Journal of Information Systems Engineering and Management
2022, 7(1), 11689
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Guiding legacy systems for evolution

PmatE: a case study of maintenance and engineering

André Monteiro[1], Gonçalo Vieira[1 *]

[1] University of Aveiro, Portugal
***Corresponding Author:** andremonteiro@ua.pt

Citation: André Monteiro, Gonçalo Vieira (2022). Guiding legacy systems for evolution. PmatE: a case study of maintenance and engineering. *Journal of Information Systems Engineering and Management*, *7*(1), 11689. https://doi.org/10.55267/iadt.07.11689

ARTICLE INFO

Received: 11 Oct. 2021
Accepted: 31 Dec. 2021

ABSTRACT

Even though software change is inevitable, accurate maintenance can extend software lifespan in a subtle way when both budget and time constraints get in the way of software replacement. In the University of Aveiro, the project PmatE – a quiz web platform created to encourage students to like Math – emerged in the early 1990's and stacked several applications over the decades without major planning, cleaning or upgrade. This resulted in a huge-sized framework that was crucial to be always available and online and had high operational cost, leading to an increasing amount of technical debt. After 3 decades, the project was studied, refactored and refurbished, leading to a stable consistent framework ready for evolution and software spinouts. This work shows how to manage and engineer solutions to maintain a legacy system and evolve it even when tied up to heavy constraints.

CCS CONCEPTS: Software and its engineering • Software creation and management • Software post-development issues • Maintaining software • Software evolution

Keywords: Software engineering, software maintenance, software evolution

INTRODUCTION

Software systems that are developed specially for an organization have usually a long lifetime (Sommerville, 2016). Such systems, developed many years ago, using technologies and interfaces that are now obsolete, are still in use and remain vital for the normal functioning of business and represent a substantial investment for organizations (Newby, 1994).

Authors in (Bennett, 1995) state clear characteristics for a legacy system: age (more than 10 years old), the size (hundreds of thousands of lines of code), an old programming language, a long history of maintenance that increased the entropy and the maintenance costs, the relevance of the mission and the internal domain knowledge.

The solution seems simple: software renewal or building from scratch. But there is a significant business risk in simply scrapping a legacy system and replacing it with a system that has been developed using recent technology (Sommerville, 2016). One can come across several issues: the system may be using application/service rules that are not properly documented elsewhere; there are clients/processes that are reliant on the legacy system; legacy systems rarely have complete specifications - during their lifetime usually have undocumented major changes.

However, even these large and durable systems must change in order to remain useful. But upgrading is also usually very expensive. The real challenge consists of finding cost-effective and quality solutions and evolving them in order to meet new requirements. Typical solutions available in managing legacy systems include ordinary maintenance, reverse engineering, restructuring, reengineering, migration, wrapping and also discarding (de Lucia et al., 2001).

One of these examples is the project PmatE – Projecto Matemática Ensino (Pinto et al., 2007), which emerged in 1989 as a refreshing way to learn Math and develop a fondness for mathematics (Anjo et al., 2005). The concept was simple but thorough: provide the students an interactive interface with mathematical quizzes and combining competitiveness with a gamified application – levels, lives, countdown timer. While "playing", the students would forget that the "game" was based on Math, one of the least popular science ever (Bhardwa, 2018; Flick and Lederman, 2003). The objective was to learn while playing, which was a hit. As an academic project born in a university, the application was always free to use, both for teachers and students (Pmate, 2019). Sponsorships were only used on main events to give students prizes.

The web application would be available for training during the whole year, culminating on a large physical event at the University of Aveiro, where schools and students would compete for top places. As the expectations evolved, the project has grown large in the last decade to an average daily use of 5000 users and a competition average above 7000 contestants. **Table 1** shows the statistics of the CNC@UA, the largest competition, for the last 5 years.

Table 1. Five-year stats of PmatE CNC@UA competitions (de Lucia et al., 2001)

Year	# of Schools	# of Teams	# of Students	# of Games
2016	204	4482	7308	5850
2017	195	5263	8468	7261
2018	190	6411	8912	8474
2019	192	6122	8565	8076
2020	Pandemic year			
2021	120	4183	4183	5116

From its creation to nowadays, the system has grown in both size and features with the existing technologies, responding to the public demand. Nonetheless the system was built in a time when hardware capacity was far more expensive than it is today and, consequently, efficiency took priority over a maintainable and upgradable system. This brought inevitable consequences in terms of system degradation, also caused by poor architecture and documentation which inevitably increases maintenance costs (Sommerville, 2016).

In 2019 the PmatE framework began to sob due to reduced personnel, low budget and lack of maintenance. The nationwide recognition, the project brand and the 3-decade effort made the coordinators assume a reinventing strategy: there was an effective need to make a major change and to evaluate of the project. There was a perception of intrinsic high value of the project, but it should be formally analyzed. The matrix proposed by (Sommerville, 2016) in **Figure 1** was used to evaluate the system and its true value chain, leading to a result high value/low quality.

