştır. Ankette, yazılım, donanım ve son kullanıcı hatalarını içeren sorular bulunmaktadır.

Kısaca, çalışma, son kullanıcı davranışıyla sistem kullanılabilirliği ve verimliliği arasında bir bağ olduğunu göstermiştir. İş eğitimi, deneyim, eğitim, çalışma zamanları, sistem arayüzü ve girdi/çıktı donanımlarını içeren etkenlerin, son kullanıcı davranışlarını etkilediği saptanmıştır. Buna ek olarak, bellek çubuklarının da sistem bozulmalarında etkin olduğu yeni bir bilgi olarak bulunmuştur. Bu tez, hataların nasıl ve neden oluştuğunu açıklayan bilgiler sunmaktadır.

Şirket politikaları, yasalar ve edililen bilgilerin kaynakları, araştırmanın kısıtlamaları sayılabilir.

Anahtar Kelimeler : Yazılım hataları, arıza, bozukluk, petrol ve gaz, insan hatası, son kullanıcı hatası, kopyala-yapıştır hatası, tork çevirme sistemi, tork dönüş sistemi, boru indirme servisi

ACKNOWLEDGEMENTS

First, I wish to express my thanks and gratitude to Allah, the most Gracious, and the most merciful for uncountable help guidance. Then, I would like to pay tribute to my supervisor Assoc. Prof. Dr. Ö. Tolga PUSATLI for his guiding hand throughout the duration of my study. His patience, motivation, enthusiasm, immense knowledge, guidance, professionalism, charm and reasonableness helped me overcome seemingly insurmountable obstacles along the way. You are a role model to follow.

From Cankaya University, thanks are due to Prof. Dr. Mahir Nakip for his advises throughout the analysis period.

From Oil and Gas sector, thanks are due to Mr. Nameer Khalil Ibrahim and Mr. Ali Nabeel Khalil for their support throughout the validation of questionnaire questions.

I extend my acknowledgement and heartfelt love to my mother, brothers, sister and wife, for continuous supporting along the period of my study.

Last but not the least; I dedicate this thesis to the memory and soul of my father (Nabeel Khalil Al-Rawi) and my uncle (the Colonel Martyr: Ibrahim Khalil Al-Rawi). They are a role model to follow.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

BGDU	Bar Graph Display Unit
CRT	Casing Running Tools
DumpT	Dump Torque
EOK	Enhanced Oil Recovery
GUI	Graphical User Interface
High ShT	High Shoulder Torque
HPT	Hydraulic Power Tong
HPU	Hydraulic Power Unit
Low ShT	Low Shoulder Torque
MaxT	Maximum Torque
MENA	Middle East and North Africa
MinT	Minimum Torque
MWD	Measurement While Drilling
NOV	National Oil Varco
O&G	Oil and Gas
optT	Optimum Torque
R	Relation
refT	Reference Torque
TTS	Torque Turn System
TTT	Torque Turn Technician

CHAPTER ONE

INTRODUCTION

It is well-known that in this era, software is used in many sectors, including sectors such as oil and gas, military, business, financial, and health. Software products are usually accompanied by various types of error making our life continually hazardous. Studies show that 50% to 70% of projects are fails, and if we were to assume that only 5% of projects fail, it would mean billions of dollars being wasted every year [5]. Any computing system may encounter errors, such as inappropriate requests from supported applications, or unexpected behavior from malfunctioning or misconfigured hardware. If a system's software does not recover from these errors correctly, more serious failures may occur, including crashes or vulnerability to attack by malicious users [104].

1.1 Researcher Motivation

The researcher is an oil and gas sector employee with long experience in maintaining and operating oil and gas infrastructure systems, especially torque turn systems. The researcher is also interested in the errors and faults that occur over operational time as such problems can increase the cost of operations, create unavailability of systems and risk of operational delays. These errors may lead to shutting down operations. The researcher has experienced a number of faults some of which are due to technicians' and operators' typos. Meanwhile, some other faults may occur due to operating system, hardware and/or software errors. Therefore, it is important to investigate the factors that cause such errors, especially user faults and to provide solutions for oil and gas firms to break down those faults in order to increase the availability of the systems.

Before we proceed, providing basic information on an operation site would increase readability of the dissertation.

The oil and gas world is broad and complex. In order to drill an oil or gas well, various types of services and people are needed. Drilling rigs start drilling operations from scratch and move through many stages using various services, such as drilling, well constructions, logging, cementing, chemicals, completion and production, until such time that the well is created and starts pumping oil for export or for storage lines. These services can be performed using various types of equipment, including hydraulic, mechanical, electrical and so on. Moreover, these services are required for various types of critical infrastructure systems in addition to experts in each type of service.

1.2 Purpose of the Study

The primary aim of this thesis is to identify and investigate the key factors slowing/interrupting the operations of Torque Turn Systems and leading those systems to failure. We focus especially on the factors which occur due to the end user, i.e., operators and technicians are investigated. We aim to understand how they are doing their jobs and their behavior toward the systems in terms of ease of use, working faster and making fewer errors. This thesis aims to provide suggestions to assist oil and gas firms to break down the faults that occur in their torque turn systems due to end users, thereby improving quality by providing key performance factors such as periodic training and certification for technicians.

1.3 Scope of the Study

As a case study, we selected multinational TTS technicians around MENA in addition to various types of torque turn systems related to different international oil and gas firms which usually work on updating their systems and trying to have properties to reduce the risk of faults and avoid the costs of interruptions.

The scope of the study includes a review about and investigation into the potential faults that critical oil and gas infrastructure systems may encounter during times of operation, especially torque turn systems, in addition to investigating faults that can occur due to human users, especially the end user.

1.4 Research Question

“Can end user behavior influence oil & gas operations availability and efficiency in TTS?”

1.5 Research Methods

As the study aims to clarify user behaviors and how users perform their tasks, we have chosen to collect information from end users directly. On our method, we prepared structured questionnaires to be completed by the users in our study. The questions have emerged from the surveyed literature both on failures due to hardware/software errors and human factors.

There are similar works to ours presented in Chapter II, however, none of them, to our knowledge, has the survey questions suitable for our field. On the other hand, we have benefitted from the literature to investigate and formulate the questions thanks to the experience of the researcher.

This research can be identified as a descriptive study in which we are endeavoring to determine the profile of end-users, their intentions hence behaviors in performing their tasks. Because the researcher is already employed in the field, we had the opportunity of benefitting from the researcher’s experience. This is termed as participant observation in the literature [105]. As the name implies, the researcher personally participates in the events and fields that are being investigated.

Finally, this research is supported by the literature, end-users’ feedback and the researcher’s personal participation.

1.6 Thesis Outline

This thesis contains five chapters: an Introduction, a Background and Literature Review, Oil and Gas Well Site and Work Force, Analysis and Results, Findings, Future Work and the conclusion of the study.

The content of each chapter is outlined as follows:

Chapter 1: Introduction

Chapter 1 is a quick and general review of the research which clearly states the motivation, purpose and scope of the study as well as the research question. The outline of this research is presented at the end of this chapter.

Chapter 2: Background and Literature Review

This chapter is divided into two parts. The first part (2.1) presents the background of software errors and their causes, their reasons and measurements. It also focuses on the importance of the performance of maintenance. The second part (2.2) presents a literature review that covers the major works on errors and how to handle them, categories and classifications of software errors and software maintenance. The second part also focuses on human error and user mistakes.

Chapter 3: Oil and Gas Well Site and Work Force

Chapter 3 describes oil and gas well site operations, the roles of actors, positions and the actions with the associated information required to complete the drilling operations of each well and the role and importance of using the torque turn system in these operations. Furthermore, the use case is presented in this chapter to handle this information. Chapter 3 concludes with a discussion.

Chapter 4: Analysis and Results

Chapter 4 summarizes the method that was used to gather the data, the results that were yielded through this method and the statistics that were used to obtain those results.

Chapter 5: Findings and Future Work

Chapter 5 summarizes the findings of this research, addresses the limitations that may restrict the findings and outlines possible future work. The conclusion and answer to the research question are presented at the end section of this chapter.

CHAPTER TWO

BACKGROUND AND LITERATURE REVIEW

2.1 Errors and Their Causes

This section presents the definitions of the terms covered in this thesis as well as preliminary knowledge to provide essential background on software errors and software maintenance. We cover this as the literature being reviewed is closely tied to these concepts.

2.1.1 Software Errors

We can define software errors as events that occur through program operations and lead to undesirable outcomes, including exceptions due to divisions by zero. Such errors may settle inside the software for long time without any activity until a particular event occurs and activates it [1]. Errors occur as a result of mistakes in documentation, code and the software. These errors may lead to faults or bugs, and eventually failure of the software, as shown in Figure 1 [2].

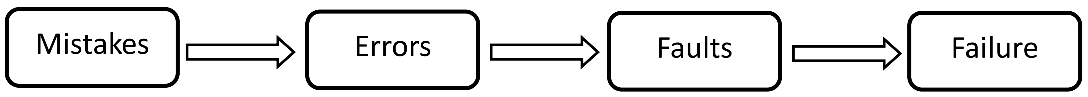


Figure 1 Software failure life cycle [2]

Software faults or bugs are the software's deviations from its required operations path. Software failure is a symptom that appears though software execution and may lead to operational crashes [3]. A system error is a mistake that occurs in the program code at the time of execution and leads to system faults which are the behaviors of the system leading to system failure. System failure is an event that occurs when system behavior is not compatible with design requirements [54].

System errors are also defined as the status when current system behavior does not match with the actual behavior for which the system was designed. Additionally, two types of error have been shown, the first of which is external errors caused by the system platform and the second of which is internal errors caused by programming mistakes [4]. Errors can arise due to the cognitive framework, and these errors can be divided into perception errors (which include errors in visual detection, visual search and listening), memory errors (which include forgetting (or miscalling) temporary or longer-term information, forgetting previous actions, and forgetting planned actions), judgment, planning and decision-making errors (which include errors in judging aircraft trajectories, making decisions and planning), and action execution errors (which include actions or speech performed notes-planned) [75]. There are three major components of an error: (1) External Error Mode (EEM), which are errors pertaining to external behavior, such as opening the wrong valve; (2) Performance Shaping Factor (PSF): These errors relate to the human interface, training and time pressure; and (3) Psychological Error Mechanism (PEM), which are the errors pertaining to internal behavior such as memory failure [80].

2.1.2 Classifications of Software Errors

Errors can be classified according to many criteria. For example, about four decades ago, [1] classified errors based on their impact on program operation. This depended on the intensity of the impact on its success; for instance, a high impact will lead to a complete software crash; a medium impact will cause a decay in performance; and a low impact will disperse the software operation. Software errors can be classified according to their programming and hardware features. A more recent classification, [4], depends on the source that causes an error. There are two sources of error, first being external sources such as the errors occurring through platform changes, and the second being internal sources such as programming mistakes and design errors.

2.1.3 Software Error Categories

It is not easy to count the number of failed projects. Through the research, we have experienced many types of software errors and faults most of which are the results of design inefficiencies and code mistakes, as given in [5]. These errors increase and become more complex whenever we increase the number of persons and modules involved in software design. According to many firm reports, failures start with the presentation of imprecise information and end with operation crashes [5]. The most common types of error are persistent errors, transient errors, design errors, performance errors, design erosion and software aging. We visit these errors in the following sections of this chapter.

2.1.3.1 Persistent and Transient Errors

The difference between these two approaches is the live cycle interval and in order to find if the error is persistent or transient, we have to take into account many factors, such as the event existence period, the detection situation, the environment situation, the logical appearance and statistical features [9].

The persistent error is a type of error that evades detection efforts and begins operating during the software execution period. It appears during software operation and operates for a long time without disappearing. These errors are a result of bad design due to the poor prediction of the requirements for which the current software is designed or the programmer’s experience being insufficient to envisage the magnitude of the problem and cover the requirements of complex applications. These types of error are large errors and cause loss of time and economic [7].

Transient errors, also known as non-persistent errors, are those which appear during the software operational period and disappear later [7]. They appear many times during software operation and cause faults in software components which in turn leads to incorrect outcomes. A number of transient errors are considered to be persistent errors, i.e., errors that occur in operating systems [6].

The multiple occurrences of these errors make repair efforts challenging because we have to perform testing many times during the software operations and we have to test the results at each point of the software and compare it with the original expected outputs. These errors usually appear in GUIs caused by some GUI classes [8].

Many studies were performed to test the occurrence of persistent and transient errors in a number of applications, such as Paint; Excel and Word. The studies showed the large number of transient errors occurring throughout the test period, as shown in Figure 2 [8].

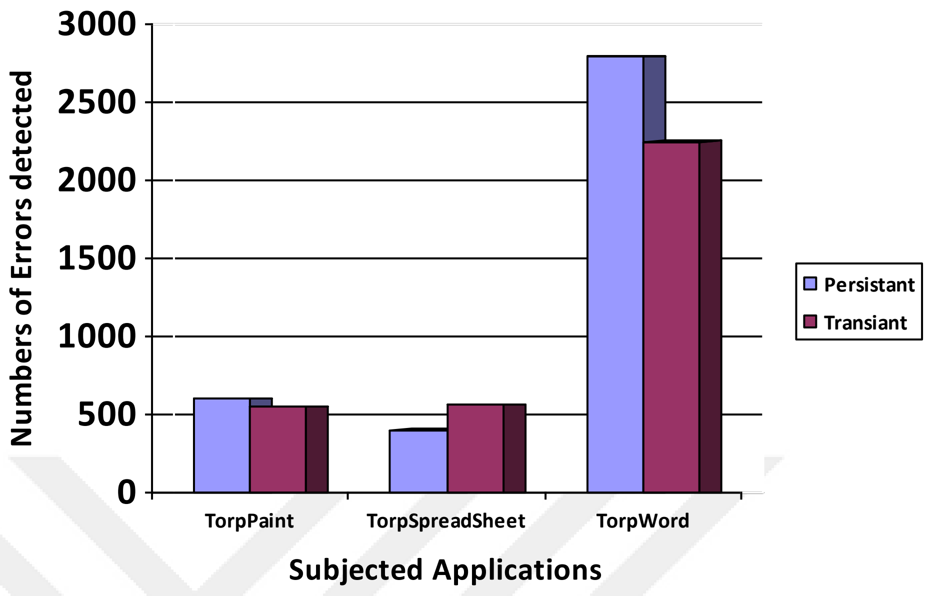


Figure 2 Numbers of persistent and transient errors [8]

2.1.3.2 Performance Errors

Depending on input value, an application may present different behaviors which may include one of the performance error factors [11]. Performance errors are programming errors that slow down software operation [12]. Performance errors occur as a result of programming mistakes attributable to poor programming skills as many applications are designed by junior programmers using different software engineering techniques or because of a "fix it later" approach. This makes errors difficult to be discovered and fixed during times of debugging thereby causing losses in cost and time. Therefore, it is better to find and fix errors during the designing phase [13]. Moreover, performance errors can occur due to data sharing, resource sharing variety and unbalanced remote access [14].

Fixing this type of error is difficult because the repair process has advantages and disadvantages and programmers need to go through the advantages only without evoking the disadvantages. Moreover, the separation process of the advantages and disadvantages is a challenging process. Performance errors slow the software and reduce the response time of the software [12]. User experience is a set of behaviors and the emotions of using software products; therefore, any bug affecting system speeds or times will help to increase the number of performance errors and thus cause poor user experiences as it is critical to know how users perceive software quality. This also leads a degradation in the quality of the software [96] [97].

Schlumberger is an international oil and gas company that is always looking to enhance its software in order to improve its performance and usability and increase the user experience. This is what happened with the Schlumberger Techlog Wellbore Software Platform as they released the latest version of this application with new features, taking care to improve performance and display interaction in the application [99].

Other examples of performance errors are those that occur in Adobe Flash CS4 Professional (10.0.2), including performance issues when dragging objects on stage, scrubbing the timeline, entering symbol edit mode in large AS 2.0 and AS 3.0 files, performance issues when opening large files or files with many nested symbols, and performance issues when working in the library such as scrolling, selecting items in the library, dragging items to the stage, and editing symbols from the library [21].

2.1.3.3 Design Erosion

Design erosion occurs because of continuous change and update of the software design. This causes erosion for the software's original design. This error forces programmers to issue new code and retire other parts. Sometimes, programmers have to redesign the software anew, which is a difficult process and causes loss in time, cost and personal effort [15]. The wholesale system replacement from scratch is dangerous because it has major effect on technology and people as new systems may require users to be trained in using the new system and the new system may lack some of the functionality of the old system. Moreover, it will have an impact on business as the purchase of new systems or developing current systems may impact company budgets. The human and technology point of view is effected because replacement may lead to a lack of some functionality resulting from the replacement process. The replacement will contain new components and this will necessitate retraining users in order to qualify them for the new components. At the same time, businesses are also affected because the replacement may require new skills and costs, implying a low return on investment. Therefore, development maintenance is preferred to improve a system without discarding an existing system [16].

Design erosion leads to a low-quality and low-productivity system, increased time to publish and software that is difficult to maintain [17]. Usually, such errors occur in open source software because this type of software continues to evolve [15].

Mozilla browser is given in [15] as an example of design erosion as it is one example of open source software. After half a year of continuously working, Netscape developers discovered that the Mozilla browser was suffering from design erosion caused by evolvement. The developers attempted to fix this issue without success. They finally decided to redesign the software anew. However, after many years of running, there has been a large amount of code added to the application and some parts were retired about 20 years ago.

2.1.3.4 Software Aging

Software aging has been occurring in most software systems that work continuously for long periods of time [58]. Software aging is defined as the accumulation of software errors and failure to perform as a user intends, as software, similarly to humans, can age [59]. Software aging is caused by the exhaustion of operating system resources. It starts with faults and ends with crashing. Normally, software does not age immediately after it starts running; however, it arises after a period of time as the software at the beginning of operations works according to the required specifications. However, after some time of operation, the software may not work according to specifications causing faults and thus system crashes [18]. System outages occur more due to software faults than to hardware faults [48].

Two kinds of aging are reported in the literature, the first of which occurring as a result of changes in platform, such as hardware component replacement or upgrading an operating system and unreleased software locks. Such changes make subsystems appear old despite there being no aging problem. The second kind of aging occurs as a result of software operating for a long time, which causes a lack in performance and ends with unplanned system crashes, including memory leakage and data corruption [18, 19, 49 and 50]. Software ages for two reasons, first being poor adaptation to the environment and the second being continuous modification to the software [61].As shown in figure 3, there are two stages of software aging. First, the software starts its normal operation and the potential for failure at this stage is zero (S0). After some time of operation, there is a possibility of failure occurring (Sp) and the system will suffer from a fault until the crashing stage thus ensuring the failure (Sf) [18].

