

SYSTEM FOR EFFECTIVE SMALL BUSINESS SUPPORT

Volodymyr Pavlenko, Oksana Lashko

Lviv Polytechnic National University, 12, Bandera Str, Lviv, 79013, Ukraine.

Author's e-mail: volodymyr.pavlenko.ki.2017@lpnu.ua

https://doi.org/10.23939/acps2021.01.____

Submitted on 05.05.2021

Abstract – This paper considers the problem of developing specialized software designed to support small businesses. It substantiates the relevance of creating such systems; architecture has been offered; and the results of development have been given. For practical use, a specific subject area has been considered, which allows to clearly understand the purpose and outcome of the work. These materials can be used to obtain ready-made solutions during the development of a software package on this topic. This document can be considered as an introductory material for the various stages of the project to develop a system of effective support of small business.

Index Terms: software development management, system modeling, business data processing, reconfigurable and self-configurable computer systems, data analysis, client-server systems

I. INTRODUCTION

To date, the use of specialized software in business has become widespread.

Programs designed for employees of companies have the ability to perform a large number of routine tasks [1]. However, modern Ukrainian entrepreneurs have a very limited set of tools to use.

In particular, there are very few programs that can help maintain a database of goods, revenues and sales, and in most cases, they are too cumbersome (1C Accounting) for use by an entrepreneur who owns a small or medium business. And there are no software systems that could be used as a single integrated system to work with any device (phone, PC or browser).

There is also a problem of lack of a special set of programs (windows, android, web) that could automate the routine work of modern employees, and help them qualitatively assess the current state of the enterprise by considering the calculated accounting, economic analysis displayed in a convenient form on their work devices. (telephones, PCs, etc.).

II. TECHNICAL TASK

Create a program for effective support of small business on the topic of "Auto Parts Store", which should

support the work with the documents of current legislation and conveniently organize the working space of the employee [2].

The software package must be able to work from many devices simultaneously using a common database.

The speed of the server part must be from 3,000 requests / hour.

The program must meet the minimum requirements for memory up to 500 MB and disk capacity up to 5 GB.

A. A. BUSINESS RULES

Every year (01.01.xx 00:00) document numbering must start from the beginning [3]. (eg "Invoice №1")

The program must have special functionality for working with documents.

Types of documents [4]:

- Score
- Sales Invoice
- Receipt
- Tax invoices
- Invoices for return
- Invoice for receipt

The required statistics to be provided by the software include:

- Gross income,
- Top 10 buyers by their costs.

B. B. MAIN FUNCTIONALITY

The main functionality of the program is based on user needs. Before developing, it is important to research the subject of the program in detail.

One of the main tasks of software design is to form a list of functionalities required by users. The list should contain all the important functions for working with data.

It is very important to pay attention to all the needs of users, because fixing old features or adding new features is an expensive thing in the later stages. That's why programmer should pay attention to them.

Functionality of the program according to the theme "Auto Parts Store":

- 1) Automatic generation of new product code (to be used as a barcode).
- 2) Download the supplier's price list.
- 3) Add, edit and delete documents.
- 4) Special numbering for each type.
- 5) Support for different types of sales: invoices, expense invoices, return invoices and tax invoices.
- 6) Formation of special forms on the basis of types of orders (html for printing, xml).
- 7) Analytics.
- 8) Inventory.
- 9) Authorization, registration.
- 10) Existence of administrators (privileged users).
- 11) Logging of performed actions by execution time.
- 12) Print created html-forms of documents.
- 13) Printing of labels with barcodes. Must contain the price and name of the product.
- 14) Scan generated barcodes.

III. ARCHITECTURE

The small business support system must work simultaneously with one database on different devices, so it will be advisable to build an architecture of "client-server" type[5].