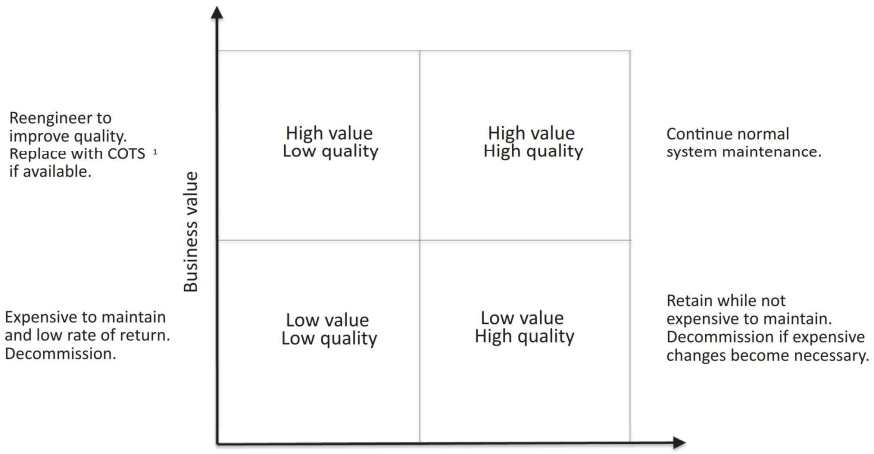

Figure 1. Legacy system decisional matrix (Alkazemi et al., 2013; de Lucia et al., 2001; Ransom et al., 1998), (Newby, 1994)

Low quality means expensive to maintain: these systems should be replaced, if suitable off-the-shelf systems are available, or reengineered to improve their quality. Making a decision about evolving a legacy system should be able to ensure an overall improvement of the quality of the legacy system, and the reduction of its maintenance costs. Nevertheless, the high value for both students and University of Aveiro, on education and marketing, drives the continuity of the project and the system reengineering and adaptability to achieve quality requirements.

In this paper, we present a case study within a university, showing how to evaluate the state of a legacy system in an academic organization and how to implement a reengineered solution that can deliver a quality product with extendable architecture and low maintenance. The paper is organized in eight sections: section Related Work presents similar work, section PmatE Framework describes the existing system's logic and architecture followed by section Analysis and Evaluation that uses a methodology to assess the system; section Implementation describes the tasks' development, followed by section Results that compilate data and gains. Section Further Work enumeratee later work to do, while section Conclusions wraps the achievements and results.

RELATED WORK

The IT industry has always pursued a way minimize the cost of maintaining legacy systems. Several assessment models, like (Newby, 1994), have been proposed and implemented, but because of the intrinsic private nature, closed source or commercial, most are not accessible or viewable. In the end, academic works are more visible and usually provide some details on published work.

Regarding software evaluation and maintenance, there are some case studies available, like the Umm Al-Qura University

that researched for a solution to cope with the changing environment without interrupting the routine working activities (Alkazemi et al., 2013). The case study uses a weighted decision-making grid to elaborate future tasks but does not implement any change; later publications from same authors only refer to framework process and strategies (Alkazemi, 2014). Another is the case study that uses automated analysis to identify dependencies and refactor code (Bavota et al., 2014). The study is interesting because of the large application range but keeps focus on programming techniques. Authors in (Griswold and Notkin, 1993) propose and implement a model and a tool to restructure code in an error-free transformation; it concludes that it should need evidence of cost-effective advantages from real-case scenarios. On (Ahmad et al., 2021) we can read about research to modernize legacy software, with a very special focus on mobile application; authors present a research method, a transformation process and an application evaluation.

As for educational games, a very recent study (The and Usagawa, 2018) analyzes the effectiveness of online quizzes, comparing traditional tests with virtual tests made on Moodle platform(Moodle Pty Ltd, 2021); it concludes elevated attentiveness and higher achievements for online tests but does not focus on quiz gamifying. Another interesting work focus on online flip learning with gamification quiz (Zainuddin et al., 2021); the study presents evidence of high engagement for the pandemic era, based on surveys and interviews, but does not show how it was implemented or architecture.

THE PmatE FRAMEWORK

The PmatE framework was initially a simple terminal application, written in C. Back in the early 90's that was a novelty that worked well with the initial Math quizzes. It had a simple UI with questions, each one with four true/false options and a sense of level progress. At the time it was an excellent start for the project's objective, it moved the students as they enjoyed "playing" with Math.

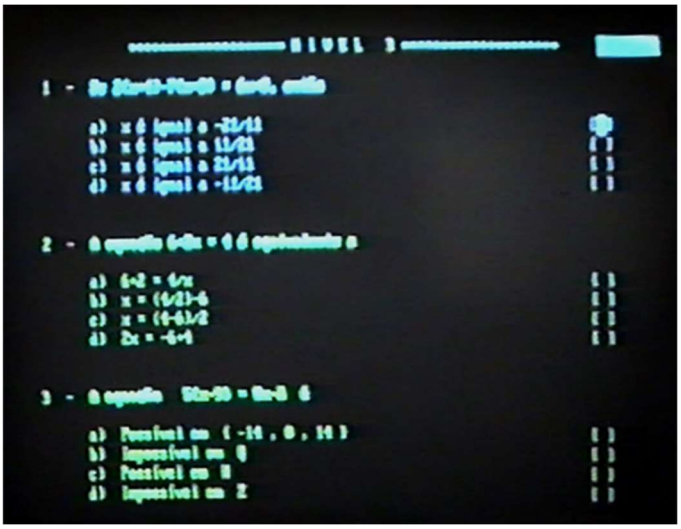

Figure 1. 1990 terminal application

The next step was to make it available across the campus network, online. So, a new application was developed in ASP and VB Script, supporting Internet Explorer. In the early 90's it was a breakthrough and local campus competitions became famous and desirable: 2000 students from the whole country gathered annually to find the best - student, team or school - in Math games.