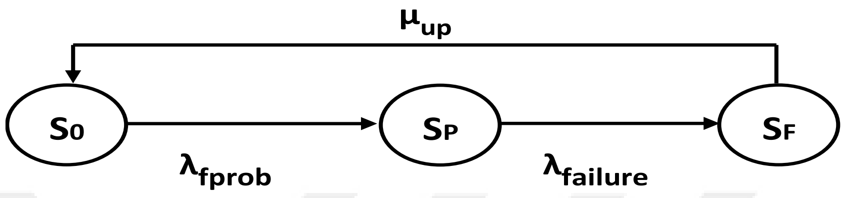


Figure 3 Stages of software aging [18]

Aging errors can be fixed through rejuvenation and by using suitable maintenance methods such as adaptive and perfective maintenance [18] [19]. Mobile device software is the best example of software running for a long time. Generally, mobile devices suffer of software aging because of their continuous running and use of many resources. Therefore, the cloud was the best way to extend the operational life and prevent software aging by offloading data from mobile devices to a cloud server [20].

One example of an aging error is a LOWIS application, which is a well services management application from the Weatherford Company in the oil and gas sector. Some errors occur through the running time leading to faults, and after all the automatic refreshing attempts, an entire running system stops. Therefore, the system was forced to restart even though the data of the last process performed may be lost [29].

2.1.3.5 User Mistakes and Human Errors

The user is the major factor that introduces software faults and can be key to understanding fault diversity [3]. Nowadays, many people are coding and most design programs as a hobby and use different tools and languages to locate errors in their programs [13]. Human error can occur in any phase of the software development lifecycle (SDLC), which is defined as any human activity leading to not achieving the goals of the system [69, 70 and 71]. User mistakes usually occur in many systems, such as wrong outputs as a result of using wrong inputs. Moreover, professional programmers make mistakes and they usually have errors in their coding [43]. Many errors can be made by the end user. Such errors are broadly divided into procedural errors and proficiency errors. Procedural errors are those which occur while carrying out prescribed or normative sequences of action, while proficiency errors are related to a lack of skills, experience or practice [51] [52]. Human errors can be divided into slips and mistakes. A slip error is a type of procedural error occurring due to perceptual reasons wherein a subject may not pick up cues or may inadvertently forget some details. For instance, the user may have knowledge about the correct action but he fails in execute it, while mistakes are a proficiency error which occur due to lack of user experience. They occur when the execution process is correct but there is something wrong in the planning such as when the user does not have knowledge about the correct action [51] [53]. Human behavior plays a significant role in error occurrence. Human behavior is based on rules, skills and knowledge. Rules-based errors rely on incorrect rules and procedures, while knowledge-based errors occur when there are no procedures or rules in the environment. Skills-based errors occur when the wrong intention results in inappropriate execution of the plan [69, 72 and 73].

Human error can also be divided into two categories: commission and omission [63]. Error of commission can occur when one performs an unrequired task as the operations usually force the operators to perform actions which are not required or out of his scope of work [101]. It also occurs when human behavior in the correct manner performs a task but the system status is not compatible with user behavior [103]. On the other hand, error of omission means that the operator fails to follow the procedure to perform his task; this is the most common human error and it is the result of a number of symptoms, including lack of experience, time pressures and lack of required staff, all of which are symptoms that may introduce later errors [102].

2.1.3.6 Copy-Paste Errors

The end user is always interested in moving data using the copy-paste activity. They copy data from a source and then paste it into a destination to reduce writing time and to avoid transaction errors. However, this activity may lead to other errors affecting productivity [100]. Errors usually occur in coding when using copy-paste activity. This task sometimes leads to duplicate data, which occurs when copying and pasting so as to reduce the time and efforts by using the same code many times rather than rewriting fresh code [68].

2.1.4 Software Maintenance

Software errors are common and it is rare to find error-free software. It is important to find a way to fix these errors but each error may occur due to another error and a current error may introduce another error for which it becomes necessary to determine the main reason so a feedback mechanism can be created [1] [4] [19].

Software maintenance is a useful process that has been found to fix these errors; however, this process needs system monitoring to identify any problems and it needs special tools and techniques [23]. Researchers expend much effort on maintenance in order to find, fix and prevent future errors [19].

Software maintenance is a major concern in the world of military embedded systems. For example, the software development cost for the US. F-16 airplane was $85 million. However, the estimated cost of maintaining the software is $250 million, as reported in [37]. Software maintenance is the modification process of software after delivery or publishing in order to improve its performance, to adapt it to environmental changes or to meet user requirements. It is a vital process without which most of existing software would fail and become inefficient for organizations. Therefore, around 80% or even more than 90% of programming efforts and organizational budgets go to software maintenance [34].

The maintenance process passes through a sequence of activities, in which firstly, an organization sends a fault report to developers and the developers diagnose the fault and design suitable maintenance methods and identify the code that needs change.

They have to know why an error occurs, when an error occurs, who made the error, and where the error occurred. They have to determine whether they can freeze the error and prevent it. If they cannot do so, they have to find a way to detect it [24]. Second, developers apply the maintenance process and test the software, and finally update any documents and retrain users for the new design [22]. The success of these activities depends on how maintainable the software is. It also depends on the technique used throughout the maintenance process as well as on user and organization behavior [19].

The literature shows another definition of software maintenance, which is the accumulation of all activities performed to provide cost-effective support to the system. These activities perform before and after the software delivery [40]. Software maintenance can also be defined as the correction of errors and implementation of new modification to enable the software to work with any new changes and requirements. The literature also shows that software maintenance should have a special procedure, especially when dealing with aging phenomena [62].

Software maintenance is a critical process but not all maintenance processes succeed. Sometimes, the process fails and leads to calamitous disasters because the process may pass through undesirable consequences [25]. The maintenance process may encounter a number of problems, including environmental problems such as resource leaks, management problems, personal problems such as lack of skill and experience, as well as user requirement problems due to maintenance processes possibly adding or removing a number of user requirements [22].

Software must be continuously evolving and maintained in order to fulfill user requirements and to improve its quality. Whenever this evolution and maintenance is interrupted for any reason, it means that software quality may be corrupted, a phenomenon known as software aging. Aging phenomena has been founded in different aspects of life such as insulation systems, textiles and physics. They show how their behavior is similar to aging in software.

Software aging can be defined as the degradation of software performance over time for which there are two reasons: the effect of changes made to the software and the failure to perform the changes according to the software owner’s requirements.

Software maintenance and evolution can be performed on software in order to fulfill user needs and satisfaction or to adapt the software to environmental changes. However, this continuous evolution may cause degradation in software quality and performance, a phenomena called software aging [62].

Many factors impact the maintenance process, including maintenance task size, the type of maintenance to be performed, and the maintainer’s dependence on the expected task, the programming language being used, the priority of the task, the age and size of the system, and the maintainer’s experience [34]. When a maintenance project is completed, it means that the system has new maintainability and the team related to the project has a good working knowledge of the system.

2.1.5 Software Maintenance Categories

Normally, software that is designed to work for a long time needs to be maintained to adapt to platform changes and user requirements [38]. Software maintenance is a process that performs to correct, adapt and perfect a system in order to increase its performance to provisionally meet an organization’s requirements [40]. In order to start with the maintenance process, it is necessary to identify where in the software an error occurs and why it happened and a suitable maintenance method to implement in order to prevent it [43]. As mentioned earlier, if we wish to fix an error, we have to identify a suitable maintenance process compatible with the cause of that error. We have experienced many types of maintenance, but the ones that are most used are corrective, adaptive and perfective maintenance [19] [39] [42]. Moreover, we can use rejuvenation (Section 2.1.6) to maintain the software suffering from aging. System maintenance accounts for around 50% of the software lifecycle, with adaptive maintenance alone representing 18.2% of project efforts and time and 13.8% of maintenance falling under the adaptive maintenance category [56].

2.1.5.1 Corrective Maintenance

These are the activities used to correct errors through running the software, including design correction and code correction [25] [39]. Collective maintenance comprises all the activities that revert the software to its initial state before the occurrence of a failure [33]. It finds the error by identifying the code responsible for the error, fixing it by designing new code to replace the buggy code, and testing it to ensure that there is no error produced through the correction process [26]. Around 20% of the maintenance effort goes to corrective maintenance according to [34]. There are many reasons for this high rate of effort in corrective maintenance, including the difficulty in defining and locating an error because a system may contain millions of lines of code and many modules and components. For the size of modifications needed to fix an error, it is necessary to know whether it needs the addition of new functionality or a rewrite of many lines of code to retrieve the current functionality since the main reason for errors is system functionality failure. Long clarifications of the error due to the developers takes time to study. Another reason for the high rate of effort is to decide whether modifications of the current block of code need to be applied or whether developers have to insert new blocks of code. The last reason is that the fixing process may introduce new errors or it may need multiple fixes [34].

It is difficult to determine where errors have been occurring. Whether they occur in the code, database or user manual, there is much information to make this determination easier, such as the path through which a system passes before point of failure, the possible path through which a system may pass after failure, and the data being processed when errors occur [35]. Faults are more difficult to isolate during system/acceptance testing than during unit testing [36].

One example of this type of maintenance is the new version of the Tolteq application – IGC product (V. 1.13), a WMD software developed by the NOV Oil and Gas Company. A log pointer bug related to a memory logging module they had experienced in the previous version was fixed [32].

2.1.5.2 Adaptive Maintenance

This is the activity of performing after delivery the application of the adaption for the system to make it compatible with the platform changes. This would include the addition of new code and retiring other code, or adding new modules and remove the others [27] [39]. The number of lines of code to be changed and the number of operator changes has a major impact on adaptive maintenance efforts. The most important metrics used to estimate adaptive maintenance efforts are the class complexity and the size of the code [34].

Adapting a system to platform changes is important to improve software performance. However, repeated adaptation may lead to system decay. Through the software life cycle, several countermeasures are applied against this decay, as shown in the figure 4: (S1) assesses maintainability problems; (S2) determines a ranking of these problems; (S3) selects some of the topmost items of that ranked list; and (S4) addresses them [41].

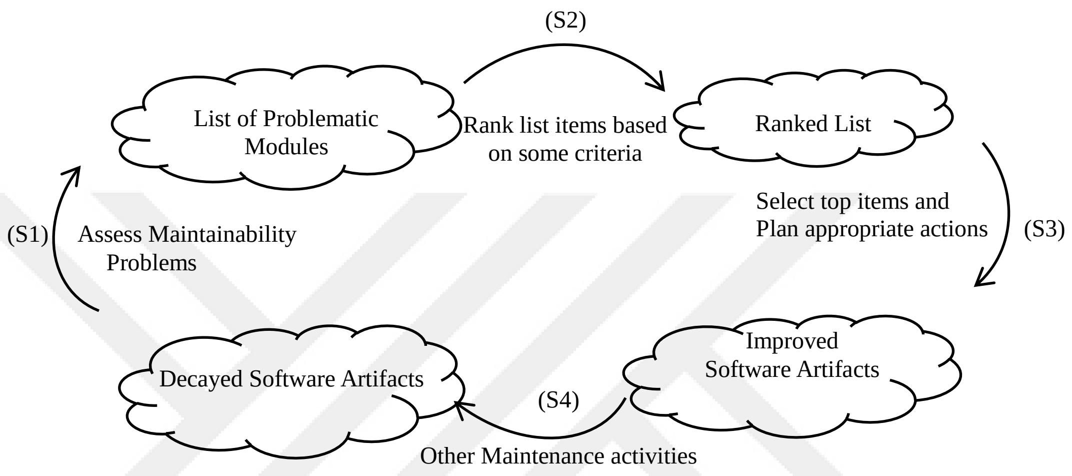

(S2)
List of Problematic Modules
Rank list items based on some criteria
Ranked List
Select top items and Plan appropriate actions
(S3)
(S1)
Assess Maintainability Problems
Improved Software Artifacts
Decayed Software Artifacts
(S4)
Other Maintenance activities

Figure 4 Software maintenance life cycle with countermeasures against software decay [41]

One example of this type of maintenance is the INTERSECT High-Resolution Simulator, software used by the reservoir engineering team at Schlumberger Oil and Gas Company. The challenge was to complete highly detailed and complex dynamic reservoir modeling with improved speed, efficiency, and accuracy to plan enhanced oil recovery (EOR) campaigns and aid corporate investment decision making The solution was run for the INTERSECT High-Resolution Reservoir Simulator to reduce runtime without compromising the quality of the results. As a result of that, they achieved their goals and this process increased runtimes up to 20 times faster than the original base case while maintaining integrity and accuracy and improving project efficiency and confidence in decision making [30].

2.1.5.3 Perfective Maintenance

These are the activities performed after delivery to achieve high performance and reliability goals with low effort and low cost [25] [39]. Studies have shown that 55% of perfective maintenance goes into perfective maintenance [35]. It is used to improve the initial statuses for systems. Perfective maintenance includes software upgradability that provides us with quickly completing and installing the initial version of the software, the possibility of uploading a software module to the system without terminating ongoing system processes thereby reducing the time and continuing the software process. Moreover, it expands the software life span and enhances its performance and reliability by keeping it up-to-date with the new technology [28].

One example of this type of maintenance is the new version of the LOWIS application (version 7), which is one of the oil and gas industry's critical infrastructure systems related to an oil and gas company (Weatherford). They have added a new service to solve one challenge occurring in the previous versions: the challenge was *"In earlier releases of LOWIS, whenever there was an update to the client files users would need to run a new installation executable to update their client software. Starting with LOWIS version 7.0 the executable you install locally, or on your Citrix server, is actually a LOWIS client updater. Instead of directly connecting to the server, it now downloads the client files from that server that will be run locally. This removes the need for end users to need to regularly update their client by manually installing newer versions, or administrators to update the client installed on the Citrix server. LOWIS will now automatically download any updated client file when you connect to the server."* [31].

Perfective maintenance involves redesigning, restructuring, redeveloping and retesting of the system. Works such as [39] underline redesigning, restructuring, redeveloping and retesting activities as they increase cost considerably most of the time.

2.1.6 Software Rejuvenation

This is the technique that terminates software running, cleaning the internal state and provides periodically proactive restarts to decrease the impact of aging, to avoid crashing and to bring it back to the initial state before any occurrence of degradation and error [44, 45 and 46]. The most important issue here is to define when, what and how to rejuvenate. Many models have been introduced in the literature to answer these questions [44, 47, 48 and 49]. Both rejuvenation and restart can be used to fix memory leak errors, but there is a difference between these two procedures .The comparison mentioned in [57] explains the difference between rejuvenation and restart. This comparison shows that rejuvenation reboots the service periodically and only the job currently in service will be dropped. Rejuvenation takes place at fixed intervals in time. On the other hand, the restart process needs to designate a fixed time to perform each service. If client requests are not performed by the requested time, the client will cancel the request and send it again telling it execute all the required services.

Rejuvenation is a cost effective and proactive fault management technique for a software application that is continuously running for a long period of time, thereby decreasing the risk of application failure and reliability and performance degradation. The rejuvenation solution schedules periodic stopping of software and reboots it so as to clean and refresh its internal state [76].

2.2 Major Works on Errors and How to Handle Them

2.2.1 Software Errors

Andrew J. and Brad A. proposed a substitution for ancient techniques to analyze the impact of programming systems on errors. Additionally, the researchers proposed finding the series of events that would lead to failure by defining a framework and methodology to sample these events by monitoring user behavior with the system. The authors built this framework based on three directions of research, namely the classifications of common programming difficulties, studies on the difficulties in the programming process, and research on behavior of human errors. Furthermore, the researchers explained that the occurrence of software errors may lead to software faults and that faults may lead to errors.

According to the researchers, the classification of the common difficulties does not explain the system error types. It in fact explains the strong correlation between errors, faults and failure. However, four important aspects of software error appear through the analysis of this classification. According to authors, the first aspect is a software error's surface qualities. The authors consider that software errors can occur as a result of language syntax. The second aspect refers to the lack of a user's programming experience, as exemplified in data-type inconsistency, design logic bugs, and so on. The third aspect is the programming activity in which a software error can occur. The fourth aspect is the action(s) that lead to errors.

The authors suggested six kinds of actions: creating, reusing, modifying, designing, exploring and understanding. According to the authors, in order to build a strong programming design, both programmers and the programming environment have to be considered because they prove that system errors do not occur only as user mistakes, and that occasionally, the programming environment plays a role in software errors in addition to time interruptions and poor design and construction.

In order to build their framework, the authors considered both programmers and the programming environment based on three aspects, the first of which is the programmer's activity, which involves specification activities, implementation activities and runtime activities.

The second aspect is the programmer's actions, which involves creating, reusing, modifying, designing, exploring and understanding. The third and final aspect is the breakdowns that results from the interactions between the programmer and the programming environment (skill breakdowns, rule breakdowns, and knowledge breakdowns) Afterwards, the authors combine these aspects into two ideas, the first being cognitive breakdowns consisting of the type of breakdown, any suitable action that may be needed to be performed when breakdowns occur, on which interface this action is performed and the information being acted upon. The second idea is the chains of cognitive breakdowns that leads to system errors.

As mentioned earlier, the researchers had defined a methodology based on designing suitable programming tasks, appointing experienced programmers to perform these tasks, recording any chains of cognitive breakdowns by tracking the path of software error to find the main cause of these errors, and finally, analyzing the records to find the relationships in the chains of cognitive breakdowns. According to the authors' results, the study showed that the average chain had 2.3 breakdowns and caused 1.5 software errors. On average, 46% of programmers' time was spent on debugging and implementing code with about 77% of all breakdowns occurring during implementation activity and about 18% of all breakdowns occurring during runtime activity and Boolean expressions represented 33% of all breakdowns.

In general, the authors showed that 18% of breakdowns were debugging breakdowns and 7% were reuse breakdowns, while nearly all of the 24% of the rule and skill breakdowns in modifying code led directly to software errors [54].

Hongfa Xu, et al. showed that a system usually has many components, and these components work well or badly depending on the impact of many external and internal factors. The researchers presented error feedback to find the reasons for the occurrence of errors, when the errors would occur and the source(s) of the errors. The error feedback consisted of two classes: the run-time class used to scan the programming code to detect the error; and the debugging class used to report information and details about the error and suggestions about how to fix those errors.

The authors' showed that each error feedback mechanism should consists of four functions. The first function is the positioning of the error source; each program contains various modules or subsystems. Sometimes, we work with data or functions related to another module or class, in which case it is very difficult to locate the

source of the error. On the other hand, if we work with data related to the same module, the source of the error in this case is easily located. The second function of error feedback is the attainment of errors. This function can be performed by using “catch…. exception” or by setting up a complete error-acquisition system based on an OS or on BIOS. The third function is the expressions of errors. It is important to provide a useful expression and vital information to the person responsible for fixing system errors so as to avoid interrupting or decreasing the entire capability of the system. This information is “date, time, source, type, event ID, user, description of errors”. The last function is the transmission of errors. After finding an error, it must be returned to the programmers for repair. However, two problems are presented through the transmission process, namely missing the error and error distortion.
According to the authors, there were a number of disadvantages with their error-feedback mechanism. Firstly, there was no standard in error expression. At times, errors were presented because of another error, and in such as case, it was very difficult to locate the source of the error. Moreover, attainment and transmission of the error was not self-governing; as a result, there system errors would probably lead to the failure of the mechanism. The authors suggested a solution to avoid these shortfalls by setting up the error-feedback mechanism on a virtual machine to work in a self-governing manner. In this case, system errors would not affect the error-feedback mechanism [4].