C. A. CLIENT

The client program consists of modules `MainWindow`, `TcpClient` and many classes of widgets and dialogs (Fig. 1):

- **MainWindow** - acts as a control center and is the main graphical object that will contain elements such as widgets.
- **TcpClient** - is responsible for communication with the server. Is a field of the `MainWindow` class and is initialized in its constructor. It implements basic public methods through which other classes will interact with the server.
- **Widgets** - a type of classes that will play a major functional role.
- **Dialogs** are a type of class that is responsible for modular, pop-up windows. Very often these windows play the role of a dynamic application configurator.

The `MainWindow` and `TcpClient` are in a single instance while there are many dialogs and widgets.

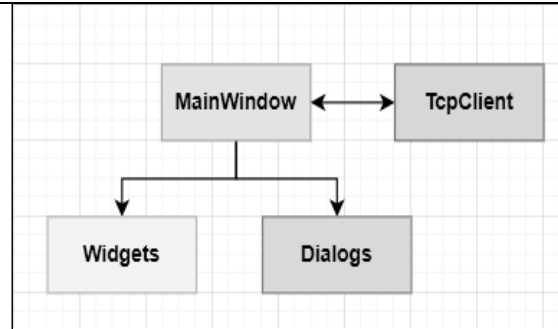


Fig. 1. The structure of the client part. `MainWindow` and `TcpClient` interact with each other and transfer information between the GUI and the server. `Widgets` and `dialogs` are responsible for interacting with the user

D. B. SERVER

It is important to have a good understanding of exactly how the client and server will interact[6]. The server part consists of 3 main modules (Fig. 2), namely:

- `MainServer` - the main class responsible for managing other modules and interacting with clients,
- `XmlParser` is responsible for working with xml-data files,
- `SQLiteHelper` is responsible for working with the SQLite database and performs basic computational tasks using the query language.

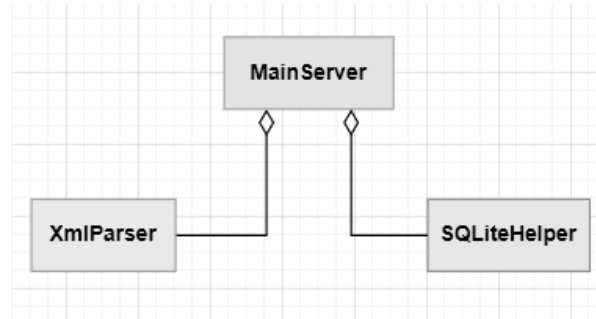


Fig. 2. The structure of the server part. The `XmlParser` and `SQLiteHelper` modules are part of the `MainServer`, but they are also completely independent parts of the code and are used only as separate objects

The `SQLiteHelper` module interacts with the SQLite database. SQLite is an open source embedded relational database.[7] The database has a specially designed application structure (Fig. 3). Tables are linked through primary and secondary keys.

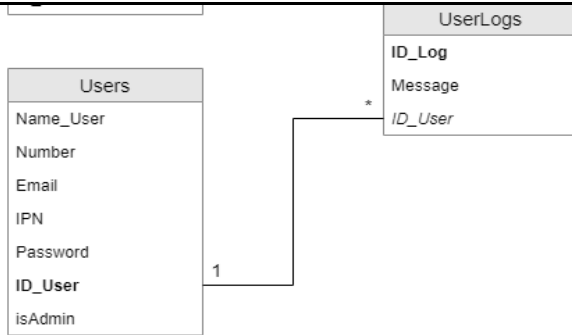


Fig. 3. Example of the structure of relationships between tables in a database. Primary keys are highlighted in bold and secondary keys in italics.

Each table has its own special purpose (Table 1) and plays an important role in the whole program. The database is created immediately the first time you start the server on a new machine.

Documents have the characteristic ListType, which is actually an enum type, i.e., an example of a record in the database: "3" - Document type "3", which is a "Tax invoice". ListNumber is a unique (for the current year only) document identifier, according to the Business Rule on annual numbering updates.

Each list (from the Lists table) is, in fact, a document whose rules of conduct and types are listed in the Business Rules.

Relationships between tables can be either to each other or to one another.

