Journal of Insights in Basic and Applied Sciences
ISSN: 3106-1737
Volume 2, Issue 3, 2026, Pages: 262-275
Journal homepage: https://jibas.com.ly/index.php/jibas/en

Software Maintenance in Healthcare Information Systems: A Case Study Approach

Ashraf Faraj Saed Albarki[1*], Eman Salem Ali Elmathkour[2], Esam Miftah Abdulnabi Aboudoumat[3]
[1] Department of Computer, College of Arts and Sciences- Qumins, University of Benghazi, Libya.
[2] Department of Computer, Institute for Engineering Professions - Al-Majori - Benghazi, Libya.
[3] Department of Computer, College of Arts and Sciences Qumins, University of Benghazi, Benghazi, Libya

صيانة البرمجيات في نظم المعلومات الصحية: دراسة حالة

أشرف فرج سعد البركي [1*]، ايمان سالم علي المذكور[2]، عصام مفتاح عبد النبي بودومات [3]
[1] قسم الحاسوب ، كلية الاداب والعلوم قمينس، جامعة بنغازي، بنغازي، ليبيا.
[2] قسم الحاسوب، المعهد العالي للمهن الهندسية الماجوري، ، بنغازي، ليبيا.
[3] قسم الحاسوب ، كلية العلوم والتقنية قمينس، بنغازي، ليبيا.

[*]Corresponding author: ashraf.elburki@uob.edu.ly

Received: May 26, 2026	Accepted: June 25, 2026	Published: July 30, 2026

Abstract

We live in the age of software and information technology, where software maintenance plays a pivotal role in developing systems within organizations to meet user needs and keep pace with the rapid advancements in information technology and modern techniques. After software delivery to the client, maintenance begins. Software maintenance is one of the most time-consuming and resource-intensive phases of the software development lifecycle. This paper aims to study the maintenance of a real-world software system in the healthcare sector. The system's operational problems were analyzed, the types of maintenance applied were identified, and the procedures followed to address technical and organizational challenges were determined. The study's findings demonstrate that implementing a structured software maintenance methodology effectively contributes to improved performance, reduced downtime, and increased user satisfaction.

Keywords: Software maintenance; Software Cost; Software Understanding.

الملخص

نعيش في عصر البرمجيات وتكنولوجيا المعلومات، حيث تلعب صيانة البرمجيات دورًا محوريًا في تطوير الأنظمة داخل المؤسسات لتلبية احتياجات المستخدمين ومواكبة التطورات السريعة في تكنولوجيا المعلومات والتقنيات الحديثة. تبدأ عملية الصيانة بعد تسليم البرمجيات للعميل، وهي من أكثر مراحل دورة حياة تطوير البرمجيات استهلاكًا للوقت والموارد. تهدف هذه الورقة البحثية إلى دراسة صيانة نظام برمجي واقعي في قطاع الرعاية الصحية. تم تحليل المشكلات التشغيلية للنظام، وتحديد أنواع الصيانة المطبقة، وتحديد الإجراءات المتبعة لمعالجة التحديات التقنية والتنظيمية. تُظهر نتائج الدراسة أن تطبيق منهجية منظمة لصيانة البرمجيات يُسهم بفعالية في تحسين الأداء، وتقليل وقت التوقف، وزيادة رضا المستخدمين.

الكلمات المفتاحية: صيانة البرمجيات**;** تكلفة البرمجيات**;** فهم البرمجيات.

Introduction

The growth of institutions is large and rapid, and this leads to constant changes in their work and requirements . At the present time, software has to be adaptable to keep up with the needs of this rapid growth. Therefore, software engineers have taken the concept of the ability to update, adapt, and maintain software into their

calculations while developing this software. Since this concept has become urgent in this field, we find that huge numbers of old systems (Legacy System), which was built in a way that does not fit with this concept, which led to the emergence of an important problem in how to deal with this software and how to Developing and maintaining it and making it fit this rapid and sometimes radical change.

Organizations have many software systems that are important to work with within the organization, but they must be changed and updated according to the organization's requirements. Most large companies seek to develop their current systems instead of developing new systems. Studies show that 75% of software professionals are busy with one aspect of maintenance and development activity.