A few years later it was patched with a few add-ons, especially with Adobe Flash to support Math symbols and formulas. As the years went by, it had great acceptance and started to grow to meet demands: more games, more templates for games, more school years involved and more courses rather than Math itself. In 2020 it held 14 main quiz categories from the 3rd to the 12th grade: from Mathematics to Biology, passing through Portuguese and English but algo Physics and Chemistry and so on. In the last years it also was extended to assist higher education classes, supporting Calculus assessments for instance. A module to build diagnostic tests was also developed, allowing teachers to retrieve important information at the start of the academic year to level-up students with their knowledge.

The registration, participation in competitions and access to results have been cost-free from the project's beginning. After registration, each student was enrolled in a class by the corresponding teacher and had access to all categories on his/her year but also all categories below their age. The training quizzes are available all-year long and are parametrized models filled with random values at the time of quiz request, generating uncountable different questions/answers. The interactive interface encloses a 4-question quiz game-like application – levels, lives, countdown timer and four pairs of True/False radio buttons, as shown in **Figure 3**.

Figure 3. 2019 quiz screen sample

After finishing the levels, or the time is up, the quiz ends and the user gets back to his/her personal area. There he/she can access to previous results and personal statistics. It seems quite simple, but inside the hood the complexity grows disproportionately.

Architecture

Disregarding the early years from the terminal C application, the actual PmatE platform started as a small ASP web application with a small webserver. Simple client-server calls, low usage and an adequate database where the questions were saved, were enough for the time-being.

As the years went by, users started to ask for more academic range and more and diversified content - it grew in

size and features, driven by users' requests. The manpower was always constituted by students or fresh-graduates' students with scholarships, which did not contribute to a good planning or structuring. The other problem was that after each development cycle, usually from 1-2 years, developers moved along to other challenges and the next staff would start with no solid ground. Each demand outputted a software islet that was built with the technology trending at that time. No updates were made to the existing software, leading to a kind of software patchwork. Every project, except for admin back-office, lies on a version control server Microsoft TFS, which allows to make version control, with some small updates or changes.

After 3 decades, it led to a complex framework, constituted of 6 applications and 1 database – with several instances – as shown on **Figure 4**.

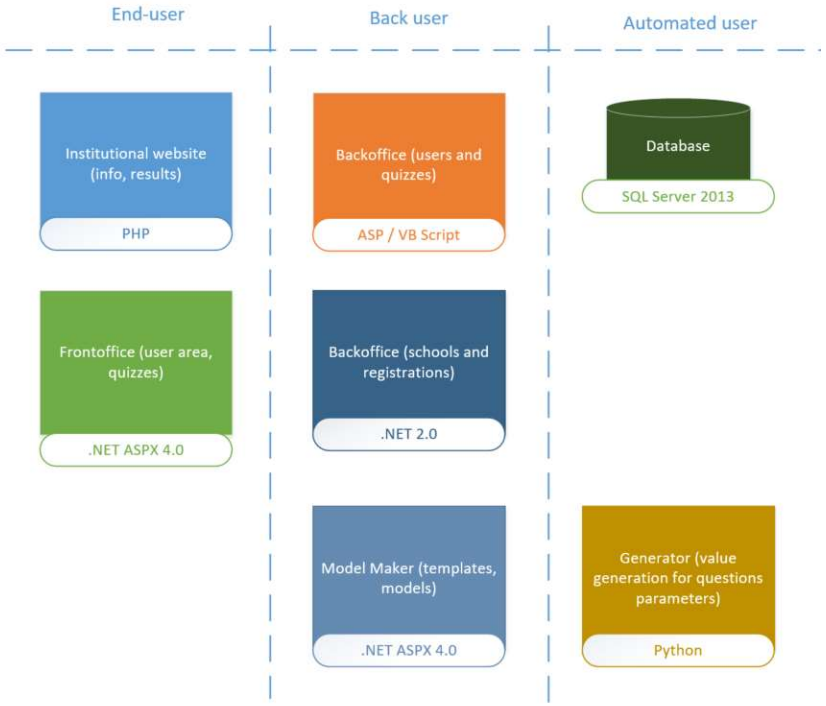

Figure 4. PmatE framework architecture

The end-user has access to the latest news and his personal area, being able to see results, statistics, training and competing without changing URL address, hand-over between PHP and .NET applications is fulfilled with simple HTTP redirects.

At the back-office, the administrators can manage all kinds of user data and quiz results but must change the URL to switch application and register schools and new students; they also must switch to another platform, ModelMaker, to develop parameterized quizzes and manage areas/categories/target years.

All the data stays persistent over a SQL database (2013), holding dozens of instances from decades; the latest backup had over 10GB and some queries would take more than 2 minutes. There is also a Python daemon that produces random values to fill parameters: on the front-office when a quiz is generated, a random model is chosen from the student's category and each parameter variable (question and/or answers) is sent to the Python daemon to be rendered with real values within the provided range, as shown on **Figure 5**.