Sunil Gupta, et al. defined an overview about the system’s importance in various aspects of life. They also show how it is difficult to account how many project fails or how much money is wasted each year because of such failures. The authors explain that software failure occurs when there are differences between current software behavior and real behavior for which systems are designed. The authors explain the relations between faults, failure and error. The authors explain that many errors may be introduced and if faults occur in a system, then it may lead to failure. Additionally, the authors explain that a system consists of many components, and if failure occurs in one of these components, it may lead to an entire system failure. The authors show that the main reasons for failure may be due to source code, design error or an error occurring in the environment. Moreover, the authors found that a system’s failure range is elevated so long as we increase the number of users and components involved in the system design process. The researchers conducted a survey to find the system failures in most aspects of life. The survey introduced many types of failure in each one of the life fields, including the medical field, aviation, missiles and space craft, the financial sector, the IT field and various other miscellaneous fields. For instance, in medical field, their survey showed a failure in radiation therapy machine software related to the THERAC 25 Company. The failure that occurred in 1995 was due to massive radiation overdoses to patients due to a rare condition. The effect of this failure was the deaths of at least 5 persons. With regard to space craft, the survey showed a failure in the software on NASA’s Mariner spacecraft.

The failure that occurred in 1962 was due to a coding error causing the rocket to wear dangerously (omission in coding). The effect of this failure at that time was $18 million being wasted. In the financial field, the survey showed a failure occurring at the Royal Bank of Scotland (RBS).

The failure that occurred in 2012 was due to a failed update leading to users being unable to make or receive payments; the effect of this failure at that time was that £125 million were wasted. In the IT sector, the survey revealed a failure occurring in Apple’s iOS 8. The failure occurred in 2014 and was due to an error in an iOS 8 update, lost phone signals, a frozen update, and unlocking problems. The effect of this failure was the inconvenience caused to Apple users. In other fields, the survey revealed a failure occurring in new logistics software for a German car maker. The failure that occurred in 2013 was a software change in the firm’s central logistics system. Finally, the researchers concluded their study with some recommendations to avoid failure in upcoming software design. The recommendations included hiring many developers, hiring experienced developers, working with qualified services firms, keeping versions organized and managing tasks, keeping working versions clean, keeping work according to quality standards and involving people in every process [5].

Rivalino M., et al. explained that software aging occurred as a result of software faults and so-called aging-related bugs. These faults would cause accumulated errors and change the behavior of the software in an unexpected manner and ultimately lead the software to failure after crashing. According the authors, memory leaks are the main cause of software aging as they are caused by the faults that occur in memory management and lead to a decrease in the memory space until the system crashes. According to the researchers, it is difficult to remove aging-related bugs during the development or design period in spite of all the programming techniques in existence to improve system reliability. Even in the execution or testing phase, when the system runs for a short period, it is difficult to detect aging bugs, especially those which occur as a result of memory leakage. Aging bugs occur only when the system runs for long time.

The authors attempted to solve this challenge by trying to find the aging bugs caused by memory leakage during the short testing period by using analysis systems. The authors made a comparison between the version of the software that was being tested and an older version of the software that had already passed the test. If there had been a difference between the deviation of the old and the new version, it would mean that the new version has aging bugs. Because memory leakage was one of the main reasons for aging, the researchers would monitor the memory reading periodically.

According to the authors, two important memory keys have to be monitored to decide on the existence of a memory leak: system metrics (aging indicators) and threshold values. The researchers showed the importance of using application specific metrics to find memory leaks; for instance, RSS (resident set size), VSZ (virtual memory size), HSZ (heap size), and HUS (heap usage) metrics that are located in operating systems. The RSS is the working set size of monitored processes. It considers only parts of the process (text, data, stack, and heap) currently loaded in the main memory. VSZ acts as the total amount of virtual memory reserved by a process. HSZ and HUS are, respectively, the total amount and the currently used amount of the process heap, which include both in-memory and on-disk pages. Following the authors' research, they decided to use the HUS metric for their work as it was the less noisy than the others and had a low rate of leak manifestation.

The authors also showed the importance of monitoring the selected metric values to detect the existence of memory leaks by using an analysis model to make comparisons between the heap usage pattern of the target application and the baseline pattern [58].

Jamaiah H., et al. showed that the software aging occur when the software results can't be enough or useful for the user requirement or it can't adapt with the environment, they also showed that software aging occurs in most software relating to most aspects of our lives. The authors showed that the maintenance of software aging becomes difficult year after year and high cost, effort and time are expended especially when the software life cycle decreases from working for 10 years previously to 1 to 3 years only now. According to the authors, software quality does not directly influence the number of years of its operation, and that it is related to the quality of the environment in which the software operates since much software fails to continue operations after a short period of time.

The authors explain the importance of building software in a flexible, modifiable and scalable nature so as to be suitable for rapid technology development and keep it young. Moreover, it must be rejuvenated to delay its aging. The authors show that there are two types of aging, one of which occurs as a result of the changes made to the software and the other occurring when the software fails to adapt to environmental changes. The authors describe the characteristics of software aging as follows: memory bloating/leaks, shared-memory-pool latching, unreleased file locks, accumulation of un-terminated threads, file-space fragmentation, data corruption/round off accrual, thread stacks bloating and overruns.

The researchers found that previous studies have a leak in detection of the external view and perspective and that they are working on finding the perspective and perception in software aging in Malaysia also they had been working to find the factors that influence software aging. The authors prepared a survey in Malaysia, including participants' age, educational degree, firms at which they worked and their positions. The authors' results show that the most influential factors for software change are business and technology requirements in addition to any failure of an existing system currently running. The results also show that aging occurs because of the software not being able to cover business and technology requirements and not being able to adapt to the environment, which leads to a decrease in the software life cycle and thus aging. The author's results also show that the most influential factor of aging is the product profile itself.

The adaptability, stability, technology, training and support, and user satisfaction are the sub-factors related to the product profile of their software. Similarly, the second most influential factor is the functional, which can be categorized with adaptability, stability and rationality. The third factor is the surroundings, which consists of sub-factors such as adaptability, training and support, stability, technology and popularity. Furthermore, the results show that the human aspect has a major aging influence on software applications in certain operating environments. The authors propose that there is the possibility of delaying software aging so long as we have a good understanding of the external factors that influence the software [59].

Jonas T., et al showed that adaptive maintenance is a useful method to improve software quality after releasing the software. However, applying adaptive maintenance many times may reduce software quality and maintainability, which may lead to decay. The authors showed that using perfective and preventive maintenance would be useful to have the software quality and maintainability last a long time. According to researchers, the operator applying the maintenance may encounter many problems. He may be provided with unsuitable resources to improve the software quality. As a result, he would have to study all the negative and positive aspects before making any decisions regarding maintenance.
The authors showed that there was a gap between developer and non-developer staff. Many problems may arise because of this gap, such as managers being unable to approve the addition of new features or fix bugs. Therefore, the authors formulated a decision-supporting technique to fill this gap by using understandable indications (such as current code complexity, past change frequency, and entanglement) within a single view.
This visualization technique combined multiple indicators such as current code complexity, past change frequency, and entanglement used to enhance circular bundle views. It helped to identify the modules of code that may lead to increases in maintenance efforts. By applying this technique, many benefits had arisen for the researchers, including building communication between developers and non-developers. The visual analysis was fast and it enabled human visual analysis to take advantage of one of its strengths, namely identifying outliers using pre-attentive perception. Ad-hoc interconnections of multiple indicators not requiring a defined mathematical operation for interconnection became possible by mapping the indicator’s values to visual variables. The authors encountered some limitation while

processing their approach. There was a difficulty with human recognition such that the modules may have had numeric numbers close to each other but not identical and the operator would possibly have thought that they were the same value. The authors found a number of indicators related to increasing the maintenance risk while increasing maintenance efforts and costs. These indicators that had been designed easily were to be suitable for use and understandable even by non-developers [41].

Gouri P. classified maintenance as being corrective, adaptive, preventive and perfective. The author showed that perfective and adaptive maintenance are used commonly to enhance the system by improving its usability. The author attempted to find the factors that improved the agility of maintenance and to find whether the adaptive or perfective type was more suitable to process with the maintenance of the system. Agility is the authority of the software to choose and react expeditiously and appropriately to various changes in its surroundings and to the demands imposed by the surroundings [98]. Because agility is important in software development, it is important also in software maintenance. According to the author, there are many factors that can impact agility in software maintenance. These factors may facilitate it or hinder it; therefore, the author attempted to analyze these factors. According to the author, organizations have schedules for maintenance. These schedules provide information about when system maintenance is to be performed. These schedules of maintenance may be monthly, semi-annual or annually. The author performed an analysis on twelve systems with different maintenance schedules. According to author’s study and some studies that applied by the some stakeholders, systems with monthly or bi-monthly maintenance schedules were rated by the stakeholders as being faster, more agile and more usable than systems maintained annually or semi-annually.

The author defined maturity of architecture as the capability of a given architecture to serve as the choice of the structural framework for deployment of given software development efforts by virtue of its past usage and performance in multiple efforts of an identical nature. According to the researcher, the maturity of the architecture has an impact because the grown architecture reduces the technical risk linked with the given architecture in advance. Moreover, it gives software users and developers more space to focus on maintenance and development activities. In addition, in organically built systems, software maintenance is executed by ad-hoc slots of components, as and when a request is created. However, maintenance results take more time than those systems that follow an industry standard. Moreover, there are other factors that impact author rating. According to the author, software designed to support a vendor’s products is more agile when compared with systems designed for industry-independent products. Another factor that impacts the rating and the frequency of schedule is the extent of regression test automation. According to the author, systems with automated regression are better than systems with non-automated regression because of the differences in procedure and the time needed to perform it. The authors found that historical rates of change requesting submission also had an impact on ratings. For systems with semi-annual schedules, enhancements were rarely requested [60].

Jamaiah H., et al. showed that software can age similarly to a human and this aging is inevitable and that it is possible to find the factors that lead to software aging. The authors showed that aging occurred because software cannot adapt to an environment and as a result of continuous modifications to the software. The authors showed the possibility of applying rejuvenation to delay aging in software if the software aging factors are clear and understood by the programmer. According to the researchers, software aging occurs when the software works for a long time. However, time is not the only factor that influences software aging and there is another factor that does so, namely software quality. The researchers showed the importance of software quality on its behavior throughout the life cycle, and so they focused on software quality in their work.
The authors took an approach known as Goal Quality Metrics (GQM) to classify the factors influencing software aging. They also used GQM to detect the cause of software aging and examined the aging measurements produced by them. GQM approaches consist of three levels: the first level being the conceptual level which represents the goal of the study; the second level is the operational level, which contains the questions related to the goal; and the third and final level is the measurement level, which contains the metrics used in the study. The authors showed that the best software is that which can maintain its quality for a long time and this takes place using the rejuvenation process.

For all the aforementioned results, it is important to measure software quality at all points in the software life cycle following the certification process in order to provide the ability to predict quality decay. According to the authors, certification is the procedure of giving value to the user or organization and a certificate must be attached with this value to ensure its validity and to prove the internal and external behavior of that software.

The authors showed that software quality should be gauged by non-developers (users, designers, consumers and purchasers), so the quality certification model must be uncomplicated, accurate and usable by any non-expert person. Most software certification methods depend on official validation, professional evaluations, designer valuation and software measurements to ensure the quality of an item. Another method that measures quality is the ISO 9126 model (an old model, currently there is ISO 25000). According to the researchers, software faults can occur as a result of software error (40%), hardware error (15%), human error (40%) and other factors (5%). Software aging occurs due to software errors. The authors showed that in order to apply the rejuvenation process, it is vital to find the aging factors by using the software aging measurement model. These factors can be the base to build a rejuvenation model to delay or prevent aging. The researchers showed that the measures being created were based on the experiences of those who used and designed the software, which means it is a measure of the external features and through this, they can link it with the internal features to enable an honest judgment. According to these measures, aging can be categorized as *little aging* and *big aging*. According to the researchers, many measures should be taken in the account to improve software quality through the rejuvenation process.

These measures include software maintenance, redesigning, repositioning and reorganizing. The measures should be applied at every point in the software life cycle until the new version is introduced to replace the previous version. According to the authors, many rules should be applied regarding software aging, such as software having to be adapted to its environment, continuous growth, increased complexity, organizational stability and the feedback system. The researchers showed that there are four main factors that influence software aging. These include the human factor, the functional factor, the history factor and the environmental factor. The human factor comprises education, staff, users, training and popularity.

The functional factor comprises software usefulness when there is no benefit to using the current software. The history factors comprise the time of designing the software, the time of buying the software and the technology. The environmental factor is the external factor comprising accessories, alternatives and changes of technology [61].

Zaiha N., et al. showed that software aging occurs over time as a result of resource use ever long time as well as being due to damaged data. According to the authors, software must be created and maintained to fulfill user requirements by improving its quality, and when the maintenance process is interrupted, the quality of any software will degrade and aging appears. The authors show that all software will suffer from aging and nobody can prevent it; nevertheless, there is a possibility to extend software life by understanding the causes that lead to this fault. The authors show the importance of measuring software quality to enable software developers to monitor software status throughout the lifetime of the software as any degradation in software quality will lead to the software aging; moreover, software quality can be measured using software metrics. The researchers showed that software aging can occur due to using resources for a long time or by using damaged data, leading to a gradual deterioration in software performance. They also showed the relationship between errors, faults and software failure; they showed that the small errors as well as the big errors can lead to software failure, this failure would contribute to software aging.
According to the researchers, there are two main focuses: the operating system and the end user. Most previous studies had focused on hardware, operating systems and memory problems and poor studies had focused on the end user aspect. The results of the previous studies show that software aging occurs due to software error (40%), hardware (15%), human error (40%) and other factors (5%). According to the authors, factors that affect software aging include its design and implementation, cost, any change in environment and technology, and the quality of the software. The authors performed an empirical study to identify and classify the factors that would affect software aging and the age of software. This study was performed in Malaysia with 111 people who comprised practitioners and users. For software age to be useful, the study showed that 32.4% of respondents gave an answer of 1-3 years, 47.7% gave 4-6 years, 15.3% gave 7-10 years and only 4.5% gave an answer of more

than 15 years. The previous results showed that the software had a short lifespan, so for that the researchers suggested developing a model to extend software lifetime.
Concerning the factors that influence software aging, the study showed 4 factors: the first of which was the functional factor with the highest score (4.48 out of 5), and which related to the software function such that if any interruption occurred in the software function, this factor would have many sub-factors, including the failure of the existing software, the support system, improvement of the software according to customer needs, interface and software performance.

The second factor was the environment factor (4.32 out of 5), which related to the change that would occur in the environment, including the addition of new technology and the software being unable to work with the new technology. Many sub-factors related to this factor include business requirements, technological evolution, changes in the business process, environmental changes, information about software and its popularity. The thirds factor was product profile (4.12 out of 5), which would contain information about the history of the software and its purchase date. The sub-factors that related to this factor included technological evolution, failure of existing software, information about the software, improvement of the software according to customer need, its popularity, policies and documentation and software maintenance. The fourth factor that influenced software aging was the human factor (4.05 out of 5), the sub-factors of which related to business requirements, changes in business processes, support systems, improvement of software according to customer need, orders from the top management, software quality, software maintenance, user satisfaction, education, training and popularity. Depending on all the previous results of the study, the researchers created a model to help developers and practitioners to monitor the quality and status of their software and to help them in extending the lifetime of the software [62].

Carol Smidts created a stochastic model that related the software failure intensity function to finding and discovering the occurrence of development and debugging errors throughout the software lifecycle by using information about the development lifecycle and the developers experience as well as other factors to predict rates of human error. This model allowed for variations in the development lifecycle length and explained the impact of this variation on development faults such as the nature of faults, the number of faults and the amount of functionality developed. According to the author, development errors occur when developers were involved in the software creation stage, while debugging errors occur when developers attempted to remove a known fault from the software; perhaps the target fault was not removed and new faults may have been introduced. The author emphasized the importance of repairs and maintenance throughout the development lifecycle (requirements, design or coding stages). Once a fault is introduced in any of these stages and the more faults are introduced, it means that more repair efforts need to occur. The author showed that when one of the development lifecycle stages (requirements, design and coding) is presented and built, many errors and faults may be introduced and those errors may disappear during the debugging stage. The author conducted a study to find the influence of repairing omission related faults and commission related faults on each of the development stages size. It was revealed that repairing the omission related faults increased the stage size by introducing new parts of the stage, but repairing commission related faults didn’t have any influence on the development stage [63].

Weider D., et al. divided the software development lifecycle into three stages: (1) low level design, which contains details about the algorithms and the components used in the design process; (2) implementation, which contains the source code and data needed in software testing; and (3) the unit test, in which each component and code scope used in the design stage is tested individually. The authors showed that 50% to 60% of software product errors occurred during the development phase. The authors also showed that it is important for engineers to have information about fault types, their frequencies and percentages and when they initially occur in order to succeed in software development and production [64].

Kanav K. and Foad S. showed that a high cognitive load environment caused challenging problems for human-computer interaction designers as these environments needed advanced decision making under time constraints in the presence of noise. The authors showed that a large number of user errors had been reported each year and they showed the great impact of user mistakes on these systems. According to the authors, in order to monitor user behavior, it is important to monitor the environment in which he works. The authors suggested two methods

to monitor user behavior and predict errors. The first method is to monitor the behavior of each user individually and create a model to predict errors that may be produced by the user.
The second method is the physiological method, which monitors the user by tracking some factors, including user experience and fatigue. The authors created a multistage analysis algorithm framework to classify the errors and provide feedback to users so as to alert them to take measures to avoid errors. According to the authors, errors can be categorized into two types: procedural errors, which occur through action execution, and proficiency errors, which occur due to lack of user ability and experience. The author's framework analyzed user errors as slips and mistakes. The authors provided a database for their framework; this database contained 80 samples of each status (no error, slip, and mistake), 60 samples for training and 20 for testing. Moreover, they used a time interval to check the time. The analysis revealed that most of the results fell between slips and mistakes, and the results ensured that slip errors were not a result of missing experience but occurring due to missing perceptual cues [51].

Benjamin W., et al. showed that in user interaction system UIs, the user has complex duties and when monitoring complex processes and choosing suitable functions for them, human errors have been committed and these systems become erroneous. The researchers showed that human error in such systems occurred due to lack of knowledge and experience, training, situational awareness, mental workload and interface design. The authors showed that users may create an abnormal situation inside the system which would recognize this abnormal situation and attempt to repair it by offering many reconfiguration options to users to help them in recognizing and avoiding repetitions such errors. Moreover, they showed that it is essential to develop user skills, knowledge and situational awareness in order to avoid and prevent these errors. According to the authors, situational awareness refers to the perception and comprehension of information and the projection of future system states and the important factors that may increase or decrease human error in user interface systems. The authors also showed that interface design and how the information was distributed also influenced the situational awareness value, and it showed how information can be accurate, how it is compatible with a user's situational awareness requirements and how it helps to reduce mental workload. The authors showed that interface reconfiguration and flexibility help to reduce human error and increase performance as they can reconfigure it according to their requirements [65].

