

Vasily Babenko


Analysis and improvements of business  
processes with IT  
Development pipeline for small business  
software solutions

Bachelor's Thesis  
Informational technology

May 2016



## DESCRIPTION

		<b>Date of the bachelor's thesis</b>	
		30.05.2016	
<b>Author(s)</b>		<b>Degree programme and option</b>	
Vasily Babenko		Informational Technology	
<b>Name of the bachelor's thesis</b>			
Analyses and improvements of business processes with IT			
<b>Abstract</b>			
<p>The aim of the study was to create a software solution for the small-business process optimization and informatization and, based on this work, to learn several technologies, methods and tools of software development. As an additional theoretical result of work, the complete software project pipeline description was expected.</p> <p>The work theory was researched and studied, solution project was planned, developed and implemented in field. During this work main aim was reached and correct results acquired. The results demonstrate needed skills and work experience acquired by the developer as well as describes and illustrates one of the possible software development processes with guidelines for future reuse.</p>			
<b>Subject headings, (keywords)</b>			
Software development, business process analysis, C#, SQL, XAML, MS Visual Studio 2013, Expression blend 4.			
<b>Pages</b>	<b>Language</b>	<b>URN</b>	
83	English		
<b>Remarks, notes on appendices</b>			
<b>Tutor</b>		<b>Employer of the bachelor's thesis</b>	
Matti Koivisto		Reaalikirppis	

## Contents

List of abbreviations .....	5
1. INTRODUCTION.....	6
2. MODERN SOFTWARE DEVELOPMENT.....	10
2.1 Software development tools .....	11
2.3 Software development planning.....	12
2.4 Competitor analysis .....	17
2.4.1 Campground Master by Cottonwood software.....	18
2.4.2 Space Rental Tracker by Spirit Works Software Inc.....	18
2.4.3 Booth tracker by Global E-SoftSys Pvt.Ltd.....	19
2.4.4 Summary of the competitor analysis .....	20
3. BUSINESS-PROCESS SPECIFICATIONS.....	21
3.1. Information aggregation methods.....	21
3.2 Data collection and description in the work.....	23
3.2.1 Data collection methods .....	23
3.2.2 Visualization of the collected data .....	24
3.2.3. Data-Flow Diagrams .....	30
3.2.4. Business Process Modelling .....	31
3.3 Functional specifications of the solution .....	32
3.4 Post-planning changes.....	35
4. PROTOTYPING .....	37
4.1. Sketches of the user interface .....	38
4.2 User interface mockups .....	43
4.2.1 The actual versions of GUI .....	44
5. DATABASE DEVELOPMENT.....	47
5.1 Database in theory .....	47
5.2 Project database implementation.....	50
5.3 Stored procedures .....	51
6. CODING.....	56
6.1 Programming environment.....	56
6.2 Coding process and code examples .....	57
7. TESTING AND IMPLEMENTATION .....	65
7.1 Modern software testing .....	65
7.2 Project testing methods.....	69
7.3 Injection in the workflow.....	73

	4
8. CONCLUSION.....	75
APPENDIX .....	77
9.1 References:.....	77
9.2 Additional documents.....	82

### **List of abbreviations**

UML – Universal Modeling Language

MoSCoW – acronym, separating all features on four categories: must be In project (Must), will be after main features implemented (Should), could be implemented, if there would be resources (Could), definitely will not be in project (Would not)

MS – Microsoft

SQL – structured query language

XAML – extensible markup language

C# - (C Sharp) object-oriented programming language from C language group.

DFD - data flow diagram

SADT – (aka IDF0) Structured Analysis and Design Technique

BPMN – Business Process Model and Notation

DDP - data-driven projecting

IDD - interface-driven development

ERM - entity-relationship model

GUI – graphical user interface

DB – database.

To be continued

## 1. INTRODUCTION

Currently, the world is going through informational phase and many areas of human life and activity could be improved with informational technology additions. In this thesis I will describe an example of improving of the business processes of Reaalikippis. This organization provides space rental services for flea-market type of selling. This work is done to show the full pipeline of the production of small-business-oriented software solution, to research and describe different methods of business analysis and software development. This work is also aimed acquiring and proving my software development and business analytics skills.