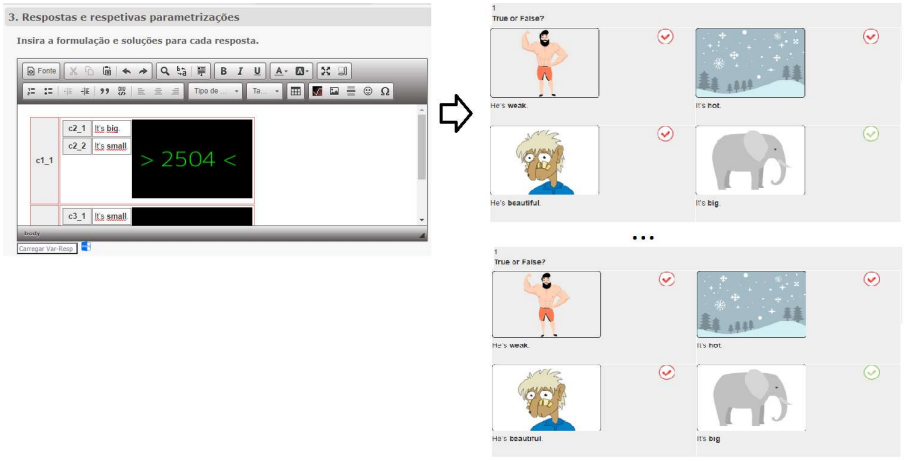

Figure 2. Parameterized quiz template and 2 template concretizations

The framework works reasonably in daily use, if we look at the upside – several technologies, most of them obsolete, exchanging data through 7 software blocks. The problem is that the lack of a coherent, united strategy to follow was decisive to increase maintenance over the years. So many systems, with many loose-coupled connections, become completely unsustainable. As a simple example, creating a new competition for 2020 could take a whole day – it requires a programmer to create four new SQL DB instances with new competitions' IDs and associate IDs to correspondent models manually. Another example is the retrieval results after the annual competitions: SQL queries running on stored procedures can take over 8 minutes for each of the 14 sub-categories. All results are then outputted as HTML to the results public page on the institutional website. Too much work, with no spare human resources or budget and no chance to put the framework offline - about 2000 users are online daily and an average of approximately 7000 on competition days.

According to (Berander and Andrews, 2005), a possible approach is to correct software faults, make it adaptive to new environments, enhance and improve efficiency and to make it preventative, by making structural changes and subsequent maintenance easier. With this in mind, the objectives of this work were then defined in 4 main goals:

A. Evolve the framework to be maintenance free
B. Speed-up results' data retrieval
C. Enhance user interface and design
D. Cleanup unnecessary software, addons and databases

The next step was to go through each goal, elicitate the problems and find a solution for it.

ANALYSIS AND EVALUATION

We utilized the framework described earlier to assess the major tasks in order to build a priority table of the current state of the system. The different contexts, together with their corresponding factors and statuses were taken in account. The objective was to identify, categorize and propose improvements. The interpretation of the values obtained out of the PmatE are given in a column called "Category" that can defines four levels of importance from A to D. These levels describe the impact of the obtained values on business continuity and system adaptability from the business perspective requirements. According to the four goals defined in chapter 3, they were also split to fit the categorization. The results of the analysis can be seen on **Table 2**.

Table 2. Framework problems

#	Problem	Solution	Category
RP1	Scattered software across several servers	Merge software on a single server with redundance and backup reinitialization	A
RP2	Obsolete software obstructing server technology	Develop modular and extensible new software with state-of-the-art programming languages	A
RP3	New academic years require hard code operations	Refactor code and build automatic stored procedures to update automatically	A
RP4	Retrieval of results take too long	Reengineer DB, build indexes and optimize SQL queries	B
RP5	HTML output results require manual file insertion and editing	Refactor code to translate SQL tables into HTML and insert a generate button for the user	B
RP6	Obsolete Quiz UI: not user-friendly, hazard colors and fit only to 640x480 resolution	Develop new friendly UI, with responsive behavior to fit all devices, including mobile	C
RP7	Obsolete front and back-office	Develop a new consistent and coherent design transversal to all software	C
RP8	Obsolete plugins, addons and webservices: Flash and VB Script	Identify all necessary and remove remaining; replace by state-of-the-art Javascript libraries	D

After the problems were identified, and as manpower was scarce, they were prioritized to allow incremental development sprints and show progressive enhancement on the platform. To achieve this, we used the combined prioritization technique referred by (Berander and Andrews, 2005), where requirements are mixed among stakeholders – in our case Product and Project Managers. The result of that prioritization is described below on **Table 3.**

Table 3. Prioritization Results. Priority, P(RPX) = RPCX × WProductM + RPCX × WProjectdM where RP is the requirement priority, and W is the weight of the stakeholder

#	Problem	Product Manager (0.60)	Project Manager (0.40)	Priority
RP1	Scattered software across several servers	0.10	0.15	0.12
RP2	Obsolete software obstructing server technology	0.10	0.15	0.12
RP3	New academic years require hard code operations	0.25	0.15	0.21
RP4	Retrieval of results take too long	0.10	0.15	0.12
RP5	HTML output results require manual file insertion and editing	0.10	0.10	0.10
RP6	Obsolete Quiz UI: not user-friendly, hazard colors and fit only to 640x480 resolution	0.25	0.10	0.19
RP7	Obsolete front and back-office	0.00	0.10	0.04
RP8	Obsolete plugins, addons and webservices: Flash and VB Script	0.10	0.10	0.10
	Total	1	1	1

After prioritization, the next phase was to implement changes according to identified significance. The focus would be on problems RP6 and RP3.

IMPLEMENTATION

Following the requirements' priority, one by one we started solving the problems. The first step was the problem's diagnosis, then develop solutions to solve them and implement changes. On this very first phase RP1, RP2, RP7 and RP8 were left on hold.