Table 1

| DB tables and their purpose | |
|-----------------------------|---|
| Table name | Description |
| Lists | Keep a record of all the "Lists" (bills, invoices, etc.). |
| Records | Contain all positions of all lists |
| ProductTypes | Contain a list of all possible types of goods. |
| Users | Keep a record of all users of the program. |
| UserLogs | Keep a list of all users' activities. |
| Customers | Keep a record of all customers. |
| Cars | Each customer can have multiple cars. |
| Sellers | Keep a record of all sales. |
| Storage | It contains a list of all those present to "structure" (in store) products. |

Each table is related to others in a special way that implements a certain type of relationship:

- Lists and Records display lists and their records, respectively. One-to-many connection principle.
- RecordTypes and Records - "one-many"
- Users and UserLogs - "one-many"
- Lists and Customers - "many-one". One list can have only one buyer (seller).
- RecordTypes and Storage - "one-one"
- Customers and Cars - "one-many"

IV. DESIGNING

Software design begins during or immediately after the completion of architecture development.

Once the requirements are established, the design of the software can be established in a software design document. This involves a preliminary or high-level design of the main modules with an overall picture (such as a block diagram) of how the parts fit together.

The language, operating system, and hardware components should all be known at this time.

Then a detailed or low-level design is created, perhaps with prototyping as proof-of-concept or to firm up requirements.

One of the design stages can be considered detailed design, which is carried out by building class diagrams separately for the client (Fig. 4) and server (Fig. 5). During the detailed design, it is important to plan the connections between the classes.

The creation of business programs is more than a way to view or automate your information process.

E. A. DETAILED DESIGN

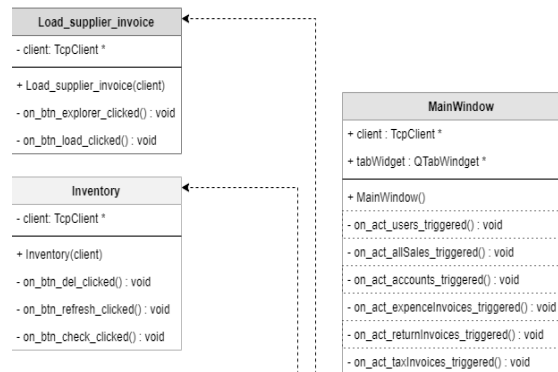


Fig. 4. Part of the client class diagram. The diagram shows the main class MainWindow and its connections to Load_supplier_invoice (dialog) and Inventory (widget)

In particular, relationships such as aggregation and composition should be indicated in the diagram and, to improve perception, they should be summed up in the appropriate field of the class to which they are associated. (Fig. 5).

The class diagrams are shown from the field and the methods of the server modules, from the program code, the objects of the type of classes.

The diagram depicts a large amount of low-root information, which is extremely useful for software developers.

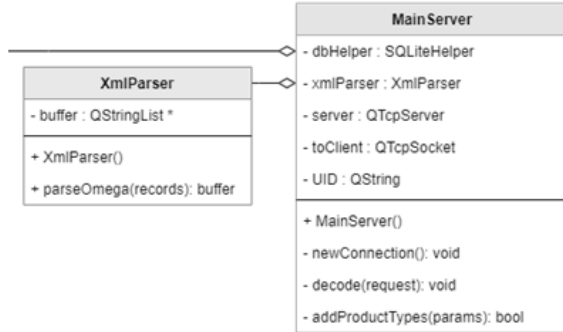


Fig. 5. Part of the server class diagram. The relationship between the XmlParser and MainServer classes. In this case, the first class is a field of the second class, which is an example of aggregation

The final stage of design in this project is the choice of how the client and server interact. As a basis for client-server interaction, it was decided to choose the most convenient way to understand "Request-Response". The server must respond a special message to each request from client.

In most cases, the server will be required to acknowledge receipt or processing of information. To log all possible types of commands, a special "request-response" table (Table 3) has been created, which contains all types (Table 2) of requests to the server and responses to them for the client.