The theoretical aim of the work is to research a business process and find ways of optimization through informatization, describe the full pipeline of one of the possible ways to develop a software solution and to point out its strong and weak points. The practical aim is to get a ready-to-use product, implement it in the working environment, get valuable skills and experience through the development process.

As a result of work, we will get a ready-to-use product, skills and experience in software development and a step-by-step plan in the development of a small-business-oriented software solution, which could be used and developed in future. This solution is expected to be inserted in a current workflow of the company with future benefits to the organization, and the step-by-step plan could be used by other persons or teams of developers as a backbone for future small-business-oriented projects.

The main point of the study is to develop an IT solution for the management optimization of the business process “Table rental service in the shop” in Reaalikirppis. This goal could be achieved by working through various steps.

The work is divided into chapters, each describing a step in the development process in the sequence. In this work the theory part will not be separated, rather it will be in combination with practical part in each chapter.

First of all, I want to briefly describe the environment in which I will work and methods and strategies of this work. Chapter 2 briefly describes the organization, tools and methods of the work with some logical conclusions and derivations.

Specifications of business processes should be clear, its data flow and operation sequence should be examined and refined into logical algorithms and procedures. This could be achieved by creating different types of models and analytic mechanics, gathering information from different sources and by different methods, and extruding all the needed conclusions from the results. Most information about all the studies I have done to achieve this I will introduce in Chapter 3.

As a conclusion from the previous step, some directions of optimization and improvements and ways of their implementation should be discovered. In our case, an obvious way of optimization through IT is creating a specialized software product. After the preparation and analysis phase is over, an environment for future development should be chosen, set and researched. Modern informational technologies are based on a huge variety of technologies, platforms, languages, and methods. From such a huge pool developers should choose tools which suite situation and/or are comfortable and convenient.

Many software solutions, especially small-business-oriented, consist of three main parts: representation, computation and data storage. In our case the representation part, user interface, will be developed first. The reasons for this I will describe in Chapter 4 along with actual development. The development of graphical user interface is usually done through preparing sketches and models based on the gathered data, discussing with end-users, correcting and prototyping the final interface. In this work this will be done in two graphical tools MS Visio and MS Expression Blend.

The storage part can usually be done in two different approaches: save files\archives or databases. In this work will be used a database approach and MS SQL server will be described as database service. Data storage part includes data storage modeling, creating database logic and developing database functionality based on the SQL language and features, provided by the server environment. All about database part of this project, described in details in Chapter 5.

The last and core part of the product is its main code, the combination of algorithms, procedures and methods that will be used to perform most features of the solution. This part will be described in Chapter 6. There is a huge variety of languages and developer environments for creating, writing and testing coding projects. This work will include information about language C#, XAML and the developer environment MS Visual Studio. As in usually done, before coding will be done modeling and planning, for better understanding and predicting the future code.

While coding and after coding is done, before the project could be released, it should be tested to find out, if it is containing any errors or matching all requirements. Testing could be done in different ways and forms, by different tools and for different purposes. In Chapter 7 I will describe the importance of testing and the basic techniques of testing planning, Risks analysis, Unit testing, Black Box testing and White Box testing.

The last step in the project is injection of created product into current workflow. Even if most of the local specifications were gathered, requirements met and tests performed, there is no 100% certainty that no errors or complaints will uncover at this stage. Software will be run in a real environment, used by real persons, and I will perform real tasks. After some period, feedback will be gathered and possible changes and improvements could be done. Feedback can also provide a direction of future development of the project and allow predicting, solution improvements to business process and its possible business value.