RP3 – New academic year creation

Expectations were that there could be a page on the back-office to create a new academic year. This was nonexistent, though. The standard procedure was to:

A. Create new ids for the 14 competitions and sub competitions in the DB;

B. Clone the 5 database instances (shown on Figure 6);

C. Go to the DB instance of the current year and update competition IDs;

D. Open webform codefile, comment 400 lines of code and copy them to new lines and replace IDs;

E. Update the front-office application to create new quizzes using new IDs and new academic year.

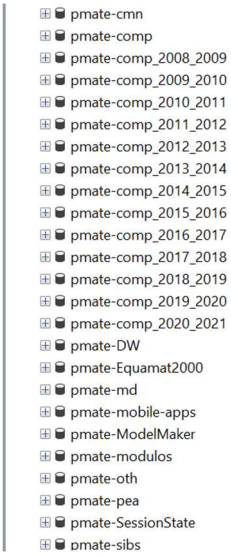

Figure 3. SQL Server DB instances

The most useful procedure would be a webform on the back-office with listings (competition, years, etc.) and a button to allow the creation and update of all variables. Based on that idea, we created that webform and linked all actions to it, so it would trigger everything without editing a single line of code. Before that, we changed the DB in order to avoid instance creations every year; we needed to adapt and insert new columns into tables and apply it on the C# code-behind. The webform did the following tasks:

A. Automate database instantiation with proper stored procedures and triggers;

B. Generate competition IDs and attach them to relational tables;

C. Refactor code to reduce cycles and optimize it;

D. Update front-office application to allow year choosing.

These modifications were made and resulted in an extremely practical - as well as transparent - approach to back-office users, which can be seen on **Figure 7**.

Figure 4. Automated launch of new Academic Year

RP6 – Obsolete Quiz UI + RP8 – Obsolete plugins

The ASPX webform was full of unnecessary things: Flash plugins, old jQuery libraries and old ASP.NET controls. It also had huge stateviews behind the page and was only loaded with session data, retrieved from the user login and DB.

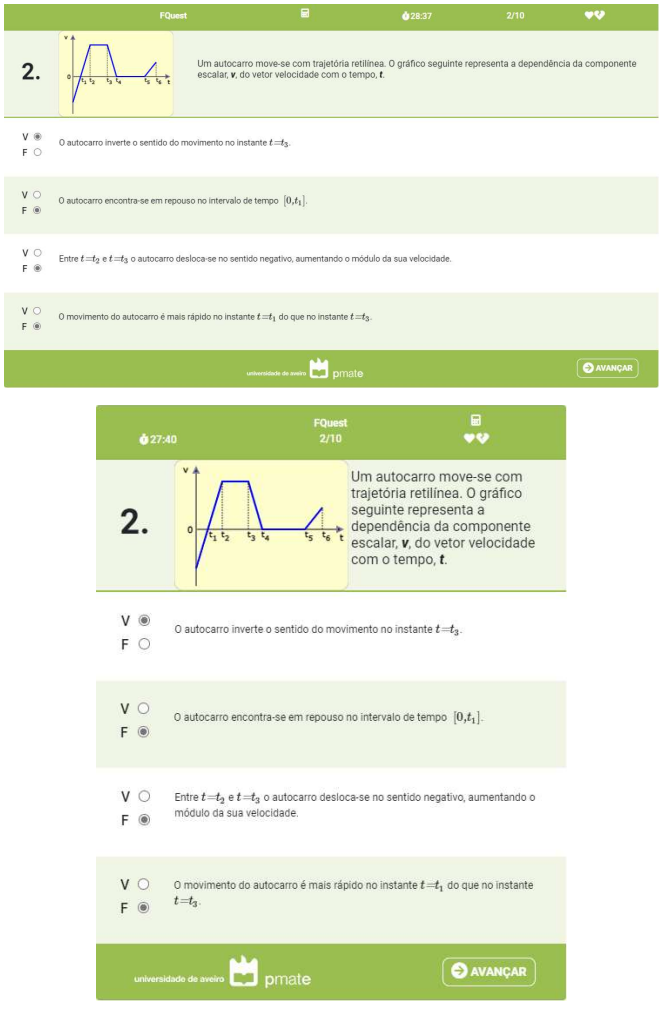

Figure 8. New quiz responsive UI

It simplified design, added responsive elements that fit any screen and a more intuitive and better user interface.

RP4 – Retrieval of results take too long

The performance problem that existed on the framework was very clear – web forms that would return some kind of result could last from 2 to 8 minutes to run, depending on the query. Considering the scale, 7-10K students, it may not seem particularly high, but the concern emerged. To solve the problem, we started to call the queries directly on the SQL DB, to discard network or connection problems. The results were astonishing, the problem was indeed with the SQL queries. We ran the SQL Live Query Statistics and the results are shown on **Figure 9**.