Software upkeep is an inherent component of the Software Development Life Cycle. Software developers must continuously be on the lookout for ways to rectify and enhance their work in order to be competitive and relevant[6].

According to Rech and Bunse [6], the software development life cycle consists of a series of stages: planning, analysis and specication of requirements, design, implementation, testing, delivery and deployment as well as operation and maintenance. Upon completion of its development, the software product is delivered to the client and regardless of the application domain, size and complexity, it will continue to undergo modications over time. Academics and software industry professionals recognize software maintenance as the most challenging and costly stage of the software life cycle [1].

Using the right software maintenance techniques and strategies is a critical part of keeping any software running for a long period of time and keeping customers and users happy [11].

Software does not get tired; it needs to meet some new requirements that enhanced the software system. The changes made in the software system can be performed by software maintenance. Software Maintenance comes under process when a software team delivers a successful project to its client within a fixed time. Software Maintenance is the modification of a software product after delivery. The Modification can be done to improve performance, correct faults and to adapt the product to a modified environment[9].

Since software maintenance is an important part of the software development life cycle and is essential to ensure that the software continues to meet the needs of users over time, it is also important to consider the cost and effort required for software maintenance when planning and developing a software system.[14]

Basic definitions Software maintenance is defined in IEEE Standard 1219 [IEEE93] as:

The modification of a software product after delivery to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.[3،4]

2.Why Software Maintenance is Needed:

There are a number of factors that provide the motivation for maintenance:

To provide continuity of service:

Systems need to keep running. Maintenance activities aimed at keeping a system operational include bug-fixing, recovering from failure, and accommodating changes in the operating system and hardware.

To support mandatory upgrades:

This type of change would be necessary because of such things as amendments to government regulations.

To support user requests for improvements:

On the whole, the better a system is, the more it will be used and the more the users will request enhancements in functionality.

3.To facilitate future maintenance work:

It is often financially and commercially justifiable to initiate change solely to make future maintenance easier. This would involve such things as code and database restructuring, and updating of documentation.[4]

Bug fixing:

The process of finding and fixing errors and problems in software.

Enhancements:

The process of adding new features or improving existing features to meet evolving user needs.

Performance improvement:

The process of improving the speed, efficiency, and reliability of software.

Porting and migration:

The process of adapting software to run on new hardware or software platforms.

Reengineering:
The process of improving the design and architecture of software to make it more maintainable and scalable.
Documentation:
The process of creating, updating, and maintaining software documentation, including user manuals, technical specifications, and design documents [13] .

4. Advantages of Software Maintenance*:*
Improved software quality:
Regular software maintenance helps ensure that the software is running correctly and efficiently, and continues to meet the needs of users.
Improved security:
Maintenance can include security updates and patches, which helps ensure that the software is protected from potential threats and attacks.
Increased user satisfaction:
Regular software maintenance helps keep the software up to date and relevant, leading to increased user satisfaction and adoption.
Extended software life:
Proper software maintenance can extend the life of the software, allowing it to be used for longer periods of time and reducing the need for costly replacements.
Cost savings:
Regular software maintenance can help prevent larger, more expensive problems, reducing the total cost of ownership of the software.
5.Disadvantages of software maintenance*:*
Cost*:*
Software maintenance can be time-consuming and expensive, and may require significant resources and expertise.
Schedule disruptions*:*
Maintenance can cause disruptions to the normal schedule and operations of the software, leading to potential downtime and inconvenience.
Complexity:
Maintaining and updating complex software systems can be difficult, and requires specialized knowledge and expertise.
Risk of introducing new bugs*:*
Fixing bugs or adding new features can introduce new bugs or problems, making it important to test the software thoroughly after maintenance.
User resistance:
Users may resist changes or updates to the software, leading to reduced satisfaction and adoption[13].
5.Software maintenance types:
Software maintenance is divided into four types, which are as follows:

1. Corrective Software maintenance.
2. Preventative Software Maintenance .
3. Perfective Software Maintenance .
4. Adaptive Software Maintenance .