Table 2

| All types of commands | |
|-----------------------|--|
| Commands | |
| Add | |
| Del | |
| Edit | |
| Get | |
| Login | |
| Uniq | |

Each type of team is specially designed to minimize their overall number. This is done to provide fast encoding and effective further code support.

It was decided to use the ":" symbol as a delimiter. By the way, choosing a different one or even using a group of characters is not a problem.

The table gives examples of each of the types of the commands in the appropriate order. The table contains: the name of the class, client's requests and server's responds.

Table 3

An example of each type of command

| GUI class | Request (client:[UID]:[command]:[params]) | Response |
|--------------------------|---|--|
| New_customer | client:[UID]:add:Customers:[name]:[iban]:[bank]:[edrpoj]:[ipn]:[address]:[number]:[email] | server:[bool] |
| Customers | client:[UID]:del:Customers:ID_Customer=[ID_Customer] | server:[bool] |
| New_customer | client:[UID]:edit:Customers:ID_Customer=[ID_User]:["column=value" / delimiter = " "] | server:[bool] |
| Customer_info | client:[UID]:get:Customer info:ID_Customer=[ID_Customer] | server:[Lists.DateTime + Records by ID_Customer] |
| Authorization | client:[UID]:login:[login]:[password] | server:[UID or -1]:[isAdmin] |
| Load_supplier_price_list | client:[UID]:uniq:addProductTypes:[supplier]:[2000 or less records] | server:[bool] |

F. B. CODING

After the detailed design, the coding stage begins, which is the conversion of the developed diagrams and algorithms into code in the selected programming language.

During this stage, the main method of software testing is selected and all items of the created technical task are implemented.

It was decided to implement the user interface of the **server part** in the form of a command line interface (CLI), because the end user will not in any way configure the server directly. It is assumed that any required settings of the server program will be performed using the graphical interface of the client program.

Qt framework provides a good library for developing CLI applications [8]. In particular, they allow to work directly with stdout and stderr streams. This allows to easily log information.

The information that will be output by the server program in stdout will be exclusively working information about:

- Starting the server,
- Network server address,
- Information about new connected client programs,
- Information about disabled client programs,

- Errors at work.

It was decided to implement the user interface (Fig. 6) with the **client part** using Qt GUI technologies [9]. The GUI model will be based on the terms of reference.

The main control is QMenuBar from the Qt GUI package. From this element the user will call the main modules of the system.

Each module is a separate QWidget that runs in the QTabWidget as a tab.

Interface structure:

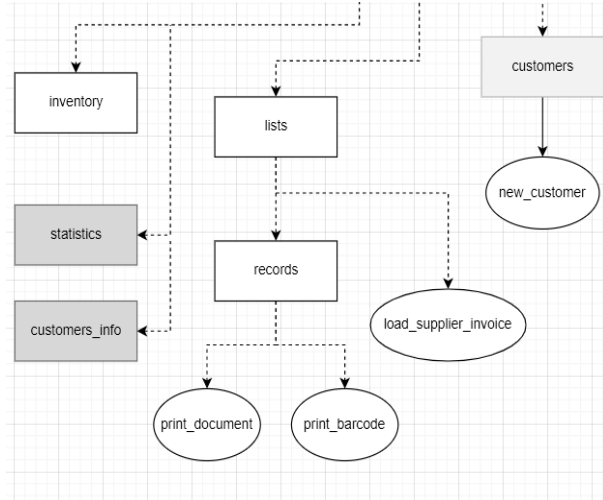


Fig. 6. Customer interface structure diagram. The figure shows all the transitions between different parts of the graphical interface. There are 2 types of possible graphic classes that will be called from the main window of the program: widget (rectangle), dialog (ellipse).

The interface is developed in Qt Creator [10].

G. C. TESTING

The testing phase (Fig. 7) is a fairly broad point and covers most of the other development stages.

Also, this stage may also cover the future phase of maintenance, i.e., after commissioning, when the tester is an employee of the enterprise.