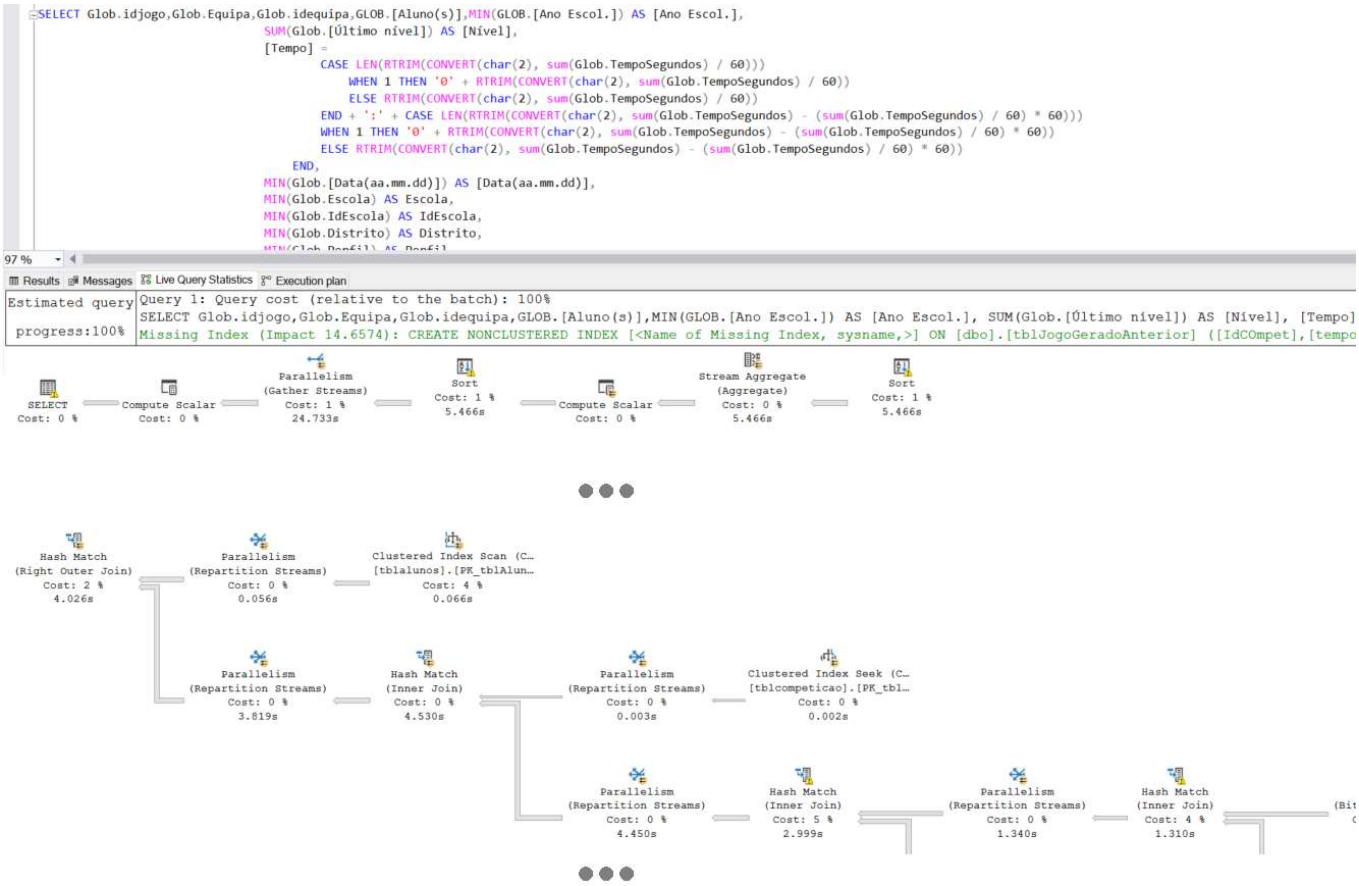

Figure 5. SQL Query Analyzer on PmatE query

Besides an erroneous construction of the queries, with unnecessary columns or table joins, the DB was not indexed. The first step was to correct queries with solely the necessary column data and table joins. This improved results and increased performance, elapsed time remained between 10-20 seconds, depending on the query. We then created indexed views of the tables from which we expected results. Here lies the purpose, because the results can only be retrieved when competitions are over, which means the view should be one view of final and consolidated results. We created a trigger to create a view of the results' table, like the one shown on **Figure 10**, and automatically create indexation.

```
CREATE TRIGGER trgTableAdded ON DATABASE AFTER CREATE_TABLE AS
BEGIN
    SET NOCOUNT ON
    DECLARE @TableName SYSNAME, @SchemaName SYSNAME, @StringToExecute NVARCHAR(4000)
    SELECT @TableName = EVENTDATA().value('(/EVENT_INSTANCE/ObjectName)[1]','SYSNAME')
    SELECT @SchemaName = EVENTDATA().value('(/EVENT_INSTANCE/SchemaName)[1]','SYSNAME')
    IF @TableName LIKE 'ResultsExport%'
        AND EXISTS (SELECT * FROM INFORMATION_SCHEMA.COLUMNS
            WHERE COLUMN_NAME = 'Results' AND TABLE_SCHEMA = @SchemaName AND TABLE_NAME = @TableName)
    BEGIN
        SET @StringToExecute = 'CREATE INDEX IX_Results ON ' + QUOTENAME(@SchemaName) + '.' + QUOTENAME(@TableName) + ' (Results);'
        EXEC(@StringToExecute);
    END
END
GO
```

Figure 6. SQL trigger to create automatic indexation

This resulted in queries with elapsed time between 2-12 seconds, which sounded quite better than 1-3 minutes.

RP5 – HTML output results require manual insertion

Upon the end of competitions, the results were extracted from the DB with SQL queries like the ones referred before on 5.3. Even though they were retrieved with HTML tags, each query rendered a HTML output which should be copied and saved on a PHP server, which would take a manual effort of hours. This was corrected to a page that would call the stored

procedure and save HTML input to the correct folder, with IIS permissions. If some results would require revision (i.e. some error on some quiz) it would be enough to delete the file from the server and access the page again to refresh the output. Again, this resulted in reasonable time savings and avoided IT personnel to handle the results. **Figure 11** shows an example of the output.