1. **Corrective software maintenance:**

is the standard, conventional kind of maintenance (for software and anything else for that matter). Corrective software maintenance is crucial when anything goes wrong with a piece of software, such as bugs and errors. Since they could have a big impact on how the software works in general, these need to be rectified immediately away. Software vendors frequently have the chance to address issues that need corrective maintenance thanks to user-submitted bug reports. The capacity to recognize problems and address them before people become aware of them is a further advantage that will improve the business's standing and reliability (no one likes an error message after all).

2. **Preventative Software Maintenance:**

refers to software updates made to a product to ensure its future viability. Therefore, software maintenance improvements that prepare for future prospective changes are preventative. This includes handling the old material and simplifying the scaling and maintenance of the code. It also includes identifying and correcting any hidden flaws in the product before they become operational flaws. Therefore, preventive software maintenance

usually happens in the background. Instead of changing the headlines, consider organization and preparedness. Preventive software modifications are probably not seen by the users, but they nonetheless have an advantage afterwards. This is due to the possibility that preventative maintenance will make larger modifications more easily implemented in the future. (Along with continued daily steadiness).

3. Perfective software maintenance:

is concerned with how the system's requirements and features change over time. Users may detect details they have missed or propose new features they would like to see included in the apps as they use them, which might lead to future initiatives or improvements. A portion of the job is taken over by perfective software maintenance, which includes both adding features that can improve user experience and deleting elements that are ineffective and non-functional. This might include functions that are not used or ones that prevent from achieving the objectives[6].

4. Adaptive Software Maintenance:

adaptive maintenance. The environment of an application might vary due to a variety of variables, such as new hardware, security concerns, and technical advancements. In most situations, these changes happen more often, thus software that doesn't get regular adaptive maintenance rapidly gets out of date. The capacity of the program to execute on a certain platform, which consists of the operating system (OS), hardware, and network, is mostly affected by adaptive modifications. Since they concentrate on how the product works inside, these changes often have little effect on consumers. An application's overall functionality often isn't impacted when it is integrated with new technology, while performance metrics like scalability and speed may see a minor increase. When adaptive is finished, users are far more likely to notice because it can prohibit the software from operating with the newest gadgets[6].

6. maintenance activities:

IEEE has introduced a framework for software maintenance activities in a sequential manner. It can be used iteratively to have the possibility to, As shown in Figure 1.

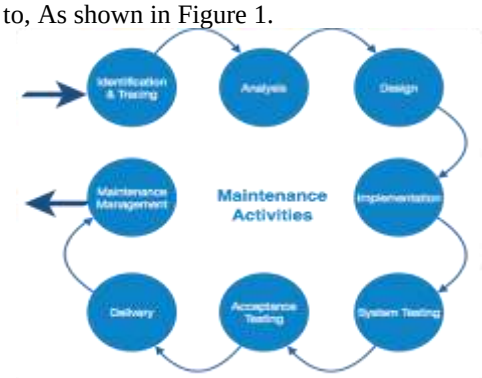

Figure 1: Maintenance Activities by IEEE

Identification & Tracing :

The process of determining what part of the software needs to be modified (or maintained). This can be user-generated or identified by the software developer itself depending on the situation and specific fault.

Analysis :

The modification's effects on the system, including any concerns for safety and security, are examined. If the likely impact is significant, other solutions are sought. The requirements specifications are then manifested into a set of necessary adjustments. Analysis and calculation of the cost of modification and maintenance are completed.

Design :

The required requirements established in the earlier stage are used to guide the design of any new modules that need to be replaced or updated. For verification and validation, test cases are produced.

Implementation :

With the aid of the structured design produced during the design phase, the new modules are programmed. Unit testing should be done simultaneously by every coder.

System Testing :

Testing is conducted on newly developed modules during integration. New modules and the system are both subjected to integration testing. Finally, using regressive testing techniques, the system is tested as a whole.

Accepttance Testing :

The system is internally tested before being put to the test of user approval. If at this point users raise any difficulties, they will either be resolved or documented for future iterations.

Delivery :

After passing the acceptance test, the system is installed either from scratch or as part of a short update package and distributed throughout the enterprise. After the program is delivered, the last testing is performed at the client's end.

Maintenance management - System maintenance must include configuration management. Version control tools are used to handle patches, semi-versions, and versions [6].