The program is designed for the Ukrainian market so you can see the Ukrainian localization.

| Date and time | Number | Customer |
|-----------------------|--------|----------|
| 1 2020-12-27 09:40:25 | 1 | John |
| 2 2021-01-09 11:58:08 | 2 | Sara |

Fig. 7. Testing of work with the list of documents. You can see 2 records with end customers.

Each document has a list of things (Fig. 8). Therefore, double-click to edit the mode of a specific document.

During the main stage of testing, the main validation of the software for compliance with the technical task was performed. A lot of information was obtained that can be compared with the analogues of the system on the market.

| Code | Catalog | Name |
|--------------|-------------|----------|
| 1 1036404 | D4ABMEG50C | Good 1 |
| 2 1037464 | 546681CA4B | Good 31 |
| 3 1101045304 | 7405.100516 | Good 27 |
| 4 1101045305 | 740.100805 | Good 123 |

Fig. 8. Testing work with list of goods from the selected doc. List belongs to end customer and seller Olexander

Checking the correctness of the database update of all available product types (Fig. 9).

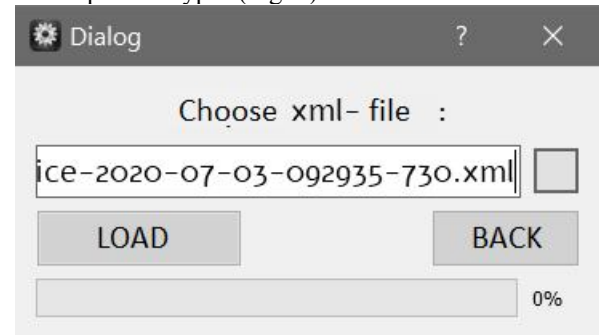


Fig. 9. Testing the work of updating the database of all types of goods. You should enter the path to xml-doc

After the completion of the testing and commissioning phase, the created system of effective small business support is a full-fledged software product and can be compared with other analogues. Let's single out architectural advantages:

- The developed system in comparison with the analogue of "1C: Enterprise" has only the necessary functionality for a small business employee. Due to the fact that the number of functions is much smaller, the maintenance of this software is cheaper.
- Unlike, for example, the analogue "Athena", the developed software supports multi-device operation and has sufficient functionality to work effectively with routine tasks.

Also, hardware requirements play an important role in such systems. For example, 1C: Enterprise has minimum requirements for hard disk (40 GB) and OP (1 GB). If the OP is less, the program will run extremely slowly. That is why the recommended requirements for 1C are at least 2 GB. Compared to the system developed in this bachelor's thesis, the difference is significant, because the created developed program uses only 500 MB of RAM.

Given the above reasons, it can be argued that the developed system in this project provides more efficient operations in a typical, small business environment.

V. MAINTENANCE

Maintenance is an important stage in the life cycle of any high-quality software that works with time-varying requirements.

If the product developer no longer supports the software installed by the client, the transition to a new one is problematic, and the support of the existing system is too expensive.

During each stage of software development, the developer must constantly check his work for errors and correct them in advance.

If there is a technical feasibility, reengineering is carried out - that is, the creation of a redesigned version of the old software taking into account the new requirements. Migration is gradual to avoid risks.

Preventative Software Maintenance helps to make changes and adaptations to your software so that it can work for a longer period of time. The focus of the type of maintenance is to prevent the deterioration of your software as it continues to adapt and change. These services can include optimizing code and updating documentation as needed.

Taking into account the reasons above, and also taking into account the type of software developed, it was decided to choose preventive software maintenance.

Preventative software maintenance helps to reduce the risk associated with operating software for a long time, helping it to become more stable, understandable, and maintainable.

For all businesses and organizations, software maintenance is an essential part of the software development lifecycle. This isn't something that one can skip or avoid.

It is absolutely necessary for the success of your software and any evolution into the future.

It is important to know that maintenance needs to go much further than fixing issues or bugs – that is only one steps of the software maintenance process.

Updating software environments, reducing deterioration, and enhancing what is already there to help satisfy the needs of all users are also included in the software maintenance examples.