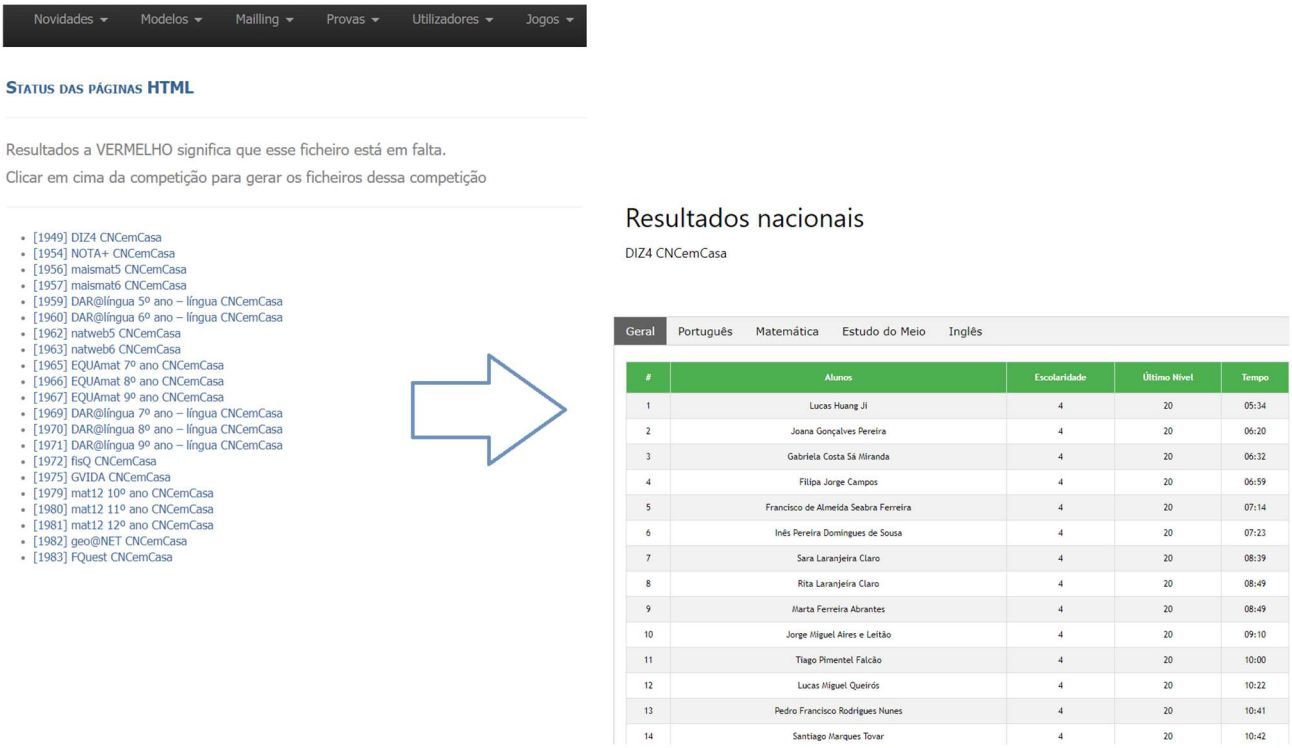

Figure 11. Generating HTML results with one-click button

RESULTS

The outcome of the task's implementation was evaluated both in efficiency and usability, depending on the requirement. This section presents a compilation of the effects.

For time-consuming tasks, we measured the current times against older one, although 1-day task for a programmer was an estimate

Table 3. Implemented RP efficiency

#	Problem	Initial Time	Current Time	Time Gain	Ratio
RP3	Opening new academic years	6:00:00	0:01:50	5:58:10	19636%
RP4	Retrieval of results for students	0:02:00	0:00:10	0:01:50	1200%
RP4	Retrieval of results for schools	0:08:00	0:00:20	0:07:40	2400%
RP5	HTML output results require manual file insertion and editing	4:00:00	0:00:40	3:59:20	36000%

For the usability assessment, we made a small survey with 5 questions on a 20 user set – we aim to make a survey on the end of each competition to broad validate the results with a mode adequate scale.

#	Problem	Initial UI	Current UI	Gain	Ratio
RP6	Obsolete Quiz UI: not user-friendly, hazard colors and fit only to 640x480 resolution	44%	92%	48%	209%

The main goal of an algorithm is to produce correct output, and the system was producing it; the main achievement we aimed was efficiency, which was extremely optimized with the best practices on the several systems.

FUTURE WORK

Due to time, manpower and budget constraints, some requirements were still pending. In this case, the centering of the software on a single server, with backup in case of fallback of course (RP1), the whole refactoring and updating of old

technologies (RP2) and the renewal of the entire back-office (RP7), written in ASP and .NET 2.0 did not go through.

RP1 was a priority but the big scale of the systems' involved were difficult to gather and reduce in such a short time. It should be a priority and should be considered to develop with new technologies.

The RP2 task was also not completed due to time constraints and to IT services technology lockdown. It was not possible to assemble a new architecture with state-of-the art technologies while implementing the other requirements.

As for RP7, as it was not a priority to stakeholders, it will be a task for the next development iteration. Still, it needs some SQL refurbish also because the pages' content holds direct SQL queries, as we can see on **Figure 12** below.

```
'Função drawMain
'-------------------------------------------------------------------------------
sub drawMain
dim idescola,iduser,sql,Conn,rs
iduser=Request.QueryString("iduser")
Set Conn=Server.CreateObject("ADODB.Connection")
Conn.Open strConn
set rsAux = server.CreateObject("ADODB.Recordset")
    rsAux.CursorType = 3
sql="select nome,nomeescola from tblusers u inner join tblescolas e on e.idescola=u.idescola where iduser="&iduser
set rs = server.CreateObject("ADODB.Recordset")
rs.CursorType = 3
rs.Open sql, Conn' ,3,3


'encontrar o ano lectivo actual
set rsAL = server.CreateObject("ADODB.Recordset")
    rsAL.CursorType = 3
dim sqlstrAL
sqlstrAL="SELECT descricao FROM dbo.NS_anoLectivo (nolock) "&_
        " WHERE (inicio <= CONVERT(DATETIME, '"&Date()&" 00:00:00', 103)) AND (fim >= CONVERT(DATETIME, '"&Date()&"  00:00:00', 103)) "
dim     esteANOlectivo
rsAL.Open sqlstrAL, Conn ',3,3
'variável que guardar o valor do ano lectivo actual
esteANOlectivo=rsAL.fields(0)
rsAL.close
%>
<h1>mudar de escola</h1>

    <p align="right"><a href="JavaScript:history.go(-1)">voltar</a></p>
```