7. Software Maintenance Cost:

The cost of software maintenance can be high. However, this doesn't negate the importance of software maintenance. In certain cases, software maintenance can cost up to two-thirds of the entire software process cycle or more than 50% of the SDLC processes. The costs involved in software maintenance are due to multiple factors and vary depending on the specific situation. The older the software, the more maintenance will cost, as technologies (and coding languages) change over time. Revamping an old piece of software to meet today's technology can be an exceptionally expensive process in certain situations. an addition, engineers may not always be able to target the exact issues when looking to upgrade or maintain a specific piece of software. This causes them to use a trial and error method, which can result in many hours of work. There are certain ways to try and bring down software maintenance costs. These include optimizing the top of programming used in the software, strong typing, and functional programming.

When creating new software as well as taking on maintenance projects for older models, software companies must take software maintenance costs into consideration. Without maintenance, any software will be obsolete and essentially useless over time. All software companies should have a specific strategy in place to tackle software maintenance in an effective and complete manner.

Documentation is one important strategy in software development. If software documentation isn't up to date, upgrading can be seemingly impossible. The documentation should include info about how the code works, solutions to potential problems, etc. QA is also an important part of a software maintenance plan. While QA is important before an initial software launch, it can also be integrated much earlier in the process (as early as the planning stage) to make sure that the software is developed correctly and to give insight into making changes when necessary [12].

Real-world elements that influence maintenance costs :

- Any software is generally thought to be 10 to 15 years old.
- Older software that was designed to run on sluggish computers with little memory and storage cannot compete with freshly released upgraded software running on contemporary technology.
- It gets more expensive to maintain outdated software as technology progresses.
- The majority of maintenance engineers are novices who fix problems by trial and error.
- The original program structure is frequently readily damaged by alterations, making it difficult for adjustments to be made afterwards.
- Changes are frequently not documented, which might lead to further conflicts in the future.[6]

Issues at the software level impacting maintenance costs

- The Software Program's Structure .
- Programming Language .
- Reliance on the outside environment .
- Reliability and accessibility of the staff.[6].

8. Program comprehension :

The program comprehension process is one of the most important parts, which is the first task in software maintenance [7]. Whenever computer software engineers are engaged with evolving an existing system, it is estimated that 50 to 90 % of the time in software maintenance is spent on program understanding. A large amount of time needs to be used to understand a system before making enhancements, where program understanding helps lessen the cost and time for software maintenance [2].

Program comprehension can be supported by producing design models from existing software [10], supported by the advise which says "practice with reverse engineering techniques improves ability to understand a given system quickly and efficiently."[2].

Program understanding is based on artifacts of the existing system; such knowledge and experience with the software system may exist in users', maintainers', and original builders' heads; and system history is documented in the form of bug reports and change records. Creating software system understanding includes obtaining information from source code, manuals, and the executing software system. Figure 6 illustrates the sources of information for creating system understanding.
There are other technologies that can be used to aid program understanding, such as browsing the source code manually, performing static and/or dynamic analysis of the subject system [8].

9. How to reduce maintentenance effort:

There are five ways to reduce maintenance costs, which are as follows:

1. Hire Experts
2. Prevent Breakdowns
3. Choose the Right Technology
4. Use Third-Party Support
5. Eliminate Repeated Failures [15].

10. Related works:

[Alexandre L'Erario, Hellen Christine Seródio Thomazinho and Jose Augusto Fabri , 2020]
They presented a paper titled: An Approach to Software Maintenance: A Case Study in Small and Medium-Sized Businesses IT Organizations, The purpose of this paper is to present a software maintenance approach used in small and medium-sized business (SMB) organizations in Brazil. Currently, these organizations represent 95.5% of the software companies in the country. The approach presented here indicates how SMB IT companies have improved their software maintenance processes. Multiple case studies were performed to validate this approach. The outcomes showed that strategies associated with managing users' knowledge and development/maintenance teams are relevant to increase the maintenance process effectiveness. This approach involves three aspects: users' knowledge management, maintenance team knowledge and the management and maintenance process. This improvement includes reducing time and also minimizing the number of tickets. The response time for tickets resolution to the end user has been reduced. In addition, IT organizations have minimized the effects associated with both staff and client turnovers[1].
[B. Kitchenham et al.,2002]
This study deals with software maintenance effort estimation techniques and assessing their efficiency. Software maintenance effort estimation is an essential element in software maintenance project management. This study focused on evaluating the efficiency of software maintenance effort estimation techniques. Data were collected from 145 software maintenance projects implemented by a single third-party company. Linear regression analysis was used to evaluate the efficiency of different estimation techniques. The study found that the estimation model based on software size and maintenance type factors was the most accurate. The model was able to predict actual maintenance efforts with an accuracy of up to 63%. In addition, the study found that the accuracy of estimates depends on other factors, such as software quality and maintenance team efficiency. The study came up with the following recommendations:

- The estimation model based on software size and maintenance type factors can be used as a basis for estimating software maintenance efforts.
- Other factors, such as software quality and maintenance team efficiency, should be considered when making effort estimates[5].

11. The practical side:

11.1 Methodology :

This study was based on a realistic system study and a detailed and in-depth analysis of the software system within its real-world environment.

Study Type:

An applied analytical study based on observation, document analysis, and system logs.

11.2 System Description:

A medical clinic management system that has been in operation for some time and is used daily to manage patient data, appointments, and medical records was selected.

11.3 Data Collection Tools for the System Under Study:

- Error Logs.
- Performance Reports.
- User Feedback.
- Code Review .

11.4 System Evaluation Criteria:

- Performance.
- Stability.
- Maintainability.
- User Satisfaction.

11.5 System Description:

- System Name: Clinic Management System.
- Operating Environment: Windows 8.0.
- Programming Language: Visual Studio 2010.
- Database: Access 2007.

11.6 System Functions:

- Patient Management.
- Appointment Booking.
- Medical Records.
- Financial Invoicing, Reports, and Inquiries.

11.7 System Problem Analysis:

The analysis of the system under study revealed several key problems, including:

1 .Slow response times when data volume increases.
2 Frequent errors during the entry and updating of medical records.
3 Difficulty in modifying the system due to inadequate documentation .4.Incompatibility of some components with operating system updates.

11.8 Software maintenance activities applied to the system under study:

Corrective maintenance:

Data storage errors were fixed by modifying queries and improving exception handling.

The code before maintenance (the incorrect code)

```
Imports System.Data.SqlClient

Dim con As New SqlConnection("Data Source=.;Initial Catalog=ClinicDB;Integrated Security=True")

Sub SavePatient()
    Dim sql As String
    sql = "INSERT INTO Patients VALUES ('" & txtName.Text & "','" & txtAge.Text & "')"

    Dim cmd As New SqlCommand(sql, con)

    con.Open()
    cmd.ExecuteNonQuery()
    con.Close()

    MessageBox.Show("تم حفظ البيانات")
End Sub
```

Problems:

1. No Try…Catch function.
2. Program fails if data is incorrect.
3. Column names not specified.

The code after corrective maintenance:

```
Imports System.Data.SqlClient

Dim con As New SqlConnection("Data Source=.;Initial Catalog=ClinicDB;Integrated Security=True")

Sub SavePatient()
    Try
        Dim sql As String = "INSERT INTO Patients (PatientName, Age) VALUES (@Name, @Age)"
        Dim cmd As New SqlCommand(sql, con)

        cmd.Parameters.AddWithValue("@Name", txtName.Text)
        cmd.Parameters.AddWithValue("@Age", Convert.ToInt32(txtAge.Text))

        con.Open()
        cmd.ExecuteNonQuery()
        con.Close()

        MessageBox.Show("تم حفظ البيانات بنجاح", "نجاح", MessageBoxButtons.OK, MessageBoxIcon.Information)

    Catch ex As Exception
        MessageBox.Show("حدث خطأ أثناء حفظ البيانات: " & ex.Message,
                        "خطأ", MessageBoxButtons.OK, MessageBoxIcon.Error)
        con.Close()
    End Try
End Sub
```

Corrective maintenance was implemented by modifying insecure code queries and replacing them with parameterized queries, in addition to improving exception handling using Try-Catch, which reduced operational errors and ensured that data was stored securely and stably.

Adaptive Maintenance:

Included updating software libraries and ensuring system compatibility with the latest operating system versions.