In our application project with the theme "Auto Parts Store" we developed a program to support small businesses working with the legislation of Ukraine, which is constantly frequently.

As a result, the program needs to be frequently updated and modified.

The main most frequently changing parts of the software are modules that work with forms for printing documents. For other changes in the legislation, editing through the graphical interface of the program is already provided, which greatly facilitates maintenance.

VI. CONCLUSION

Therefore, this document discusses the main points of the process of developing specialized software to support small businesses.

During the time allotted for the implementation of the project, the way to develop business support systems was studied and performed. The work was carried out according to the numerical framework specified in the calendar plan.

This paper considers the main points of the process of developing specialized software to support small businesses. In particular, the main tasks were performed:

- requirements were received from end users and a technical task was created on their basis;
- the selection of optimal means for creating this type of software;
- developed software architecture;
- detailed design and coding;
- the created software was tested, namely its validation for compliance with the initial requirements.

A full-fledged software system for effective support of small business on the subject of "Auto Parts Store" was created, which supports the work with the documents of current legislation and conveniently organizes the working space of the enterprise. The created system has the following architectural advantages

- The resulting software has the ability to work from many devices.
- The program meets the requirements for memory and has a size of up to 500 MB.
- The disk size is less than 5 GB. The speed of the server part exceeds 3000 requests / hour.
- Due to the fact that the number of functions is much smaller, the maintenance of this software is cheaper.

After commissioning by end users, a sufficient number of positive feedback was received, which confirms the criterion of project success.

The developed software is effective when used in small enterprises, as it covers the main disadvantages of analogues in the market of business support programs:

- redundancy and high cost of maintenance,
- truncated functionality, which is not enough to effectively perform certain tasks
- inability to work with multiple devices simultaneously.

References

- [1] Spinellis, D., (2016). Managing a Software Business. *IEEE Software*, pp. 4-7.
- [2] Voas, J., (2004). Software Engineering's Role in Business. *IEEE Software*, pp. 26-27.
- [3] Kluza, K. and Nalepa, G., (2017). A method for generation and design of business processes with business rules. *Information and Software Technology*, pp. 123-141

-
- [4] Knowledge-Based Systems, (1988). Knowledge-based decision support in business: issues and solutions.
 - [5] Refactoring.guru. (2021). *Design Patterns*. Available at: <<https://refactoring.guru/design-patterns>> (Accessed 30 May 2021).
 - [6] Zhu, H., (2005). Software design methodology. *Oxford: Elsevier Butterworth-Heinemann*.
 - [7] Allen, G. and Owens, M., (2010). *The definitive guide to SQLite*. New York, NY: Apress. p. 24
 - [8] Qt Project. (2021). *Qt Examples And Tutorials*. Available at: <<https://doc.qt.io/qt-5/qtexamplesandtutorials.html>> (Accessed: 30 May 2021).
 - [9] Rischpater, R., (2013). *Application development with Qt Creator*. Birmingham: Packt Publ.
 - [10] Qt Project. *Qt Creator*. (2021). (Version 4.12).
 - [11] Thankappan, V., (1980). Small Business Observations, *American Journal of Small Business*. pp.1-2.

Pavlenko Volodymyr is a Junior C++ Qt Software developer. Nowadays he works as a freelancer.

From 2013 to 2017, he completed his secondary education at the Talne Lyceum of Mathematics and Economics.

After that he entered Lviv Polytechnic National University, where from 2017 to 2021 he received a bachelor's degree in "Computer Engineering".

Oksana Lashko was born in 1976 in Lviv, Ukraine. She received the B.S. and M.S. degree in Lviv Polytechnic State University, Lviv, in 1999. From 1999 to 2002, she was postgraduate at Lviv Polytechnic National University. Since 2007, she is a senior lecturer at the Computer Engineering Department of Lviv Polytechnic National University. Her research interests include the development of signal processing

tools at the algorithmic and software levels, research of image encoding and compression problems.