Sumie Y., et al. showed that human errors occur due to many reasons, including lack of skills required to perform a task, lack of knowledge about how, why and when to perform a task, limitation in a user's abilities, mistakes because of wrong action and wrong planning, slip errors because of wrong operations, lapse errors occurring when the user forgets how to perform a task, and violation errors because of non-compliance with specific standards and rules. In order to reduce and prevent human error, the researchers created a model to adopt them, which they called an analytic hierarchy process model. The authors conducted a study on human error when using an AV remote controller and they showed errors occurring due to users making mistakes and using wrong buttons, and users pressing the right button but the button itself performing the wrong action or an error coming through the button size. The authors showed that errors occurred because of the age of the operator such as elderly persons having vision problems but children not having those [66].

Fuqun H. and Bin L. showed that the developer's cognitive abilities are important in software development and that cognitive errors are very common representing 87% of system residual failures. Therefore, there is a need to develop users' cognitive abilities in order to prevent cognitive failure because if developers can understand why, when and how error occurs, they can prevent errors more effectively. The authors emphasized the factors that influence human error in order to predict the human error probability. The researchers showed that human error cannot be prevented using external devices and that it can be prevented only through the developers themselves by studying their own cognitive processes.
The researchers created defect prevention based on a human error theories (DPEHE) framework to explain the cognitive mechanism. This framework relied on the knowledge and monitoring of developers. The framework consisted of three stages: the knowledge training stage with 2 aspects, the first being the cognitive model used to govern human error and is the second being developers having to know why users

produce errors, what type of error they produce and how to prevent these errors during programming. The second state is the regulation training stage which helps developers to promote the awareness and self-regulation of human error prevention. Developers are supported by two DPEHE checklists. Developers monitor the development process and when they identify an error during the debugging and testing phase, they attempt to find the cause of this error by using the regulation lists. In the third and final stage, developers employee the experiences and self-regulation acquired during the regulation stage to continue to improve the systems and prevent defects.

According to the authors, software developers have to understand the cognitive process in software development in order to understand the error mechanism. They showed that software development performance has two activities, namely routine and design. Routine activities do not need any effort for problem solving, which represents a small part of the development process. The main part of the process is the coding and design activities, with coding representing routine problem solving and design representing non-routine problem solving.

The researchers showed that errors would occur due to two reasons, the first being due to limitations in working memory leading to error in the attention mode, and the other reason being due to a lack of human knowledge, which leads to errors in the schematic mode. Errors in the attention mode may be attributable to the user possibly allocating attention to the wrong features or ignoring some information. Alternatively, the user may have omitted proper checks or he may have performed mistimed checks. When there is a working overload, the cognitive load can exceed the user's working memory capacity. Therefore, working overload error can be due to task complexity, workspace limitation.

The researchers created an error base mode which was created based on the most common errors encountered, validated and accepted by software developers in the past. They would avoid errors that would rarely occur. Moreover, they created a scenario for each error mode with explanations and examples to help software developers to understand these error modes. Errors such as classical modes are summarized by reason, post completion errors, lack of knowledge, and problem representation errors. Furthermore, the researchers adapted, tailored, and reorganized these modes to strike a balance between usability, consistency and completeness. The researchers provided a heuristic strategy based on the error base mode and cognitive mechanism of problem solving. This was an effective strategy used to enhance self-regulation and provide a strong ability to help developers to choose the correct approach to solving and preventing errors.

The authors provided three stages of problem solving: problem representation, solution generation and solution evaluation. They also provided strategies for each stage and they explained when, why and how to use each strategy. The researchers showed that the error mechanism and prevention strategy is insufficient for use by developers in real situations, but they needed to have knowledge about the situation and this knowledge came through training, whether or not it is prone to error. Therefore, the researchers designed two regulation checklists used in developer training to increase and improve the ability to prevent errors. These checklists include a problem solving regulation checklist (PSRC), which is used prior to and in the course of software development. The other checklist, the root cause identification checklist (RCIC), is used after defects are found in the debugging or testing phase.

The authors performed two studies, the first of which was in an international organization at CMM Level 5, at a software development institution in the Chinese Aviation Industry at CMM Level 1. The authors used both quantitative and qualitative data in their study. The quantitative data were collected through the participants' assessment and the qualitative data were collected through open questions to developers about their ability to prevent errors using DPEHE. The researchers' study showed that DPEHE would promote a programmer's software defect prevention capability by providing links between error modes, error-prone scenarios, error prevention strategies and programming defect examples [67].

2.2.2 Software Maintenance

Jane H., et al. defined adaptive maintenance as the modifications made to software systems to improve performance and adapt it to environmental changes. The authors explained that managers and developers misjudged the time and effort needed to maintain software. According to the authors, many problems would occur during a system change process, including a lack of validated and adopted tools for planning, estimating and performing maintenance. Therefore, the authors attempted to define the metrics relating to adaptive maintenance by using a metrics based model to estimate the effort required to perform an adaptive maintenance process depending on personal hours. The authors built this model based on determining the metrics that correlated with maintenance efforts. According to the researchers and studies shown in [55], the most important metrics that affect the efforts of adaptive maintenance are the percentage of operators changing and the number of lines of code that are edited, added, changed or deleted [56].

Philipp R. and Katinka W. explain the impact of software aging and its rejuvenation on network software. The authors define software aging as the degradation of software performance until the software crashes. The authors showed that the rejuvenation process is an evident process that solves aging problems by periodically restarting the software. The authors define rejuvenation as the process that prevents operation drop time and avoids system crashes caused by software aging since rejuvenation reduces task execution times and avoids task failures. The researchers attempted to explain the usability of using rejuvenation and restarting to avoid aging problems.

The authors assumed a work area that contained clients and service providers. Many clients were able to send service requests to services provider and the service provider was able to provide service on request at the time by using a queuing process (first-come first-served). The authors assumed two types of model to study aging behavior, the first of which being the Weibull-distributed model, which follows crashes that occur in the intervals of the Weibull-distributed length. The second model is the explicit model, which follows the aging that occurs due to memory leaks. According to the authors, when a crash occurs in the server, the server will lose every task in its queue. According to the authors, both rejuvenation and restart will improve the performance and maintain system availability. Rejuvenation saves on execution time and reduces loads but the server will lose the job in a crash. A restart will take more time to complete any tasks and it may cause further load on the server because the client will continue in its requests. However, it will ensure that every service request is performed.

After all this discussion, the authors found that restarts can be useful if aging does not affect server performance; however, with explicit aging and when the load is high, it is not possible perform a restart. They instead use rejuvenation as in this case, a restart would increase the load on the server [57]. This result was also improved by **Rivalino M., et al.,** as they defined software rejuvenation as the best method to decrease aging effects by resetting the application whenever aging effects occur [41].

2.2.3. Human Errors

Zahra A., et al. showed that the major factors contributing to change requirements in software engineering are human errors and faults. The authors showed that not all software projects are free of the requirements for change. Most projects would need additions, modifications and deletions. These processes would increase the risk of a project and as long as humans perform these processes, the human factor would be the most important aspect in software projects. The authors showed that human error included failure to set an objective, substitution of a word or alphabet, omitting words or sounds, gaps in attention and memory failure, omitting a particular activity, and using or disregarding a particular activity [69].

J. Ernstsen and S. Nazir showed that errors would occur in most socio-technical frameworks (systems that need interaction between humans and machines). The researchers showed that 80 to 85 percent of these errors were due to humans. Therefore, many resources and efforts have been spent on improving human performance and the reduction of human errors. The authors showed that the role of people in such systems is critical because it may cause damage to equipment, touch people’s lives, cause severe injuries, or contribute to environmental pollution.

The authors also showed that the assessment of human reliability is a vital approach to predict human error, and the various studies of human error yielded a variety of human reliability assessment methods (HRA), most of which can be divided into quantitative or qualitative approaches. In order to understand and predict human error, the HRA function finds the errors that occur through an operation, finds its probability of occurring and finds methods to reduce such errors. The authors showed that the HRA of socio-technical frameworks is complex because there are many interdependent and dynamic variables influencing human reliability.

The authors attempted to reduce the high percentage of human error occurrence by building a systematic human error reduction and prediction approach (SHERPA) to study error types and error remedies apparent in pilotage systems (one of the socio-technical frameworks). The authors conducted the study in Norway, where they collected data through interviews and observation. They focused on the errors most likely to occur, errors deemed as critical and the high frequency of the occurrence of the same error type. With SHERPA, the authors divided the errors into five categories, namely action errors, checking errors, retrieval errors, communication errors and selection errors. The authors' study shows that action errors are the most frequent errors and second most frequent are communication errors [74].

Steven and Barry conducted a study on human error in air traffic control and showed that human error involved perception, memory, decision-making, communication and team resource management (TRM). The author's literature showed that human error may be skills, rules and knowledge-based behavior. Additionally, it can be classified as slips, violations, mistakes and lapses [75].

Etman and Halawa conducted a study on the safety design in maritime incidents. The authors showed that the focus on the human factors and errors was more important than the ship's design and operation. The International Maritime Organization (IMO) focused its attention on the human role in maritime accidents in the mid-1980s.

The authors showed that the main focus of the IMO was the enhancement of the design and operation but there was a lack of attention given to human factors such as the lack of proper competency, multinational crews, education and training systems and many others. The authors showed that human role is vital in the maritime sector and that people working in this sector must be well trained and motivated.

According to the authors, there are many factors influencing the role of people in the maritime sector, including competence, as the authors showed that additional to the human skills and training, they have to have good assimilation and understand the subjects. The human attitude towards education and training would be given by their mental ability, intelligence, personality, character and sensitivity. Self-awareness and self-evaluation were the key drivers. Motivation can be driven by the teamwork, good communication, direction and empowerment. In order to help people to perform their duties and jobs, they have to have a good life style, attend alcohol and drugs tests, and be encouraged periodically to exercises and watch their diet. A safe and secure work environment includes a safe work area, the use of full safety equipment, and the use of safe and proper tools.

According to the authors, in order to perform in the maritime safety culture, maritime administrations should take care in people's training, education, work environment, working and rest hours, and living conditions onboard. According to the authors' literature and the study performed by the IMO, there were many factors that would lead to accidents in the maritime sector, with the human factor being the main cause of 80-100% of these accidents. The authors show that the common human causes of accidents are stress, lack of training, lack of knowledge and education, lack of motivation, lack of communication, carelessness, and operator errors [77].

Paul C. & Ann B. show that user errors commonly occur through the use of interactive systems and these errors are disastrous as device errors. The researchers showed that people behave rationally, as they draw on their goals and knowledge and attempt to perform tasks, and as a result, identical types of persistent human error occur. The authors defined a user model to detect these persistent user errors.

The model detects three classes of persistent errors, the first of which is post completion errors, which occur when a user terminates a current remaining outstanding completion task. The second class is device delay errors, which occur when there is a device delay in performing a task without feedback for the user and the user attempts to repeat the last action again. The third class is communication-goal errors [78].

John G. and Sandra L. showed that human error can be identified, controlled and minimized if we know the potential source of them.

The authors established an approach to identify the critical procedures in preventing major accidents and human error related to these procedures and how it can be controlled to minimize the errors. Depending on a report from the Exxon Mobil Company, they made an assessment of all the operational procedure steps to find potential and relative errors.

When the assessment found the relative errors, they continued to find the cause of the errors, the consequences of those errors and the existing control and recovery actions. Table (1) below contains examples of error categories and specific error types used in the classification of potential human errors.

This is a comprehensive list based on the Health and Safety Executive Guidance Document on Human Failures provided on the Exxon Mobil website [79].

Table 1 Examples of error categories and specific error types used to classify potential human errors [79]

Error Type	Error
Action Errors	❖ Operation too long/short ❖ Operation mistimed ❖ Operation in wrong direction ❖ Operation too little/too much ❖ Operation too fast/too slow ❖ Misalignment ❖ Right operation on wrong object ❖ Wrong operation on right object ❖ Operation omitted ❖ Operation incomplete ❖ Operation too early/too late
Checking Errors	❖ Check omitted ❖ Check incomplete ❖ Right check on wrong object ❖ Wrong check on right object ❖ Check too early/too late
Information Retrieval Errors	❖ Information not obtained ❖ Wrong information obtained ❖ Information retrieval incomplete ❖ Information incorrectly interpreted
Information Communication Error	❖ Information not communicated ❖ Wrong information communicated ❖ Information communication incomplete ❖ Information communication unclear
Selection Errors	❖ Selection omitted ❖ Wrong selection made
Planning Errors	❖ Plan omitted ❖ Plan incorrect
Violations	❖ Deliberate routine ❖ Deliberate exceptional

Barry K. defines major error types including slips and lapses, which include wrong sequences to perform tasks, the amount of performance (too much, too little), errors related to maintenance, cognitive errors related to misunderstanding of a system's design or procedure, and the lack of training and skills. Maintenance errors and latent failures include maintenance and testing errors that lead to immediate failure. Errors of commission occur when the operator performs an incorrect and unrequired task. Additional error types include rule violations, idiosyncratic errors and software programming errors [80].

Vicki A. and Donald G. showed that human error can lead to economic loss as a result of equipment damage or system outages. They also showed that human error occurred in an airline company in 1998 which had caused an outage for two hours. This outage at that time led to 265 delays. According to the researchers, there are three major sources of human error: communication and coordination errors, procedural errors, and errors resulting from installing new software or equipment.

The authors conducted a study on the airlines' facilities companies with a number of maintenance control center (MCC) specialists. The researchers found that human error in the MCC would occur due to communication and coordination errors that were caused by miscommunication between two teams or between two persons, such as when changing from an engine generator to a commercial generator without informing the radar center.

Moreover, errors were due to incomplete or incorrect information, caused by using an out-of-date database, such as appointing a task for a technician who was not available or on vacation. The work shift and workload errors also occasionally contributed to problems, such as a work shift continuing a long time or assigning many tasks to be perform in a short time.

The researchers also conducted a study with a number of operations control center (OCC) specialists, and discovered that human error in OCC was related to procedural errors. At times a specialist had not been following a procedure or the procedure was not clear. There were also remote maintenance monitoring errors such that a number of the specialists had not been familiar with the remote maintenance monitoring. Usability errors showed that the usability of the interface design needed to be examined and that operators should have been trained to use it. Finally, insufficient training/insufficient experience errors occurred because most of the experienced specialists were retired and the new specialists were lacking in training and experience [81].

Ender A. shows that the major cause of maritime accidents is human error and those errors are caused by the environment, organization and technology. Therefore, ergonomic design has to exist in systems that have interactions with humans in a specific work environment to support the abilities and limitations of the human user. The researcher showed that the human factor in maritime accidents has been growing. In 1960, the human factor comprised 30% of an accident, but in the present time, it is now 70-90% of accidents. The researcher shows that the poor training and education of a ship's employees is one of the main causes of most accidents that occurred in the present days [82].

Neville A., et al. built a human error template (HET) that used an External Error Mode (EEM) to predict pilot error. The HET and EEM comprised twelve types of errors, namely fail to execute, task execution not complete, task executed in wrong direction, task repeated, wrong task execution, task executed on the wrong interface element, task executed too early, task executed too late, task executed too much, task executed too little, misread information [83].

James J., et al. divided human error into: (1) unintentional errors, which occur when the user lacks experience with the task he performs; and (2) intentional errors, which occur when the user believes that his action or idea is better than the prescribe action. The researchers suggested that managers should determine whether a user has the required skills and attitude to perform an assigned task because they study the Performance Shaping factors (PSFs) that influence human performance. According to the authors, there were three PSFs that can affect human performance, namely internal PSFs, external PSFs, and stressors. Internal PSFs are the user's abilities, skills and training as shown in the table (2):

Table 2 Internal Performance Shaping Factors [84]

Internal Performance Shaping Factors		
❖ Training/skill ❖ Practice/experience ❖ Knowledge of required performance standards ❖ Stress: mental or bodily tension	❖ Intelligence ❖ Motivation/work attitude ❖ Personality ❖ Emotional state ❖ Gender	❖ Physical condition/health ❖ Influences of family and others ❖ outside persons or agencies ❖ Group identifications ❖ Culture

External PSFs are divided into two characteristic groups: situational characteristics and task, equipment and procedural characteristics, as shown in the table (3).

Table 3 Two groups of characteristics of External Performance Shaping Factors [84]

Situational Characteristics	Task, Equipment and Procedural Characteristics
❖ Architectural features ❖ Environment: temperature, humidity, air quality, lighting, noise, vibration or general cleanliness ❖ Work hours/work breaks ❖ Shift rotation ❖ Availability/adequacy of special equipment, tools or supplies ❖ Staffing levels ❖ Organizational structure: authority, responsibility or communication channels ❖ Actions by supervisors, co-workers or accreditation and regulatory personnel ❖ Facility policies	❖ Procedures: written or unwritten ❖ Written or oral communications ❖ Cautions and warnings ❖ Work methods/practices ❖ Dynamic vs. step-by-step activities ❖ Team structure and communication ❖ Perceptual requirements ❖ Physical requirements: speed and strength ❖ Anticipatory requirements ❖ Interpretation/decision making ❖ Complexity: information load ❖ Long- and short-term memory load ❖ Calculation requirements ❖ Feedback: knowledge of results ❖ Hardware interface factors: design of control equipment, test equipment, process equipment, job aids or tools ❖ Control-display relationships ❖ Task criticality ❖ Frequency/repetitiveness

Stressors are divided into psychological and physiological stressors, as shown in table (4):

Table 4 showing psychological and physiological stressors [84]

Psychological Stressors	Physiological Stressors
❖ High task speed and heavy task load	❖ Fatigue
❖ Suddenness of onset	❖ Long duration of stress
❖ High jeopardy risk	❖ Pain or discomfort
❖ Threats of failure or loss of job	❖ Hunger or thirst
❖ Monotonous or meaningless work	❖ Temperature extremes
❖ Long, uneventful vigilance periods	❖ Radiation
❖ Conflicting motives about job performance	❖ Exposure to diseases
	❖ Vibration
❖ Negative reinforcement	❖ Movement constriction
❖ Sensory deprivation	❖ Movement repetition
❖ Distractions: noise, glare or movement	❖ Lack of physical exercise
❖ Inconsistent cueing	❖ Disruption of circadian rhythms
❖ Lack of rewards, recognition or benefits	

The authors showed that in order to improve human performance, managers have to address two basic error types, namely errors caused by human characteristics that are not related to the work situation (accounting for 15-20% of errors), and errors caused by factors related to the design of work situations (accounting for 80-85% of errors). The authors showed that if the PSF of the work situation is not compatible with human abilities, attitudes and limitations, errors will arise because of insufficient procedures, insufficient equipment design, insufficient training, poor communication between workers, incompatible human interests and insufficiently labeled equipment [84].