After the operating system upgrade:

- Database connectivity issues appeared.
- Some devices do not support legacy SQL Server authentication.
- Provider errors appeared.

Cause: Using an outdated provider and not adjusting compatibility with the latest systems.

Code prior to adaptive maintenance (not compatible)

```
Imports System.Data.OleDb

Dim con As New OleDbConnection(
    "Provider=SQLOLEDB;Data Source=.;Initial Catalog=ClinicDB;User ID=sa;Password=123"
)

Sub LoadData()
    con.Open()
    Dim cmd As New OleDbCommand("SELECT * FROM Patients", con)
    cmd.ExecuteReader()
    con.Close()
End Sub
```

Problems:

- Using an outdated provider (SQLOLEDB).
- Compatibility issues with modern Windows.
- Lower performance.

The code after adaptive maintenance is compatible with modern systems.

```
Imports System.Data.SqlClient

Dim con As New SqlConnection(
    "Data Source=.;Initial Catalog=ClinicDB;Integrated Security=True"
)

Sub LoadData()
    Try
        con.Open()
        Dim cmd As New SqlCommand("SELECT * FROM Patients", con)
        Dim reader As SqlDataReader = cmd.ExecuteReader()
        con.Close()

    Catch ex As Exception
        MessageBox.Show("خطأ في الاتصال بقاعدة البيانات: " & ex.Message)
        con.Close()
    End Try
End Sub
```

Adaptive maintenance was implemented by updating the database communication libraries and migrating from legacy providers incompatible with modern operating system versions to the SqlClient library, which contributed to improved compatibility and stability and ensured the continued operation of the system without failures.

We encountered a problem after installing the system on a 64-bit Windows operating system.

The following error appeared:

The requested .NET Framework Data Provider is not installed

Reason:

- The program is compiled as a 32-bit system.
- Using the Provider is not compatible with 64-bit systems.

State before adaptive maintenance

Project setup:

- Target CPU: x86.
- Using a legacy provider.

Calling code

```
Imports System.Data.OleDb

Dim con As New OleDbConnection(
    "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=ClinicDB.mdb"
)
```

Problems:

• Jet OLEDB does not work on 64-bit systems.

• The program fails to run on modern systems.

Adaptive Maintenance (Solution)

Changing Project Settings in Visual Studio 2010:

Project:→ Properties→ Compile→ Advanced Compile Options→ Target CPU = Any CPU

The program is compatible with both 32-bit and 64-bit systems.

It uses SqlClient, which is compatible with 64-bit systems.

```
Imports System.Data.SqlClient

Dim con As New SqlConnection(
    "Data Source=.;Initial Catalog=ClinicDB;Integrated Security=True"
)

Sub LoadPatients()
    Try
        con.Open()
        Dim cmd As New SqlCommand("SELECT * FROM Patients", con)
        Dim reader As SqlDataReader = cmd.ExecuteReader()

        While reader.Read()
            Console.WriteLine(reader("PatientName").ToString())
        End While

        con.Close()

    Catch ex As Exception
        MessageBox.Show("32/64 خطأ توافق-bit: " & ex.Message)
        con.Close()
    End Try
End Sub
```

Adaptive maintenance was implemented to address compatibility issues with 64-bit architecture operating systems by modifying the project settings to Any CPU and updating the database connection libraries using SqlClient, thus ensuring efficient system operation on both 32-bit and 64-bit environments.

Improvement and Maintenance:
Search algorithms were improved and database indexes were added, resulting in reduced response time.

Problem:
When searching for a patient by name:
Response is very slow as the number of records increases.
Cause:

- Using LIKE '%text'%
- Lack of indexes

The code before the optimization maintenance (poor performance) :

```
Imports System.Data.SqlClient

Dim con As New SqlConnection(
    "Data Source=.;Initial Catalog=ClinicDB;Integrated Security=True"
)

Sub SearchPatient()
    Dim sql As String = "SELECT * FROM Patients WHERE PatientName LIKE '%" & txtSearch.Text & "%'"
    Dim cmd As New SqlCommand(sql, con)

    con.Open()
    Dim da As New SqlDataAdapter(cmd)
    Dim dt As New DataTable()
    da.Fill(dt)
    con.Close()

    DataGridView1.DataSource = dt
End Sub
```

Problems:

- Full Table Scan.
- Unoptimized query.
- Does not use indexes.