Figure 7. Example of direct SQL queries: mudarescola.asp file

The best practices indicate that they should be transformed in transact-ready queries and isolated in stored procedures, which will save a lot in maintenance but also can speed up tasks.

One ultimate task to fulfill should be a user evaluation test: gather a significant group of users and test usability and satisfaction. This would raise the level of the application to the highest standards.

CONCLUSIONS

A legacy application rarely can be optimized without significant effort. It is often involves hard and complex work that is not visible. Thus, a legacy system has a different lifecycle where decision processes must be carried out carefully in order to support the successful management of the system. These processes are usually analyzed in an implicit way, but when the complexity of the problem is not common, one must apply a systematic approach for defining which modifications should be made or recommended.

This paper presented a use case of how to evolve a legacy system in order to maintain its intrinsic high value. Its model was assessed, analyzed and then it was refurbished. The business goals of the legacy system were defined by taking into account the points of view of different stakeholders within the organization. The framework interventions were carried-out on critical software components, thus providing a conversion result brought user usability, controlled costs and easier maintenance for the coming years. Even though it lacks a larger scale validation, with end-user tests, it augmented user friendliness, made data and interface loadings faster for all users as well as saved servers' resources.

The use case provides a reference model that shows a potential solution to legacy systems management, and more specifically how address them by taking into account both business and technical issues. Furthermore, models, techniques, and tools to support the framework instantiation were presented and applied.

As this was the very first phase of the renewal, we are certain that continuing the best practices described in the current literature will result in a systematic increase of framework quality and value. In the second phase of the project we will address the remaining problems.

Nevertheless, the experiment was validated with excellent results. The conclusive results of the development will be presented in future work.

REFERENCES

Ahmad, A., Alkhalil, A., Altamimi, A.B., Sultan, K. and Khan, W. (2021). Modernizing Legacy Software as Context—Sensitive and Portable Mobile-Enabled Application. IT Professional, 23(1), pp.42-50.

Alkazemi, B., (2014). A Framework to Assess Legacy Software Systems. J. Softw., 9(1), pp.111-115.

Alkazemi, B.Y., Nour, M.K., Meelud, A.Q. (2013). Towards a Framework to Assess Legacy Systems, in: 2013 IEEE International Conference on Systems, Man, and Cybernetics. pp. 924–928.

Anjo, A., Pinto, J.S., Oliveira, M.P., Isidro, R.O.G. and Pais, S.I.V. (2005). Computerized Diagnostic Test. Cadernos de

Matemática, 5(3).
Bavota, G., Gethers, M., Oliveto, R., Poshyvanyk, D., Lucia, A. de, (2014). ACM Trans. Softw. Eng. Methodol. 23.
Bennett, K., (1995). Legacy systems: Coping with success. IEEE software, 12(1), pp.19-23.
Berander, P., Andrews, A., (2005). Requirements Prioritization, in: Aurum, A., Wohlin, C. (Eds.), Engineering and Managing Software Requirements. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 69–94.
Bhardwa, S., (2018). The least and most popular undergraduate courses in the UK [WWW Document]. Times Higher Education. URL https://www.timeshighereducation.com/student/news/least-and-most-popular-undergraduate-courses-uk (accessed 12.1.20).
Flick, L.B., Lederman, N.G. (2003). School Science and Mathematics 103, 117–120.
Griswold, W.G., Notkin, D. (1993). ACM Trans. Softw. Eng. Methodol. 2, 228–269.
de Lucia, A., Fasolino, A.R., Pompelle, E. (2001). A decisional framework for legacy system management, in: Proceedings IEEE International Conference on Software Maintenance. ICSM 2001. pp. 642–651.
Moodle Pty Ltd, (2021). Moodle - Open-source learning platform | Moodle.org [WWW Document].
Newby, M. (1994). Proceedings Software Education Conference (SRIG-ET'94) 96–102.
Pinto, J.S., Oliveira, M.P., Anjo, A.B., Pais, S.I.V., Isidro, R.O., Silva, M.H. (2007). International Journal of Mathematical Education in Science and Technology 38, 283–299.
Pmate, (2019). Relatório Competições Nacionais de Ciência 2019 [WWW Document]. URL https://pmate.ua.pt/pmate/relatorios/2019/relatorio_CNC_2019_PmatE_UA.pdf (accessed 9.21.19).
Ransom, J., Sommerville, I., Warren, I. (1998). A method for assessing legacy systems for evolution. In Proceedings of the Second Euromicro Conference on Software Maintenance and Reengineering (pp. 128-134). IEEE.
Sommerville, I. (2016). Software Engineering GE. Pearson Australia Pty Limited.
The, M.M., Usagawa, T. (2018). International Journal of Emerging Technologies in Learning (iJET) 13, 157–176.
Zainuddin, Z., Farida, R., Keumala, C.M., Kurniawan, R., Iskandar, H. (2021). Interactive Technology and Smart Education ahead-of-print.