Cannon A. B., et al. Conducted a study on user errors on mobile devices. The authors showed that the age of the user and input content have a major impact on error rates. Moreover, they showed that there were two important metrics for evaluation of the input content, namely accuracy and speed. According to the authors' literature, there are three groups of typing error: substation, insertion and omission. The researchers presented two methods used for typing into mobile devices: keypad devices and touch screens. They showed that users made fewer errors when using keypad devices. According to the researchers’ literature, keypad devices were faster than touch screens as they found that typing time when using a touch screen was 73% longer than a keyboard. However, the authors found that the age of the user has an impact on this value. They found that the older user works faster when using a touch screen, but those users made fewer errors when using keypad devices. The authors attributed those errors to the lack of knowledge and experience in using touch screens. The researchers conducted a study on participants from two different generations, and the results of their study differed from their literature. The study showed that users work faster and make fewer errors when using touch screens and it showed that age has no impact on user error [85].

Gheorghe and Cecelia studied human error in the power sector and showed that human error was the main key in operational unavailability, equipment damage and accidents. They showed that the training and experience of operators had a major impact on the availability of the systems, as inexperienced operators caused a greater number of errors when performing complex tasks. The authors defined human error as any user behavior or factor that can cause negative results of a task. The authors showed that there was a strong relation between the complexity of a task, human experience and human error, as inexperienced operators would produce a greater number of errors when the complexity of a task was high, as shown in the figure (5).

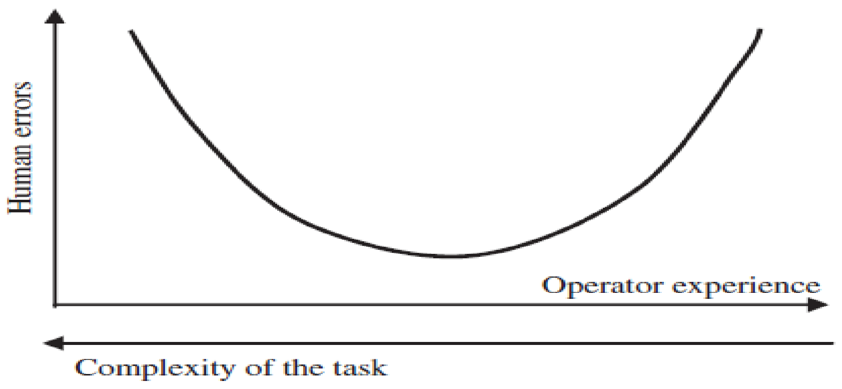


Figure 5 Relationship between human error, operator experience and complexity of a task [86]

The researchers showed that errors would usually occur because of system design, environment and human factors. They also showed that system design could be controlled to reduce human error while the environment and the manner in which an operator would use the system was very difficult to control. Moreover, they showed that human factors have a major impact on the availability and security of systems as the user has direct use of the system infrastructure. There are many factors that have an impact on the incidence of human error, as shown in the table (5):

Table 5 shows the factors that have an impact on the occurrences of human error[86]

No.	Factors	Description
1	Competency	Knowledge of the job, skills and attitude towards the job
2	Communication	The ability to express information
3	Procedural Factors	Clarity regarding standards and procedures and whether they are adhered to
4	Mental and Physical Factors	Stress, cognitive overload and exhaustion
5	Socio-Environmental Factors	Personal pressures such as family pressures and organizational pressures such as work relations
6	Motivation	Individual and organizational aspects such as job satisfaction and leadership style
7	Ergonomic Factors	Light, noise, space, etc. This includes health, safety and shift cycles

The authors divided human error into two types: error of omission, which means a user executes a task but missing some of the procedure steps, and error of commission, which means that the user performs a task following a different procedure. The researchers showed that human error can occur due to a lack of procedure, a lack of personal training and poor environments as well as occurring due to the design of the system interface, as misunderstanding the interface may increase occurrences of errors [86].

Pierre and Paula studied human error in mining and manufacturing. They showed that 84% to 94% of errors was due to human error. The authors showed that the working shift and the time had a strong impact on the occurrence of human error, and it was observed that errors increased during the night, especially from 01:00 am to 08:00 am [87].

Anne I., et al. studied human error in aviation systems and showed that more than 90% of errors were human in origin. The authors showed the importance of periodic training of the personnel as every system is being continually upgraded and a lack of training would increase the number of errors [88].

Tianyi C., et al. studied the typing and pointing errors in mobile web. The authors showed that the design of a system interface had a strong impact on increasing or decreasing errors and they showed that users usually produced more errors when using the web on their mobile devices. Moreover, their work was inefficient when compared with desktop or laptop computers.

They also showed that user experience had an impact on error occurrences as experienced users produced fewer errors when using the mobile web. The researchers' study revealed many types of typing errors, including key ambiguity errors, which would occur when typing a letter different from the target letter, missing key error, which would occur when the user would press the targeted key but without applying sufficient force to select it, additional key errors, which would occur when the keys are very near to each other so when the user selects the target key, another key might also be selected, bounce errors, which would occur when the user presses the targeted key more than one time, long key press errors, which would occur when a key is pressed for too long a time and causing unwanted copies of the targeted letter, and finally, transposition errors which This error occurs when two characters adjacent to each other are typed in reverse order [89].

Phillip P., et al. studied human error in the transportation sector and defined human error as the inability of the user to achieve the required target through predefined conditions or achieving a task different from the targeted task. The authors showed that to prevent or to decrease human error, many barriers should be created such as procedure, experience or training barriers, as those barriers would be used to prevent any undesirable events [90].

Won C.C. and Tae studied the effects of human error on electrical components and showed that human error had been the main cause of the errors that impact electrical components. The authors classified human error in three groups: (1) Errors of commission, accounting for 97% of errors; (2) Mistake/slip/lapse/violation errors, accounting for 74%; and (3) Latent errors, which account for 95%. The authors showed that in order to prevent an error, human behavior had to be studied. They described many casual factors that influenced human behavior, as shown in the figure (6) [91].

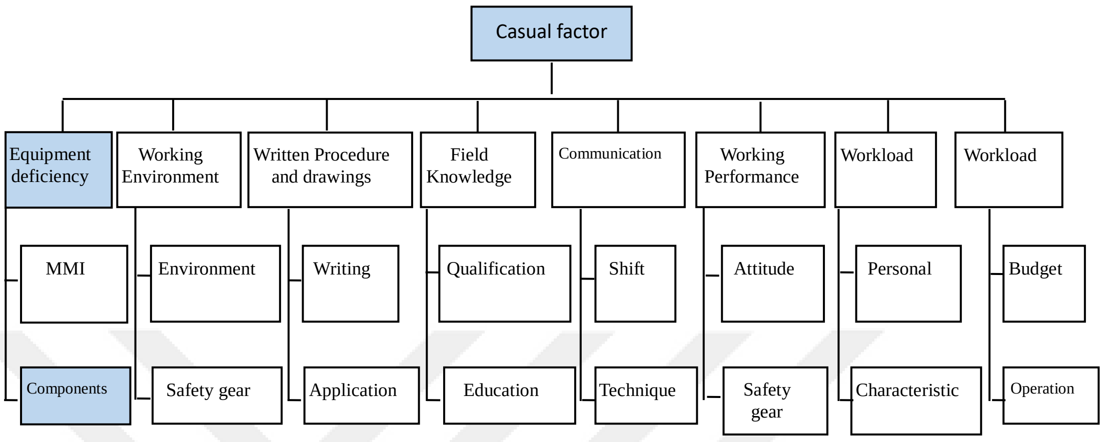


Figure 6 Casual factors that influence human behavior [91]

Shi W., et al. showed that human error is the most major cause of error and failure in production systems. The authors showed that many factors would influence human behavior and lead them to make mistakes. Some of these factors are physiological, psychological, training, experience and environmental, as shown in the figure (7) [92]:

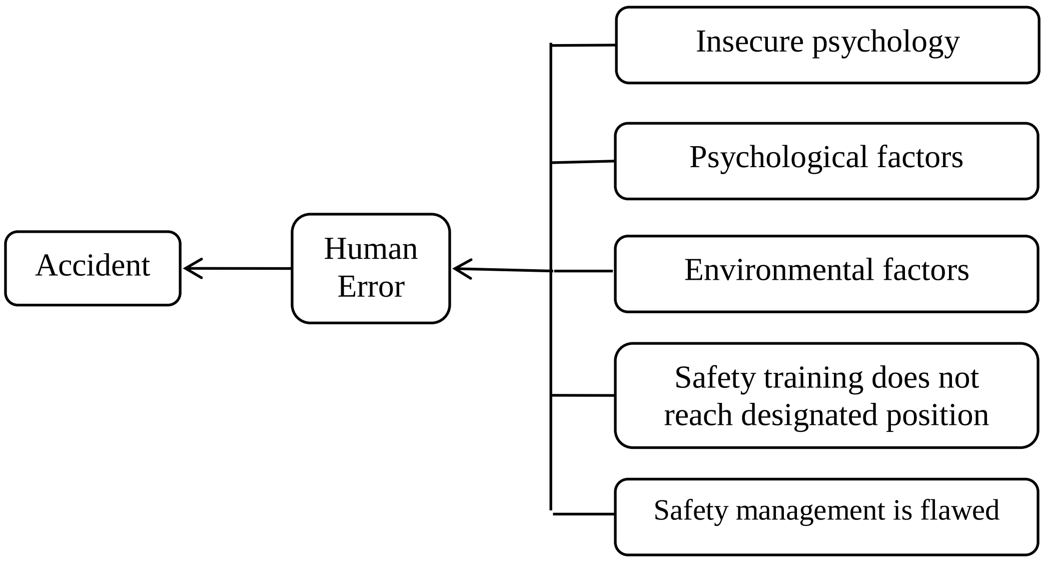


Figure 7 Model of human error [92]

Dahlias S., et al. studied road accidents and showed that human error accounted for over 90% of road accidents. They also showed that people's lack of knowledge and experience is the most important reason for accidents in addition to non-periodic training as users may face difficulties remembering the steps and sequences that lead them to perform the task. The authors divided human error into three groups: perceptual errors, which occur as a result of long working shifts and the time of the day; distraction errors, occurring when the user's mind is concentrating in something else; and response errors, occurring when the user does not make the correct or most appropriate response or there is a delay in responding to an emergency situation. Finally, the authors showed that human thinking and reaction times are the most important factor influencing task reliability [93].

Marie B. studied human error in dynamic environments. She defined human error as the entire task performed a person and exceeding the limit of acceptance. She also defined human error as every activity performed by a person not leading to the achievement of target goals. The author divided errors in terms of human cognitive activity levels, namely skill-based level, rule-based level and knowledge-based level. The author showed that many temporal errors may occur and her literature revealed five groups of temporal errors: incorrect estimate of sequence of actions, incorrect estimate of duration, failure in the evaluation of the right time to act, failure in the anticipation of an event, and failure in the synchronization of collective actions [94].

Arnstein F. categorized errors as active errors and latent errors. Active errors are those that occur directly before an accident. The author divided active errors in terms of human cognitive activity level, the first of which is the skill based level, including slips and lapses. The second active error in terms of human cognitive activity level is the rule based level, such that commitment to following rules and procedures is important to reduce human error. The third error is the knowledge based level; the author showed the importance of the training and experience of the person to reduce human error. The author categorized active errors technically as capture errors, description errors, memory errors, sequence errors and mode errors. Latent errors would occur due to a lack of training, lack in supervision, following poor or incorrect procedures as well as to social factors such as human language knowledge and physiological states [95].

2.3 Discussion

In this chapter, we reviewed works reported in the literature on software failures and maintenance, mainly relating to oil and gas industry. These studies can be clustered as follows:

- **Education**

  Educational level is one of the human factors that influences software efficiency and have an impact on software life cycles [61] [62]. People with high educational levels have the ability to improve the culture of safety [77] since lower educational levels had been one of the main causes of most accidents in the 19th century [82].

- **Experience**

  People without sufficient experience in their field cannot imagine the size of a problem that they encounter and they are not able to find suitable solutions for their problems [7]. Experience level is the main factor for mistakes, as persons with little experience cannot recognize the correct course of action to perform [51] [53]. As a result, they cause more errors when performing complex tasks [86]. Moreover, the level of experience can help to predict the error rate [63] as it is the main cause of human error [65].

  Therefore, in order to decrease human error, the level of experience must be one of the barriers to be overcome [89]. One of the recommendations that can be given to firms would be to hire experienced people [5]. Error rates will increase when the experienced specialists are retired and junior specialists come to replace them [81]. These newer specialists do not have sufficient experience to perform the tasks assigned to them and thus they will cause unintentional errors [84].

- **Language**

  A misunderstanding of the rules or procedures may lead to rule-based errors [69, 72, and 73]. These errors may occur if a specialist cannot understand procedures well or if procedures are not clear [81].

- **Work Load and Training**

  The assessment of human reliability and ability is a vital factor to predict human error [74]. The length of a working shift has an impact on error occurrence, as long shifts mean assigning too many tasks for a person and this may lead to human error [81]. Long working shifts can lead to perceptual errors [93]. Moreover, errors increase during night shifts, especially between 1 am and 8 am [87].

- **System Interface**

  The use of touch screens, keyboards and touch screens with pens has a major impact on error frequency as users produce fewer errors when using keyboards [85]. This interface design has an impact on human error as much human error occurs due to interface design. Interface flexibility can help to reduce human error and increase performance [65]. Furthermore, misunderstandings of the interface may increase error occurrence [86]. The usability of an interface should be examined because low usability may lead to human error [81].

- **Extra Activities**

  Copy-paste is usually used by the people to reduce writing efforts in repeated text. Sometimes, however, wrong or duplicated data are copied, which leads to errors [68] [100]. Unrequired activities lead to errors of commission [80]. Operators are usually involved in unrequired tasks and perform tasks outside their scope of work which thus leads to error [101].

CHAPTER THREE

OIL AND GAS WELL SITE AND WORK FORCE

3.1 Introduction

As mentioned earlier, drilling an oil well passes through many stages and services; one of those services is the well construction service.

Well construction services, including casing and tubular running services, using various sizes and types of tubing and casing joints to create the well string. The casing or tubing is a joint that is usually about 40 feet long and screwed together to form longer lengths of casing, called casing strings or tubing strings. Each joint has a collar or coupling slightly larger in diameter than the joints and it also has female threads used to connect the two male joint ends, as shown in the figure (8).

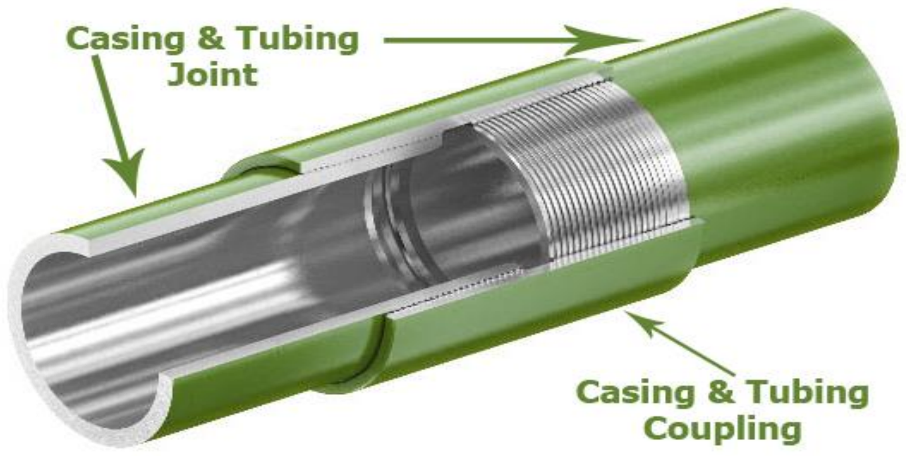


Figure 8 Casing and tubing joints with coupling

Casing service starts in the initial stages of the drilling by running various sizes of casing pipes depending on the well depth and according to the casing program. Here, a well is drilled to a certain depth, cased and cemented, and then the well is drilled to a greater depth, cased and cemented again, and so on. Each time the well is cased, a smaller diameter casing is used. The casing aids the drilling process in several ways.

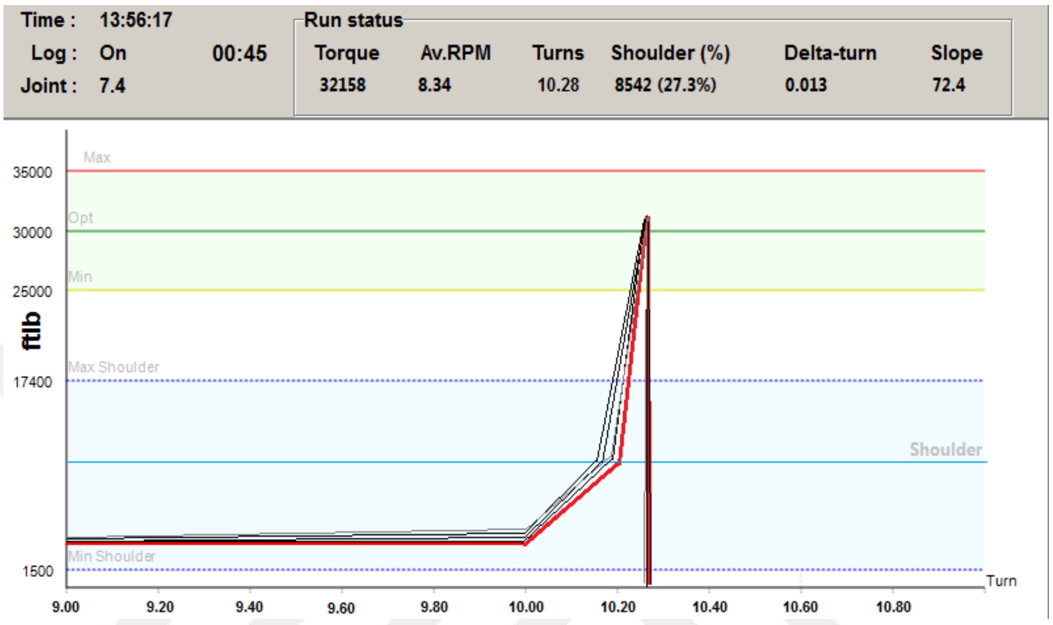
It prevents fluid loss into or contamination of the production zones and provides a strong upper foundation to allow for the use of high-density drilling fluid to continue drilling deeper.
Tubing services are used to run the production pipe line and the fluid’s route to the surface through the tubing string to the export or storage devices. This tubing runs down into the well within the casing. Tubing joints are threaded together into a long string, which is then perforated near the bottom to allow fluid from the formation to flow into the tubing.
In order to perform casing and tubing running services, various types of resource are used, including hydraulic equipment (HPU, HPT and CRT), rig mechanization tools, handling gear, experienced human, additional to use one of the oil and gas critical infrastructure systems called Torque Turn system (TTS).
The TTS Technician uses TTS with a power tong or CRT to apply the required torque to the thread. The torque needed to make up the casing thread is monitored and controlled in order to ensure that the casing thread does not leak. The make-up torque ratings rely on casing size, grade, metallurgy, weight, and the thread compound friction factor being utilized.