Optimization maintenance (performance improvement)

Search algorithm improvement (Parameterized + Starts With)

```
Imports System.Data.SqlClient

Dim con As New SqlConnection(
    "Data Source=.;Initial Catalog=ClinicDB;Integrated Security=True"
)

Sub SearchPatientOptimized()
    Try
        Dim sql As String =
            "SELECT PatientID, PatientName, Age " &
            "FROM Patients WHERE PatientName LIKE @Name + '%'"

        Dim cmd As New SqlCommand(sql, con)
        cmd.Parameters.AddWithValue("@Name", txtSearch.Text)

        con.Open()
        Dim da As New SqlDataAdapter(cmd)
        Dim dt As New DataTable()
        da.Fill(dt)
        con.Close()

        DataGridView1.DataSource = dt

    Catch ex As Exception
        MessageBox.Show("خطأ أثناء البحث: " & ex.Message)
        con.Close()
    End Try
End Sub
```

•Allows the use of indexes, reducing execution time.
Improvement maintenance was implemented by optimizing the search algorithms and replacing unoptimized queries with transaction-based queries, in addition to creating non-clustered indexes on the most frequently used search fields, resulting in reduced response time and improved system performance.

Preventive Maintenance:
Parts of the code have been restructured and documentation updated to facilitate future maintenance.

- Before Preventive Maintenance:
- Duplicate code in multiple forms
- Difficulty in future modifications
- Lack of documentation explaining function functions

Before Preventive Maintenance: (Disorganized code)

```
Imports System.Data.SqlClient

Sub LoadPatients()
    Dim con As New SqlConnection("Data Source=.;Initial Catalog=ClinicDB;Integrated Security=True"
    con.Open()
    Dim cmd As New SqlCommand("SELECT * FROM Patients", con)
    Dim da As New SqlDataAdapter(cmd)
    Dim dt As New DataTable()
    da.Fill(dt)
    con.Close()
    DataGridView1.DataSource = dt
End Sub
```

Problems:
• Creating a new connection in each function.
•Code duplication.
•Difficulty in testing and modification.
After preventative maintenance (documentation +Refactoring)
Create a custom class to connect to the database

```
Imports System.Data.SqlClient

''' <summary>
''' فئة مسؤولة عن إدارة الاتصال بقاعدة البيانات
''' </summary>
Public Class DatabaseHelper

    Private Shared ReadOnly connectionString As String =
        "Data Source=.;Initial Catalog=ClinicDB;Integrated Security=True"

    ''' <summary>
    ''' إرجاع اتصال جاهز بقاعدة البيانات
    ''' </summary>
    Public Shared Function GetConnection() As SqlConnection
        Return New SqlConnection(connectionString)
    End Function

End Class
```

Using classes in forms

```
Imports System.Data.SqlClient

Sub LoadPatients()
    Try
        Using con As SqlConnection = DatabaseHelper.GetConnection()
            Dim sql As String = "SELECT PatientID, PatientName FROM Patients"
            Dim da As New SqlDataAdapter(sql, con)
            Dim dt As New DataTable()
            da.Fill(dt)
            DataGridView1.DataSource = dt
        End Using
    Catch ex As Exception
        MessageBox.Show("خطأ في تحميل البيانات: " & ex.Message)
    End Try
End Sub
```

●Cleaner, reusable code, easier future maintenance

Preventive maintenance was implemented by restructuring the code and separating the database communication logic into independent classes, in addition to updating the internal code documentation, which contributed to improving maintainability and reducing the likelihood of future errors.

12. Results and Discussion:

Maintenance results showed a significant improvement in system performance of approximately 40%, a decrease in operational errors, and an increase in user satisfaction. These results confirm the effectiveness of implementing different types of maintenance within a clear, systematic framework.

Challenges faced by the maintenance team during system updates and maintenance:

- Lack of documentation in the initial system release.
- Absence of original developers.
- Limited time and resources.

13.Recommendations:

This study confirms that software maintenance is essential for ensuring the sustainability of software systems. The paper recommends integrating maintenance plans into the early development phases, adhering to software standards, and continuously updating documentation.