3.2 Human Role in Torque Turn Systems

The human role in this field can be:

- Blue collar employees to handle jobs manually (in manufacturing, construction, maintenance, etc.) such as technicians and engineers who perform the work at the sites.
- White collar employees to handle office operations such as customer services, HR, sales, etc.
- Crew chiefs and supervisors to handle maintenance and monitor work as well as be in touch with the customers.
- Managers to oversee the departments and ensure that work in performed according to customers’ needs.

TTS is a real time computerized device with data acquisition systems embedded in it. TTS is usually connected with external hydraulic equipment such as HPT, CRT via the BGDU. This external equipment is used to make up and break down the casing and tubing joints. The TTS receives the torque and turn values that come from the external device through the BGDU and calculates the values according to a special formula to show the running results as a graph on a screen, as shown in the figure (9).



Time : 13:56:17
Log : On 00:45
Joint : 7.4

Run status

Torque	Av.RPM	Turns	Shoulder (%)	Delta-turn	Slope
32158	8.34	10.28	8542 (27.3%)	0.013	72.4

Figure 9 Running result as shown on TTS screen, Figure shown casing or tubing running graph

As shown below in the use case diagram, four actors are involved in these operations:

Torque Turn Technician:

This person’s responsibilities and tasks include working with the TT Tech Supervisor to validate requirements and verify jobs and equipment, operate and maintain other machine actors (power tong and CRT), stage all TTS equipment including the torque turn box, tension/compression load cells, computers, the correct cables, power boxes, (if applicable) explosion proof boxes, inverters, dump valves, have knowledge of connections and thread designs, independently set up and safely operate the TTS system as well as understand torque turn and torque time theory, perform routine cleaning and packing of TTS equipment and be responsible for rigging up and rigging down the TTS tools on the rig site as well as occasionally performing basic maintenance on site when required.

Torque Turn Supervisor:

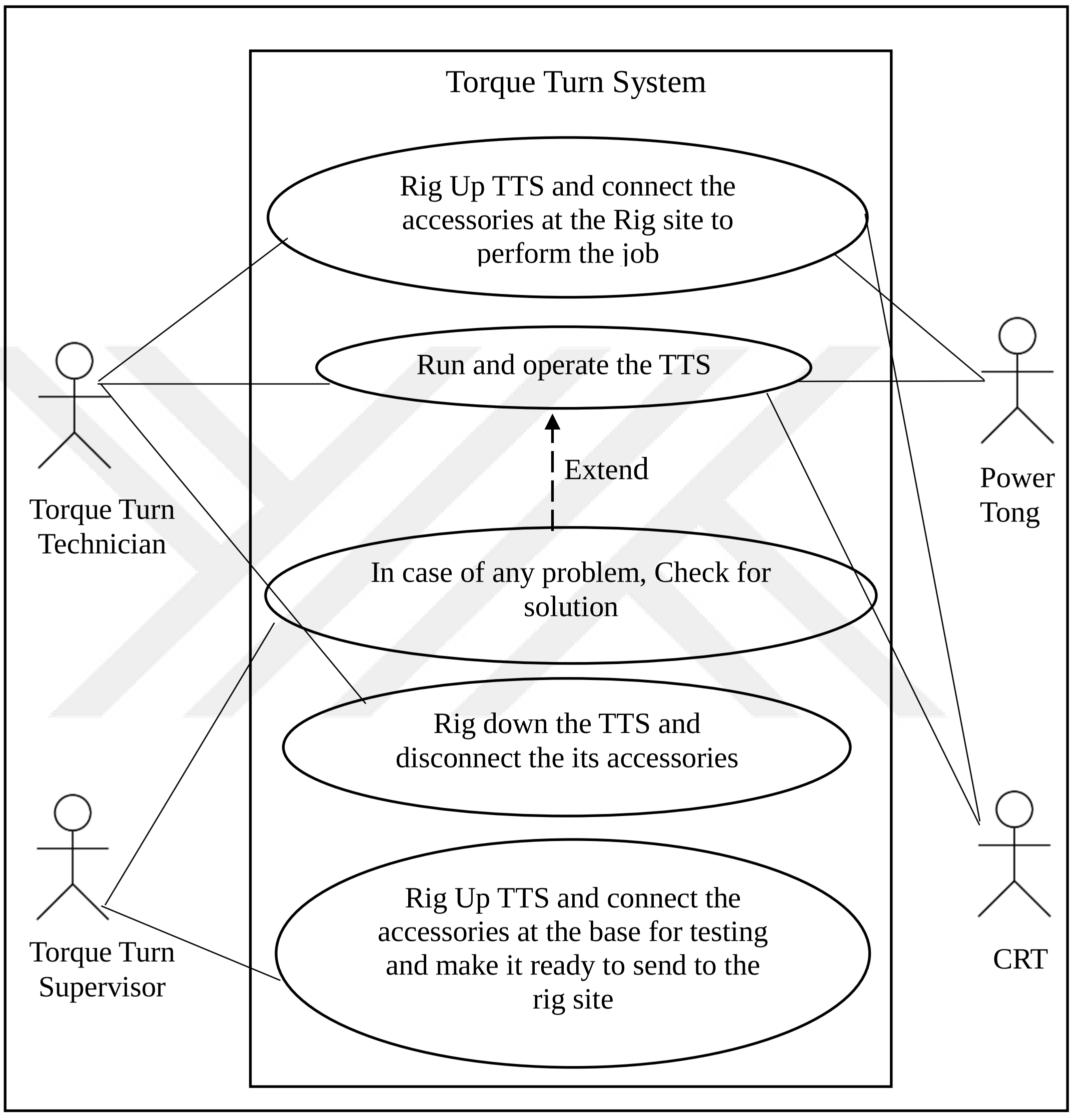
This person’s responsibilities and tasks include maintaining TTS equipment in accordance with the maintenance program, inspecting, assembling, disassembling, and checking the functionality of TTS equipment. He may assist TTS techs to set up and safely operate tubing/casing tong, power units, casing and tubing handling equipment as well as perform routine calibration and troubleshooting on TTS Tension gear i.e., load cell and torque gauges, train new TTS hand personnel in operation and maintenance procedures of power equipment and TTS systems, oversee TTS personnel development of assets in advancement to the next level with the competence development team, maintain an efficient inventory stocking system of the TTS Lab and equipment, upgrade all TTS units so they are using the correct and up-to-date versions of TTS Software and tracking of TTS equipment and technicians’ locations .He is also responsible for extracting job dates through the TTS unit and sending them to customers.

Hydraulic Power Tong:

This is a machine operated by a liquid which moves in a confined space under pressure. It makes a rotating motion when breaking out, or making up casings, tubing, drill pipes and other pipes. This tong has a self-locking mechanism and large capacity pliers used to grip and drill casing and tubing string components so as to apply torque, and make or break casings and tubing placed in drill holes in order to maintain the opening of the well. Some TTS accessories (load cell, turns sensor, BGDU and damp valves) installed on the power tong acquire torque and turns data and send them to the TTS Unit.

Casing Running Tools:

This is a machine used with casing running only, it has the same duties of the power tong but with a difference in the operation type. It makes a rotating motion when breaking out or making up casings, and it helps to provide automation of running casings and ensures that casings reach the bottom without fail. It has been proven that pipe rotation and reciprocation while cementing helps to achieve a more reliable and secure cement job. Casing running tools provide the ability to rotate, circulate, and reciprocate pipes simultaneously during cementing operations. It is built in the TTS accessories (load cell, turns sensor) and it connects with the TTS unit over a wired or wireless connection to obtain the torque and turns data followed by sending the data to the TTS unit.



Torque Turn System
Rig Up TTS and connect the accessories at the Rig site to perform the job
Run and operate the TTS
Extend
In case of any problem, Check for solution
Rig down the TTS and disconnect the its accessories
Rig Up TTS and connect the accessories at the base for testing and make it ready to send to the rig site
Torque Turn Technician
Torque Turn Supervisor
Power Tong
CRT

Figure 10 Torque Turn System Use Case Diagram

3.3 Discussion

As we have presented in this chapter, there are various tasks at the rig site. The turn torque system is one of the most sophisticated series of tasks as it requires both physical activities and mastering computerized equipment. The skills of technicians are crucial. We recall that the researcher has considerable experience in the field.

His knowledge is also a source in addition to the literature reviewed in the second chapter. Thus, we have added the following points of the technicians to be investigated in addition to what we have placed in Section 2.2, Computer literacy, experience in torque control and monitoring systems, and technician behavior as in using equipment.

CHAPTER FOUR

ANALYSIS AND RESULTS

4.1 Introduction

The second and third chapters presented the following points to be investigated at the rig site: education, experience, language, work load and training of the employees, interface, and extra activities. We formulated these points into interview questions. Later, we approached two key informants to validate the questions. Finally, we present Figure 11 and Table 8.

4.2 Interview Questions

Section One: Technician Skills and Qualifications

Q.1. What is your level of education?
Q.2. What is your level of experience in the oil and gas industry?
Q.3. What is your gender?
Q.4. Select the system(s) with which you are familiar.
Q.5. Can you understand the English guides and manuals on your own?
Q.6. Select what you are familiar with.
Q.7. Regarding the Torque Turn System, select from below the certificates you have.
Q.8. How often do you receive assessments from the Torque Turn System Trainer?

Section Two: Work Period

Q.9. How long did your longest work shift at the rig site continue?

Section Three: You may select more than one option.

Q.10. During the running operation, you make fewer errors when you are using…
Q.11. During the running operation, which of the following is easy to use?
Q.12. You work faster when you are using...

Q.13. You find that the Torque Turn software interface is easy to follow.

Q.14. You find that the Torque Turn software interface is robust, i.e., does it crash or does it do what it is not intended to do?

Q.15. You find that the Torque Turn software interface helps you to work quickly.

Section Four: Technician Behavior

Q.16. Do you use copy-paste in repeating tasks? For instance, copying comments from previous graphs and using them for the next graph's comments.

Q.17. You do calculations in your mind when you need to calculate and fill job data such as torque value for maxT., minT., optT., high shT., low shT., dumpT and refT.

Q.18. You use an electronic calculator when you need to calculate and fill job data, such as torque value for maxT., minT., optT., high shT., low shT., dumpT and refT.

Q.19. You double check any data you filled before starting a job.

Q.20. You check the final torque and shoulder point values for each joint that you run in the hole.

Q.21. You left the acquiring or ready-mode window and go to another window, such as when reviewing previous graphs.

Q.22. You do further activity which is not required; for instance, performing calibrations to load cells and torque gauges.

Q.23. You use your personal memory stick to transfer data to or from the Torque Turn System.

Q.24. You use a company memory stick to transfer data to or from the Torque Turn System.

While conducting the interviews, the participants are informed not to disclose any private information related to them and their organizations so as to maintain anonymity.

The third question, about the gender of the technician, yielded an interesting result. All of the respondents appeared to be male while this question was recommended by the person regarding how to validate our question, taking into account the differences between male and females in term of ability in working under pressure, behavior and decision making. Nevertheless, we intended to report it.

4.3 Results

The questionnaire with 24 questions was administered to 81 respondents. The results are as follows:

R1. The analysis reveals a strong relation between making fewer errors and ease of use when using the keyboard. Technicians who used keyboards made fewer errors and they found them easy to use. There were 38 responses with this feedback.

R2. A meaningful relation can be seen between making fewer errors and working faster when using a keyboard. Technicians who use keyboards make fewer errors and find it faster. There were 33 responses with this feedback.

R3. A strong relation can be seen between ease of use and working faster when using a keyboard. Technicians who use keyboards find it easy to use and do their jobs faster. There were 34 responses with this feedback.

R4. The analysis reveals a strong relation between making fewer errors and ease of use when using touch screens. Technicians who use touch screens make fewer errors and found them easy to use. There were 40 responses with this feedback.

R5. The analysis reveals a strong relation between making fewer errors and working faster when using touch screens. Technicians who use touch screens make fewer errors and perform their jobs faster. There were 34 responses with this feedback.

R6. The analysis reveals a strong relation between ease of use and working faster when using touch screens. Technicians who use touch screens found them easy to use and performed their jobs faster. There were 37 responses with this feedback.

R7. The analysis reveals a strong relation between making fewer errors and ease of use when using touch screens with pens. Technicians who use touch screens with pens made fewer errors and they found them easy to use. There were 52 responses with this feedback.

R8. The analysis reveals a strong relation between making fewer errors and working faster when using touch screens with pens. Technicians who used touch screens with pens made fewer errors and performed their jobs faster. There were 51 responses with this feedback.

R9. The analysis reveals a strong relation between ease of use and working faster when using touch screens with pens. Technicians who used touch screens with pens found them easy to use and performed their jobs faster. There were 58 responses with this feedback.

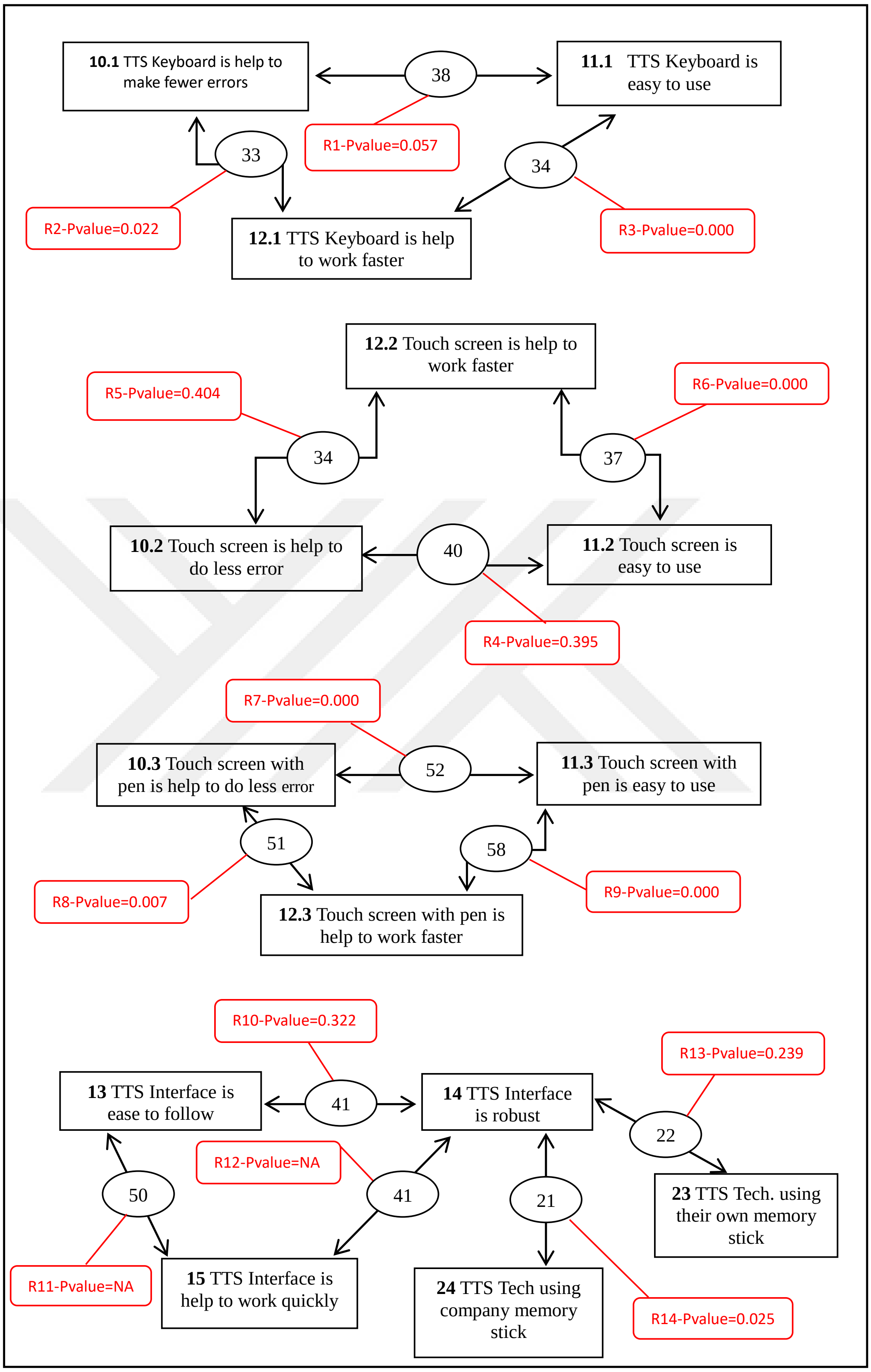
R10. A strong relation is revealed between finding the torque turn system interface easy to follow and finding it robust. Technicians who usually found the interface easy to follow would find it robust. There were 41 responses with this feedback.

R11. A strong relation was revealed between finding the torque turn system interface easy to follow and finding that it helped to work quickly. Technicians who usually found the interface easy to follow would work quickly. There were 50 responses with this feedback.

R12. The analysis reveals a strong relation between finding the torque turn interface robust and finding it helps to work quickly. Technicians who did not find the interface robust did not find that it helped to work quickly. There were 41 responses with this feedback.

R13. The analysis reveals a strong relation between finding the torque turn system interface robust and using a personal memory stick. Technicians who usually found the interface robust used their personal memory sticks to transfer data from and to the torque turn system. There were 22 responses with this feedback.

R14. The last relation revealed through the analysis was the relation between finding the torque turn system interface robust and using a company memory stick. Technicians who usually used a company memory stick to transfer data from and to the torque turn system; they did not find the interface robust. There were 21 only respondents with this feedback.



10.1 TTS Keyboard is help to make fewer errors
38
11.1 TTS Keyboard is easy to use
R1-Pvalue=0.057
33
34
R2-Pvalue=0.022
12.1 TTS Keyboard is help to work faster
R3-Pvalue=0.000
12.2 Touch screen is help to work faster
R5-Pvalue=0.404
R6-Pvalue=0.000
34
37
10.2 Touch screen is help to do less error
40
11.2 Touch screen is easy to use
R4-Pvalue=0.395
R7-Pvalue=0.000
10.3 Touch screen with pen is help to do less error
52
11.3 Touch screen with pen is easy to use
51
58
R8-Pvalue=0.007
R9-Pvalue=0.000
12.3 Touch screen with pen is help to work faster
R10-Pvalue=0.322
R13-Pvalue=0.239
13 TTS Interface is ease to follow
41
14 TTS Interface is robust
22
R12-Pvalue=NA
23 TTS Tech. using their own memory stick
50
41
21
R11-Pvalue=NA
15 TTS Interface is help to work quickly
24 TTS Tech using company memory stick
R14-Pvalue=0.025

Figure 11 Results of the relation between human factors and the number of respondents to each relation after analysis

4.4 Data Analysis

An analysis has been made of the respondents' feedback and we took the feedback results to identify the findings, limitations and future work, as we present later in Chapter 5.

4.4.1 Data Sets

In this section, we present the data collected from the Torque Turn System technicians through a questionnaire with 24 questions. The questionnaire had been distributed to more than 200 TTS technicians; however, we received only 81 responses. The targeted persons were TTS technicians working with various oil and gas companies around MENA. All were male with over 5 years of experience in the field. Our questions focused on the information that yielded through the literature about the factors impacting human error, i.e., experience, education, training and work shifts.