14. Conclusion

This study concluded that software maintenance is a fundamental and crucial element in the information systems lifecycle, particularly in healthcare systems that demand high levels of reliability, continuity, and accuracy. The case study results demonstrated that implementing a structured and comprehensive maintenance methodology, encompassing corrective, adaptive, optimization, and preventative maintenance, effectively contributes to improving system performance, reducing operational errors, enhancing maintainability, and increasing user satisfaction.

The study also showed that neglecting or postponing maintenance leads to increased long-term costs and system complexity, especially in the absence of documentation or when the system is not compatible with modern technological and environmental developments. Practical experience demonstrated that improving the code structure, updating the operating environment, enhancing search algorithms, and implementing sound programming practices have a tangible impact on system stability and sustainability.

Therefore, this paper recommends integrating maintenance plans into the early stages of software development, adhering to standards, continuously updating documentation, and focusing on building flexible systems that can adapt to future changes. Adopting maintenance as an ongoing strategic process, rather than a follow-up action, is a key factor in ensuring software quality and continuity, particularly in vital sectors such as healthcare.

References

[1] Alexandre L'Erario, Hellen Christine Seródio Thomazinho and José Augusto Fabri (2020).An Approach to Software Maintenance: A Case Study in Small and Medium-Sized Businesses IT Organizations,

International Journal of Software Engineering and Knowledge Engineering, Vol. 30, No. 5 (2020) 603–630.DOI: 10.1142/S0218194020500217.

[2] Ali, M. R. (2005).Why teach reverse engineering, ACM SIGSOFT Software Engineering Notes, Vol. 30, No. 4, pp. 1–4, July 2005.

[3] Bennett, K. H. and Rajlich, V. T.Software Maintenance and Evolution, Research Institute for Software Evolution, University of Durham, UK; Department of Computer Science, Wayne State University, Detroit, MI 48202.

[4] Grubb, Penny and Takang, Armstrong A. (2003).Software Maintenance: Concepts and Practice (Second Edition). World Scientific Publishing Co. Pte. Ltd.

[5] Kitchenham, Barbara, Pfleeger, Shari Lawrence, McColl, Beth, & Eagan,Suzanne (2002). An empirical study of maintenance and development estimation accuracy,Journal of Systems and Software, 64 (2002) 57–77.

[6] Pabasara, B. K. T. and Hettige, H. B. (2022).Maintain to gain fame: A Research Review on Software Maintenance, 15th International Research Conference of KDU 2022. Available on ResearchGate.

[7] Sadiq, J. and Waheed, T. (2011).Reverse Engineering & Design Recovery: An Evaluation of Design Recovery Techniques, in Computer Networks and Information Technology (ICCNIT), Proceedings of the 2011 International Conference, pp. 325–332.

[8] Tilley, S. R. and Smith, D. (1995).Perspectives on Legacy System Reengineering, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, Draft Version 0.3, 1995.

[9] Uttamjit Kaur and Gagandeep Singh (2015).A Review on Software Maintenance Issues and How to Reduce Maintenance Efforts, International Journal of Computer Applications (0975–8887), Volume 118, No. 1, May 2015.

[10] F. Wotawa, "Software Maintenance Lecture Notes", Institut for Software technologie, Inffeldgasse 16b/2, A-8010 Graz, Austria , Oct 2009.

[11] [Online] Available:https://cpl.thalesgroup.com/software-monetization/four-types-of-software-maintenance[Accessed: 06-09-2024, Time: 10:00 PM].

[12] [Online] Available:https://cpl.thalesgroup.com/software-monetization/four-types-of-software-maintenance[Accessed: 11-02-2024, Time: 07:05 PM].

[13] [Online] Available:https://www.geeksforgeeks.org/software-engineering-software-maintenance/[Accessed: 06-09-2024, Time: 10:00 PM].

[14] [Online] Available:https://www.geeksforgeeks.org/software-engineering-software-maintenance/[Accessed: 06-09-2024, Time: 10:00 PM].

[15] [Online] Available:https://www.revelo.com/blog/software-maintenance-costs [Accessed: 11-09-2024, Time: 12:00 AM].