In terms of experience, 68 respondents were senior technicians (over 5 years of experience), 12 respondents were junior technicians (1 to 4 years of experience) and only one respondent was a trainee (less than one year of experience), as shown in figure 12.

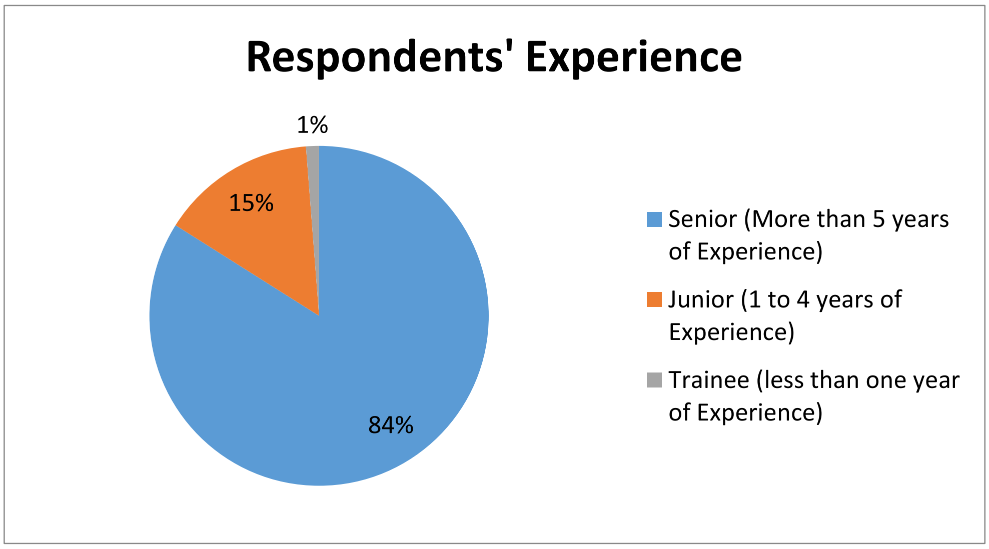


Figure 12 Respondents' experience level

In terms of education, 39 respondents had Bachelor's degrees, 20 had diplomas, 12 had a secondary degree or less, 5 had a master degree and 5 had other certificates, as shown in figure (13).

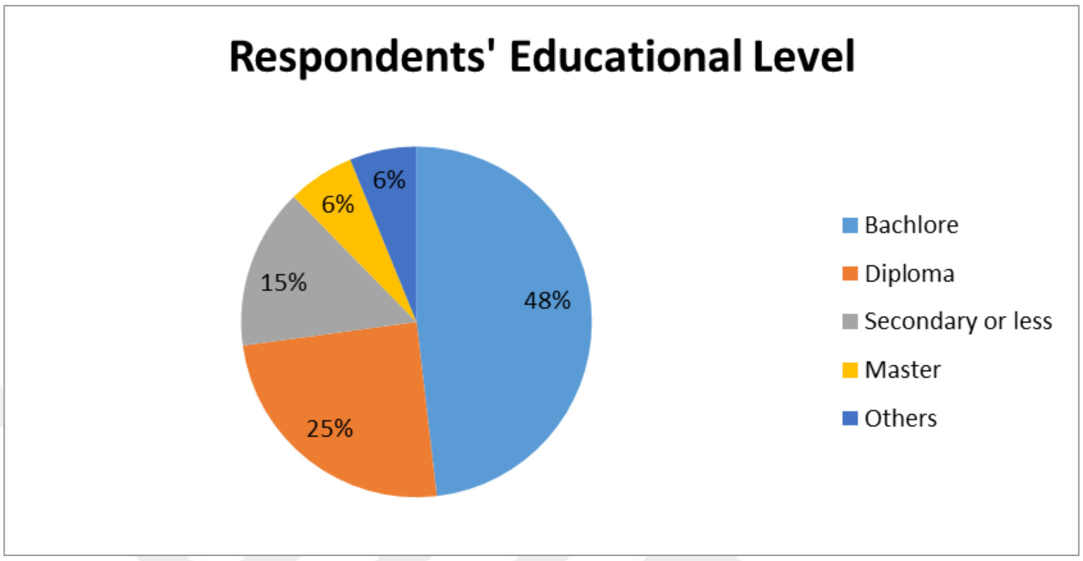


Figure 13 Respondents' educational level

In terms of working shifts, 24 respondents worked a 12-hour working shift, 20 worked an 18-hour working shift, 15 worked a 6-hour working shift, 11 worked a 30-hour working shift, while only 9 worked a 24 hour working shift and one respondent preferred to not to respond to this question, as shown in the figure (14).

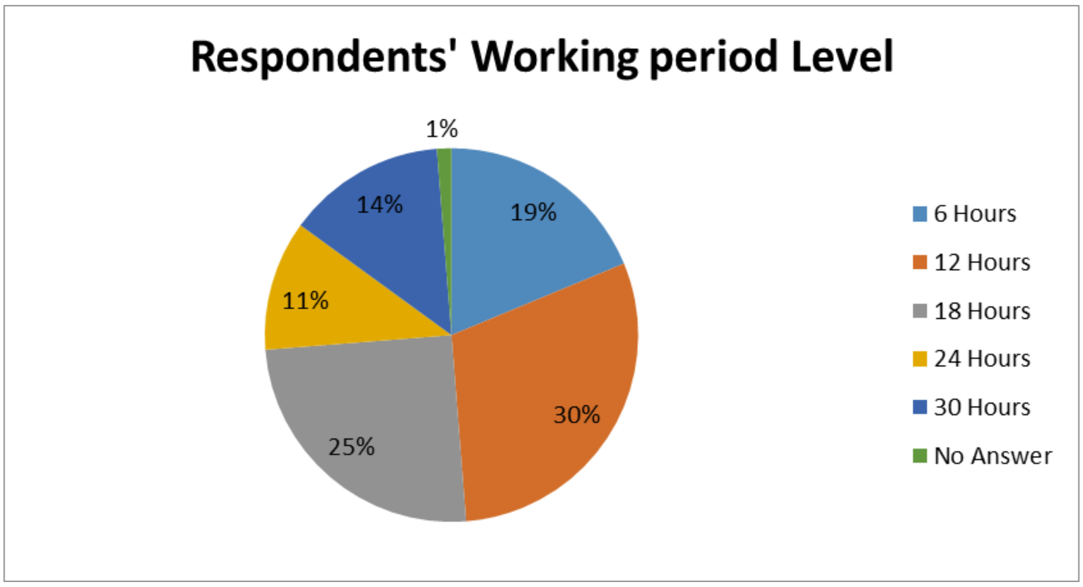


Figure 14 Respondents' working period

In terms of periodic training, 59 respondents would take periodic assessment and training, 15 respondents would take irregular assessment and training, while only 6 respondents had never received any training and one respondent preferred to not to respond to this question, as shown in the figure (15).

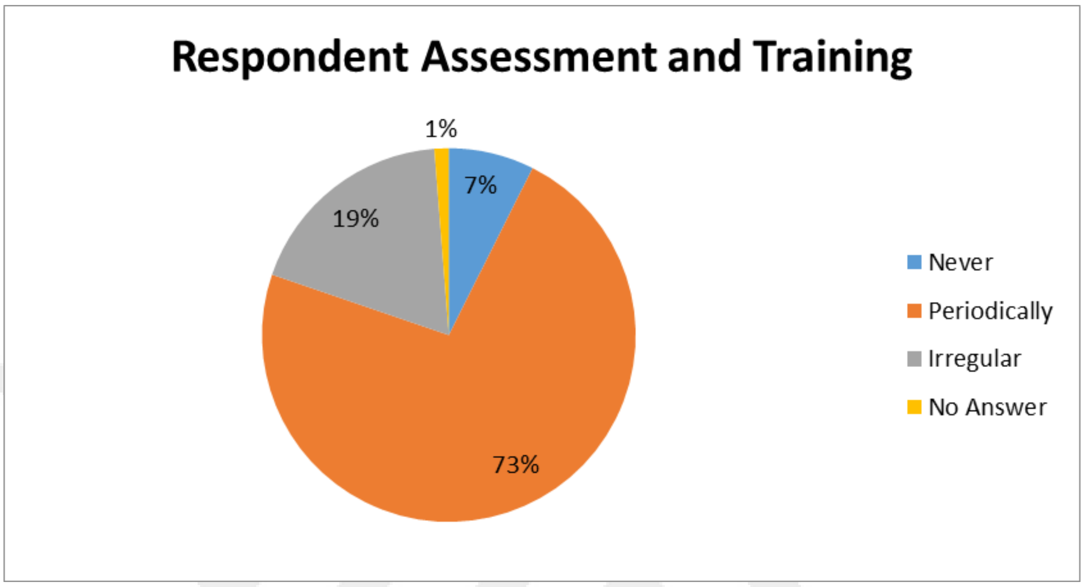


Figure 15 Attendance of periodic assessment and training

4.4.2 Statistical Analysis

We used the chi-square test for independence to compare every two variables of our questionnaire in a contingency table to see whether the distributions of categorical variables differ from one another. Too many relations were yielded through the analysis, so we ignored the relations whose difference in proportion were not significant as the p-value was much greater than 0.05. Therefore, we listed the relation whose difference of proportion is significant with a p-value near to or less than 0.05, as a low value means there is a high correlation between the two sets of data.

Chi-square uses the formula below to perform the test.

$$\chi_c^2 = \sum \frac{(O_i - E_i)^2}{E_i}$$

Where:

- c = degree of freedom
- O = questionnaire observation value
- E =our expected value
- The summation symbol means that we perform a calculation for every single data item in the data set.

As shown above in Section 4.3, the analysis yielded only 14 relations with a significant difference of proportion, as shown in the table (6).

Table 6 P-value of analysis relations

Relation	P-Value	Relation	P-Value	Relation	P-Value
R1	0.057	R6	0.000	R11	N.A.
R2	0.022	R7	0.000	R12	N.A.
R3	0.000	R8	0.007	R13	0.239
R4	0.395	R9	0.000	R14	0.025
R5	0.404	R10	0.322		

We can see in the above table that relations R1, R2, R3, R6, R7, R8, R9 and R14 were significant with chi-square values with p-values less than 0.05, while relations R4, R5, R10 and R13 were not significant with chi-square values with p-values greater than 0.05. Moreover, relations R11 and R12 had zero p-values, but we still recommend those relations for our work and future studies should be established to focus on those relations.

CHAPTER 5

FINDINGS AND FUTURE WORK

5.1 Findings

Finding 1: There is a strong relation between less error and ease of use; there were 38 responses on using a keyboard (as shown above in R1) and 40 responses on using touch screens (as shown above in R4). While the relation is stronger when using touch screens with a pen, there were 52 responses on this question (as shown above in R7).

Finding 2: There is a strong relation between less error and working faster. There are 33 responses on using Keyboard (as shown above in R2) and 34 responses on using touch screen (as shown above in R5). While the relation was stronger for using touch screens with pens, there was 51 responses to it (as shown above in R8).

Finding 3: There is a strong relation between ease of use and working faster. There were 34 responses to using keyboards (as shown above in R3) and 37 responses to using touch screens (as shown above in R6). While the relation is stronger for using touch screens with a pen, there were 58 responses to it (as shown above in R9).

Finding 4: There is a strong relationship between finding the interface easy to follow and finding it robust. There were 41 responses for that (as shown above in R10).

Finding 5: There is a strong relationship between finding the interface robust and finding that it helps to perform work quickly. There were 41 responses to it (as shown above in R12).

Finding 6: There is a strong relationship between finding the interface easy to use and finding that it helps to perform work quickly. There were 50 responses to it (as shown above in R11).

Finding 7: There is a strong relationship between finding the interface robust and using a personal memory stick. There were 21 responses to it (as shown in R13).

Finding 8: There is a strong relationship between finding the interface robust and using a company memory stick. There were 21 responses to it (as shown above in R14).

Finding 9: A major similarity was found between relation no. 1 and relation no. 4, as their numbers respondents were nearly identical, numbering 38 for relation no. 1 and 40 for relation no. 4. A difference was found for relation no. 7 when comparing it with relation nos. 1 and 4 as there were 52 respondents for relation no. 7.

Finding 10: A major similarity was found between relation no. 2 and relation no. 5, as the numbers of respondents were nearly identical, numbering 33 for relation no. 2 and 34 for relation no. 5. A difference was found with relation no. 8 when comparing it with relation nos. 2 and 5, as there were 51 respondents for relation no. 8.

Finding 11: A major similarity was found between relation no. 3 and relation no. 6, as the numbers of respondents were nearly identical, numbering 34 for relation no. 3 and 37 for relation no. 6. A difference was found with relation no. 9 when comparing it with relation nos. 3 and 6, as there were 58 respondents for relation no. 7.

Finding 12: We couldn’t find enough information as this information might not public or may not many people worked on such subject.

5.2 Research Limitations

The work presented in this thesis has the following limitations:

Limitation 1: Collecting Information

The targeted firms are distributed over broad areas in the Middle East and North Africa. Not all of them were investigated and it was difficult to conduct face-to-face interviews with every TTS technician in those firms.

Limitation 2: Information Resources

There was a lack of literature related to errors due to end users, especially in oil and gas operation systems. Moreover, senior management refused to allow the researcher to have access to information about previous systems errors and maintenance activities. Moreover, data didn’t collect at the wok time.

Limitation 3: System Analysis and Design

In Chapter 3, we identified the actors, actions and information associated with torque turn systems and represented them by drawing use case diagrams. The technology, databases behind the system, structured ER diagrams and deployment models needed to be taken into consideration when analyzing the system.

Limitation 4: Software Development

The development in software should be taken into account, as I/O devices and the interface will change to follow updates of infrastructure.

Limitation 5: Legal Issues

Organizational barriers on the legal side are excluded from this study. Some important questions were excluded from the questionnaire because the respondents would not answer them as they would have impacted their company policy.

5.3 Future Work

Future Work 1: Conducting Meetings and Surveys

To overcome limitation 1, we recommend data collection through face-to-face interviews as much as possible as data will be collected through semi-structured interviews. Data collected in this manner can be evaluated and the results may provide precise information regarding end user errors in the oil and gas sector.

Future Work 2: Access Data Sources

From limitation 2, it becomes necessary to obtain real examples through the oil and gas sector regarding end user errors and software maintenance, also it is necessory to collect the data directly through the field at the time of work.

Future Work 3: System Development

To overcome limitations 3 and 4, a real study should be conducted on the torque turn systems taking into account the continuous development of its interface and accessories.

Future Work 4: Investigation of Legal Issues

To overcome limitation 5, further research is needed to investigate in depth a firm’s rules and legal barriers as detailed research on laws and regulations will strengthen the study.

5.4. Conclusion

Many factors can impact system availability, one of which includes end user behavior, the major reason for system unavailability. In this thesis, we presented the factors that can have an impact on end user behavior i.e., experience, training, the interface, working load and so on.

Our study shows the impact of computer input devices (keyboards, touch screens, touch screens with pens) on end user behavior and thus on system availability. In findings 1, 2 and 3, we learned that using a touch screen with a pen is easier and is the best method to make fewer errors in addition to helping to perform work quickly.

In addition, our study shows the impact of the interface on system availability. In findings 4, 5 and 6, we learn that the end user works faster when the interface is easy to use.

Memory sticks also have an impact on system availability. In findings 7 and 8, we learn that there is a relationship between the use of memory sticks and finding the interface to be robust. The respondents, who were usually using their personal memory sticks, saw the interface as robust, while the respondents using company memory sticks.

In this thesis, we have found that many external and internal factors can influence end user behavior and thus, affect operational availability and efficiency. External factors include system interfaces, input and output devices and memory sticks. Internal factors include training, experience, education and working shifts.

About the research questions:

“Can end user behavior influence oil and gas operations availability and efficiency in TTS?”

In light of our findings in Section 5.1 and considering the limitations in Section 5.2, we believe the results obtained in our analysis support our hypothesis that end user behavior has an impact on oil and gas availability and efficiency, keeping in mind that most of the potential factors that influence user behavior are due to a lack of experience, education, training, system interfaces and working shifts.

REFERENCES

[1] Schneidewind, N. F. (1975). Analysis of error processes in computer software. In *ACM Sigplan Notices* (Vol. 10, No. 6, pp. 337-346). ACM.

[2] Kumar, P., & Khan, R. A. (2015). Classification of Software Requirement Errors: A Critical Review. *International Journal of Computer Applications*, *132*(7), 9-14.

[3] Huang, F., Liu, B., Song, Y., & Keyal, S. (2014). The links between human error diversity and software diversity: Implications for fault diversity seeking. *Science of Computer Programming*, *89*, 350-373.

[4] Xu, H., Chen, W., & Qian, H. (2011). Research on error feedback mechanism of information system. In *2011 2nd International Conference on Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC)* (pp. 5112-5115). IEEE.

[5] Gupta, S., Mishra, A., & Chawla, M. (2016). Analysis and recommendation of common fault and failure in software development systems. In *2016 International Conference on Signal Processing, Communication, Power and Embedded System (SCOPES)* (pp. 1730-1734). IEEE.

[6] Dong, L., Melhem, R., Mossé, D., Ghosh, S., Heimerdinger, W., & Larson, A. (1999). Implementation of a transient-fault-tolerance scheme on DEOS-A technology transfer from an academic system to an industrial system. In *Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium* (pp. 56-65). IEEE.

[7] Glass, R. L. (1981). Persistent software errors. *IEEE Transactions on Software Engineering*, (2), 162-168.

[8] Memon, A., & Xie, Q. (2004). Using transient/persistent errors to develop automated test oracles for event-driven software. In *Proceedings. 19th International Conference on Automated Software Engineering, 2004.* (pp. 186-195). IEEE.

[9] Sosnowski, J. (1994). Transient fault tolerance in digital systems. *IEEE Micro*, *14*(1), 24-35.

[10] Wu, J., & Shin, K. G. (2005). SMRP: fast restoration of multicast sessions from persistent failures. In *2005 International Conference on Dependable Systems and Networks (DSN'05)* (pp. 150-159). IEEE.

[11] Luo, Q., Poshyvanyk, D., Nair, A., & Grechanik, M. (2016). FOREPOST: a tool for detecting performance problems with feedback-driven learning software testing. In *Proceedings of the 38th International Conference on Software Engineering Companion* (pp. 593-596). ACM.

[12] Nistor, A., Chang, P. C., Radoi, C., & Lu, S. (2015). Caramel: Detecting and fixing performance problems that have non-intrusive fixes. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering* (Vol. 1, pp. 902-912). IEEE.

[13] Chambers, C. C. W. (2014). Helping end-user programmers find and fix performance problems in visual code.

[14] Zhu, L., Jin, H., & Liao, X. (2016). A Tool to Detect Performance Problems of Multi-threaded Programs on NUMA Systems. In *2016 IEEE Trustcom/BigDataSE/ISPA* (pp. 1145-1152). IEEE.

[15] Van Gurp, J., & Bosch, J. (2002). Design erosion: problems and causes. *Journal of systems and software*, *61*(2), 105-119.

[16] Pérez-Castillo, R., de Guzmán, I. G. R., & Piattini, M. (2011). Diagnosis of software erosion through fuzzy logic. In *2011 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)* (pp. 49-56). IEEE.

[17] De Silva, M., & Perera, I. (2015). Preventing software architecture erosion through static architecture conformance checking. In *2015 IEEE 10th International Conference on Industrial and Information Systems (ICIIS)* (pp. 43-48). IEEE.

[18] Jiang, L., & Xu, G. (2007). Modeling and analysis of software aging and software failure. *Journal of systems and software*, *80*(4), 590-595.

[19] Pusatli, O. T. (2009). *Interoperability and Information System Replacement in the Health Sector*. University of Newcastle.

[20] Wu, H., & Wolter, K. (2015). Software aging in mobile devices: Partial computation offloading as a solution. In *2015 IEEE International Symposium on Software Reliability Engineering Workshops (ISSREW)* (pp. 125-131). IEEE.

[21] Adobe Flash CS4 Professional (10.0.2) addresses issues regarding the compiling of large project files. (2017), retrieved from http://helpx.adobe.com/.

[22] Yip, S. W., & Lam, T. (1994). A software maintenance survey. In *Proceedings of 1st Asia-Pacific Software Engineering Conference* (pp. 70-79). IEEE.

[23] Sosnowski, J., Dobrzyński, B., & Janczarek, P. (2017). Analyzing problem handling schemes in software projects. *Information and Software Technology*, *91*, 56-71.

[24] Endres, A. (1975). An analysis of errors and their causes in system programs. *IEEE Transactions on Software Engineering*, (2), 140-149.

[25] López, C., & Salmeron, J. L. (2012). Monitoring software maintenance project risks. *Procedia Technology*, *5*, 363-368.

[26] Jambor-Sadeghi, K., Ketabchi, M. A., Chue, J., & Ghiassi, M. (1994). A systematic approach to corrective maintenance. *The Computer Journal*, *37*(9), 764-778.

[27] Kozlov, D., Koskinen, J., Markkula, J., & Sakkinen, M. (2007). Evaluating the impact of adaptive maintenance process on open source software quality. In *First International Symposium on Empirical Software Engineering and Measurement (ESEM 2007)* (pp. 186-195). IEEE.

[28] Tai, A. T., & Alkalai, L. (1998). On-board maintenance for long-life systems. In *Proceedings. 1998 IEEE Workshop on Application-Specific Software Engineering and Technology. ASSET-98 (Cat. No. 98EX183)* (pp. 69-74). IEEE.

[29] DURNAN, H. (2017). What To Do When an Error Occurs on the LOWIS Server, retrieved from: https:// softwaresupport.weatherford.com.

[30] Case Study: ONGC Improves Field Simulation Runtime by 20 Times Using the INTERSECT High-Resolution Simulator (2016), retrieved from: https://www.slb.com.

[31] DURNAN, H. (2017). Changes to Updating the LOWIS Client in 7.0, retrieved from: https://softwaresupport.weatherford.com.

[32] Tolteq Release Notes - iGC, retrieved from: https://www.NOV.com.

[33] Gao, B., Guo, L., Ma, L., & Wang, K. (2012). Corrective maintenance process simulation algorithm research based on process interaction. In *Proceedings of the IEEE 2012 Prognostics and System Health Management Conference (PHM-2012 Beijing)* (pp. 1-5). IEEE.

[34] Li, J., Stålhane, T., Kristiansen, J. M., & Conradi, R. (2010). Cost drivers of software corrective maintenance: An empirical study in two companies. In *2010 IEEE International Conference on Software Maintenance* (pp. 1-8). IEEE.

[35] Rao, B. S., & Sarda, N. L. (2005). Execution model for outsourced corrective maintenance. In *The Fifth International Conference on Computer and Information Technology (CIT'05)* (pp. 944-948). IEEE.

[36] Evanco, W. M. (2001). Prediction models for software fault correction effort. In *Proceedings Fifth European Conference on Software Maintenance and Reengineering* (pp. 114-120). IEEE.

[37] Schrank, M. J., Anderson, A. C., Bisignani, M. E., & Boyce, G. W. (1995). Assessing the capabilities of military software maintenance organizations. In *Proceedings of 14th Digital Avionics Systems Conference* (pp. 314-319). IEEE.

[38] Dalla Preda, M., Gabbrielli, M., Giallorenzo, S., Lanese, I., & Mauro, J. (2015). Developing correct, distributed, adaptive software. *Science of Computer Programming*, *97*, 41-46.

[39] Sherer, S. A. (1992). Cost benefit analysis and the art of software maintenance. In *Proceedings Conference on Software Maintenance 1992* (pp. 70-77). IEEE.

[40] Rashid, A., Wang, W. Y., & Dorner, D. (2009). Gauging the differences between expectation and systems support: the managerial approach of adaptive and perfective software maintenance. In *2009 Fourth International Conference on Cooperation and Promotion of Information Resources in Science and Technology* (pp. 45-50). IEEE.

[41] Trümper, J., Beck, M., & Döllner, J. (2012). A visual analysis approach to support perfective software maintenance. In *2012 16th International Conference on Information Visualisation* (pp. 308-315). IEEE.

[42] Cote, V., & Pierre, D. S. (1990). A model for estimating perfective software maintenance projects. In *Proceedings. Conference on Software Maintenance 1990* (pp. 328-334). IEEE.

[43] Pavlic, Z., Lugaric, T., & Silic, M. (2012). Debugging in consumer-programming oriented environments. In *2012 Proceedings of the 35th International Convention MIPRO* (pp. 841-846). IEEE.

[44] Hanmer, R. S., & Mendiratta, V. B. (2010). Rejuvenation with workload migration. In *2010 International Conference on Dependable Systems and Networks Workshops (DSN-W)* (pp. 80-85). IEEE.

[45] Huang, Y., Kintala, C., Kolettis, N., & Fulton, N. D. (1995). Software rejuvenation: Analysis, module and applications. In *Twenty-Fifth International Symposium on Fault-Tolerant Computing. Digest of Papers* (pp. 381-390). IEEE.

[46] Levitin, G., Xing, L., & Ben-Haim, H. (2018). Optimizing software rejuvenation policy for real time tasks. *Reliability Engineering & System Safety*, *176*, 202-208.

[47] Vaidyanathan, K., & Trivedi, K. S. (2005). A comprehensive model for software rejuvenation. *IEEE Transactions on Dependable and Secure Computing*, *2*(2), 124-137.

[48] Garg, S., Puliafito, A., Telek, M., & Trivedi, K. (1998). Analysis of preventive maintenance in transactions based software systems. *IEEE transactions on Computers*, *47*(1), 96-107.

[49] Trivedi, K. S., Vaidyanathan, K., & Goseva-Popstojanova, K. (2000). Modeling and analysis of software aging and rejuvenation. In *Proceedings 33rd Annual Simulation Symposium (SS 2000)* (pp. 270-279). IEEE.

[50] Parnas, D. L. (1994). Software aging. In *Proceedings of 16th International Conference on Software Engineering* (pp. 279-287). IEEE.

[51] Kahol, K., & Saeidi, F. (2009). Haptic system to alert users before impending human errors. In *2009 IEEE International Workshop on Haptic Audio visual Environments and Games* (pp. 36-41). IEEE.

[52] Dekker, S. W. (2007). Doctors are more dangerous than gun owners: a rejoinder to error counting. *Human factors*, *49*(2), 177-184.

[53] Dix, A., Finaly, J., Abowd, G. D., & Beale, R. Human-computer interaction (3RD Ed.) (pp. 258-288).

[54] Ko, A. J., & Myers, B. A. (2005). A framework and methodology for studying the causes of software errors in programming systems. *Journal of Visual Languages & Computing*, *16*(1-2), 41-84.

[55] Huffman Hayes, J., Mohamed, N., & Gao, T. H. (2003). Observe-mine-adopt (OMA): an agile way to enhance software maintainability. *Journal of Software Maintenance and Evolution: Research and Practice*, *15*(5), 297-323.

[56] Hayes, J. H., Patel, S. C., & Zhao, L. (2004). A metrics-based software maintenance effort model. In *Eighth European Conference on Software Maintenance and Reengineering, 2004. CSMR 2004. Proceedings.* (pp. 254-258). IEEE.

[57] Reinecke, P., & Wolter, K. (2010). A simulation study on the effectiveness of restart and rejuvenation to mitigate the effects of software ageing. In *2010 IEEE Second International Workshop on Software Aging and Rejuvenation* (pp. 1-6). IEEE.

[58] Matias, R., Andrzejak, A., Machida, F., Elias, D., & Trivedi, K. (2014). A systematic differential analysis for fast and robust detection of software aging. In *2014 IEEE 33rd International Symposium on Reliable Distributed Systems* (pp. 311-320). IEEE.

[59] Yahaya, J. H., Abidin, Z. N. Z., & Deraman, A. (2015). Perspective and perception on software ageing: The empirical study. In *2015 10th International Conference on Computer Science & Education (ICCSE)* (pp. 365-370). IEEE.

[60] Prakash, G. (2010). Achieving agility in adaptive and perfective software maintenance. In *2010 14th European Conference on Software Maintenance and Reengineering* (pp. 61-62). IEEE.

[61] Yahaya, J. H., Abidin, Z. N. Z., Ali, N. M., & Deraman, A. (2013). Software ageing measurement and classification using Goal Question Metric (GQM) approach. In *2013 Science and Information Conference* (pp. 160-165). IEEE.

[62] Abidin, Z. N. Z., Yahaya, J. H., & Deraman, A. (2015). Software ageing measurement model (SAMM): The conceptual framework. In *2015 International Conference on Electrical Engineering and Informatics (ICEEI)* (pp. 456-461). IEEE.

[63] Smidts, C. (1999). A stochastic model of human errors in software development: impact of repair times. In *Proceedings 10th International Symposium on Software Reliability Engineering (Cat. No. PR00443)* (pp. 94-103). IEEE.

[64] Yu, W. D., Barshefsky, A., & Huang, S. T. (1997). An empirical study of software faults preventable at a personal level in a very large software development environment. *Bell Labs Technical Journal*, *2*(3), 221-232.

[65] Weyers, B., Burkolter, D., Kluge, A., & Luther, W. (2010). User-centered interface reconfiguration for error reduction in human-computer interaction. In *2010 Third International Conference on Advances in Human-Oriented and Personalized Mechanisms, Technologies and Services* (pp. 52-55). IEEE.

[66] Yamada, S., Nonaka, T., & Hase, T. (2010). Suitable graphical user interface selection based on human errors using analytic hierarchy process. In *2010 IEEE International Conference on Systems, Man and Cybernetics* (pp. 3028-3032). IEEE.

[67] Huang, F., & Bin, L. I. U. (2017). Software defect prevention based on human error theories. *Chinese Journal of Aeronautics*, *30*(3), 1054-1070.

[68] Li, Z., Lu, S., Myagmar, S., & Zhou, Y. (2006). CP-Miner: Finding copy-paste and related bugs in large-scale software code. *IEEE Transactions on software Engineering*, *32*(3), 176-192.

[69] Askarinejadamiri, Z., Zulzallil, H., Ghani, A. A. A., & Wei, K. T. (2017). Impact propagation of human errors on software requirements volatility. *Int. J. Adv. Comput. Sci. Appl.*, *8*(2), 227-237.

[70] Walia, G. S., & Carver, J. C. (2009). A systematic literature review to identify and classify software requirement errors. *Information and Software Technology*, *51*(7), 1087-1109.

[71] Senders, J. W., & Moray, N. P. (1995). Human error: Cause, prediction, and reduction.

[72] Norman, D. (2013). *The design of everyday things: Revised and expanded edition*. Basic books.

[73] Embrey, D. (2005). Understanding human behaviour and error. *Human Reliability Associates*, *1*(2005), 1-10.

[74] Ernstsen, J., & Nazir, S. (2018). Human error in pilotage operations. *TransNav: International Journal on Marine Navigation and Safety of Sea Transportation*, *12*.

[75] Shorrock, S. T., & Kirwan, B. (2002). Development and application of a human error identification tool for air traffic control. *Applied ergonomics*, *33*(4), 319-336.

[76] Machida, F., Kim, D. S., Park, J. S., & Trivedi, K. S. (2008). Toward optimal virtual machine placement and rejuvenation scheduling in a virtualized data center. In *2008 IEEE International Conference on Software Reliability Engineering Workshops (ISSRE Wksp)* (pp. 1-3). IEEE.

[77] Etman, E., & Halawa, A. (2007). Safety culture, the cure for human error: A critique. *Dmitriy Zhukov*, 115.

[78] Curzon, P., & Blandford, A. (2001). Detecting multiple classes of user errors. In *IFIP International Conference on Engineering for Human-Computer Interaction* (pp. 57-71). Springer, Berlin, Heidelberg.

[79] GOULD, J. & LOVELL, S. (2009). Human error analysis at a refinery, *IChemE*, 549 – 553.

[80] Kirwan, B. (1998). Human error identification techniques for risk assessment of high risk systems—Part 1: review and evaluation of techniques. *Applied ergonomics*, *29*(3), 157-177.

[81] Ahlstrom, V., & Hartman, D. G. (2001). *Human error in airway facilities* (No. DOT/FAA/CT-TN01/02). William J. Hughes Technical Center (US).

[82] ASYALI, E. (2003). Impact of Man-Machine Interface on Maritime Casualties. *Proceedings of International Association of Maritime Universities (IAMU)*, 89-90.

[83] Stanton, N. A., Salmon, P., Harris, D., Marshall, A., Demagalski, J., Young, M. S., & Dekker, S. (2009). Predicting pilot error: testing a new methodology and a multi-methods and analysts approach. *Applied ergonomics*, *40*(3), 464-471.

[84] Rooney, J. J., Heuvel, L. N. V., & Lorenzo, D. K. (2002). Reduce human error. *Quality progress*, *35*(9), 27-36.

[85] Cannon, A. B., Strawderman, L., & Burch, R. (2015). Evaluating change in user error when using ruggedized handheld devices. *Applied ergonomics*, *51*, 273-280.

[86] Grigoraş, G., & Bărbulescu, C. (2013). Human errors monitoring in electrical transmission networks based on a partitioning algorithm. *International Journal of Electrical Power & Energy Systems*, *49*, 128-136.

[87] Ruckart, P. Z., & Burgess, P. A. (2007). Human error and time of occurrence in hazardous material events in mining and manufacturing. *Journal of hazardous materials*, *142*(3), 747-753.

[88] Isaac, A., Shorrock, S. T., & Kirwan, B. (2002). Human error in European air traffic management: the HERA project. *Reliability Engineering & System Safety*, *75*(2), 257-272.

[89] Chen, T., Yesilada, Y., & Harper, S. (2010). What input errors do you experience? Typing and pointing errors of mobile Web users. *International journal of human-computer studies*, *68*(3), 138-157.

[90] Polet, P., Vanderhaegen, F., & Zieba, S. (2010). An Iterative Learning System to Learn from Human Errors in Transport Systems. *IFAC Proceedings Volumes*, *43*(13), 47-52.

[91] Cho, W. C., & Ahn, T. H. (2019). A classification of electrical component failures and their human error types in South Korean NPPs during last 10 years. *Nuclear Engineering and Technology*, *51*(3), 709-718.

[92] Wenwen, S., Fuchuan, J., Qiang, Z., & Jingjing, C. (2011). Analysis and control of human error. *Procedia Engineering*, *26*, 2126-2132.

[93] Sam, D., Velanganni, C., & Evangelin, T. E. (2016). A vehicle control system using a time synchronized Hybrid VANET to reduce road accidents caused by human error. *Vehicular communications*, *6*, 17-28.

[94] Bes, M. O. (1999). A case study of a human error in a dynamic environment. *Interacting with Computers*, *11*(5), 525-543.

[95] Arnstein, F. (1997). Catalogue of human error. *British journal of anaesthesia*, *79*(5), 645-656.

[96] Nistor, A., Jiang, T., & Tan, L. (2013). Discovering, reporting, and fixing performance bugs. In *Proceedings of the 10th Working Conference on Mining Software Repositories* (pp. 237-246). IEEE Press.

[97] Hashemi, M., & Herbert, J. (2014). Uixsim: A user interface experience analysis framework. In *2014 5th International Conference on Intelligent Systems, Modelling and Simulation* (pp. 29-34). IEEE.

[98] Vishakha, Ms. (2014). Achieving Agility in Software Maintenance. In *International Journal of Engineering Research & Technology (IJERT)*, 3(5) (pp. 174 178).

[99] Techlog software new release, (2019). https://www.software.slb.com

[100] Stolee, K. T., Elbaum, S., & Rothermel, G. (2009). Revealing the copy and paste habits of end users. In *2009 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC)* (pp. 59-66). IEEE.

[101] Thompson, C. M., Cooper, S. E., Kolaczkowski, A. M., Bley, D. C., Forester, J. A., & Wreathall, J. (1997). The application of ATHEANA: A technique for human error analysis. In *Proceedings of the 1997 IEEE Sixth Conference on Human Factors and Power Plants, 1997.'Global Perspectives of Human Factors in Power Generation'* (pp. 9-13). IEEE.

[102] Love, P. E., Edwards, D. J., Irani, Z., & Walker, D. H. (2009). Project pathogens: The anatomy of omission errors in construction and resource engineering project. *IEEE Transactions on Engineering Management*, *56*(3), 425-435.

[103] Sträter, O., Dang, V., Kaufer, B., & Daniels, A. (2004). On the way to assess errors of commission. *Reliability Engineering & System Safety*, *83*(2), 129-138.

[104] Saha, S., Lozi, J. P., Thomas, G., Lawall, J. L., & Muller, G. (2013). Hector: Detecting resource-release omission faults in error-handling code for systems software. In *2013 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)* (pp. 1-12). IEEE.

[105] Hayllar, B., Veal, A. J., & Sherval, M. (1996). *Pathways to research*. Rigby Heinemann.

APPENDIX

CURRICULUM VITAE

PERSONAL INFORMATION

Surname, Name: ALRAWI, Layth Nabeel

Date and Place of Birth: 05 March 1986, Mosul, Iraq

Marital Status: Married

Phone: +9647701712496

Email: Lnalrawi@gmail.com

EDUCATION

Degree	Institution	Year of Graduation
M.Sc.	Cankaya University, Graduate School of Natural and Applied Science, Computer engineering department	2019
B.Sc.	Mosul Univ., Faculty of Computer sciences and Mathematics, Mosul	2008
High School	Al-Resala Al-Islamiyah Secondary school, Mosul	2004

WORK EXPERIENCE

Year	Place	Enrollment
2017- Present	Parker Drilling Company, ITS Division	Operations Manager, North Iraq
2014-2016	Nabors Drilling, Tesco Corporation Division	TRS Supervisor
2008-2014	Weatherford Oil Tools, TRS Dept.	JAM, TTS Engineer

LANGUAGE SKILLS

- Arabic-Mother Language.
- English (reading and writing).
- Turkish (Intermediate).