In this work I will mention various technologies and tools, I used for developing my practical project. These technologies are:

- Systematic process analyses
- Expert interview technologies and brainstorming
- Functionality descriptions in IDEF0 (SADT) and BPMN notations
- Description of informational specifications of process by DFD
- Universal language of system and process modeling UML (diagrams Use Case, Classes, Activity)
- Object-oriented programming language C#
- Script language XAML
- Environment for rapid-development MS Visual Studio 2013
- Environment for development and prototyping MS Expression Blend 4
- Database server MS SQL Server 2012
- UML environment Visual Paradigm 13
- Business graphic creator tool MS Visio 2013
- Text editor MS Word 2013
- Testing techniques Unit testing, White Box testing, Black Box testing, Test-driven development, Risk-based testing

## 2. MODERN SOFTWARE DEVELOPMENT

Modern methods for software development represent one significant opportunity to improve a company's bottom line. Once viewed as simply a way to automate back-office operations, today's software development features advanced approaches that create more agile, end-to-end, continuous delivery (better known as DevOps) capabilities that can quickly align a company's technology infrastructure with its rapidly changing business needs.

Modern environment and needs will require IT/software professionals to acquire new skills and improve their development processes. Software organizations need to expand capabilities, evolve practices and focus more strongly on results as well as improve collaboration throughout the software lifecycle. Business, development and operations teams also need to determine the right mix of sourcing for their software needs, and obtain skills and capabilities from outside the organization when necessary. (IBM Global Business Services 2013).

This work is based on a real practical task, resulting from the needs of a small-business organization (next K). K is a service provider, working in providing and organizing a place for small personal commercial needs. The clients renting a physical place (next "table"), where they can sell different material goods as in a centralized market place with shop mechanics. The organization accounts the all sales and organizing the marketplace by support, cleaning and efficient table distribution. The last mechanics were chosen for improving with informational technologies by development special software solution for optimization, automation and boosting current connected processes.

K already implemented software solution for sales support, which highly increased efficiency of all connected operations and gave a huge advantage in workflow. The sales management technology was installed as a compilation of working machines, equipment, and special software, developed for similar needs and mostly optimized for solving all tasks needed.

## 2.1 Software development tools

In the modern software development a wide range of tools, languages and techniques are used for developing products with different parameters for different needs. Under the term of programming language is usually meant a formal constructed, artificial language, usually based on other language or its abbreviations which are designed for translating commands and instructions to a machine, in our case a computer. In fact programming languages only exist for the purpose of bridging the gap in the level of abstraction between the hardware and the real world. Languages differ by various parameters, such as complexity, possibility of using different mechanics etc. The role of languages in programming has been downgraded in favor of software methodology and tools; not just downgraded, but totally repudiated, when it is claimed that a well-designed system can be implemented equally well in any language (Ben-Ari 2006). Currently the tendency is for increasing the popularity of high-level object-oriented languages with dedicated interpretations and high multiplatforming. The straight programming with machine code with low-level of abstraction is not very popular now, as compilers, specially developed and optimized for this task, do most of conversion to a machine code. This allows increasing the speed of the code development, its quality, clearness and optimization. Also, modern programming languages are much easier to learn and understand than a raw machine code.

Various languages use a broad range of Integrated Development Environments (IDEs). IDE is a software application that provides comprehensive facilities to computer programmers for software development. An IDE normally consists of a source code editor, build automation tools and a debugger. Most modern IDEs have an intelligent code completion. IDE usually helps developers to write code faster, in a clear, human-readable form, with some error-correcting tools and some tips (help) in the navigation and auto-completion in phases. The classic software development process typically begins with a high-level architecture process which includes modeling the software objects and their interactions. Next, an editor is used to write the source code according to the proper syntax, and a compiler is invoked to translate and link the software to an executable binary format. Finally, a debugger is used to catch any errors, thereby ensuring the correct program behavior. Classic IDEs such as Microsoft's Visual Studio or Borland's JBuilder

have revolutionized the software development process by providing enhanced tools support for editing source-code, as well as modeling and debugging tools which have enabled developers to produce higher quality software while simultaneously reducing the required effort. (Kim 2002).

Even though it was mentioned that modern developers can solve tasks with broad variation of tools and the tools choice is not very important, one should think about it before starting the development. The choice of development tools depends on the task, client IT environment, and developer preferences and experience. In fact, even if some tools are not perfect for the solution, but are better known by the developer than the tools that suit better, the developer should use the most convenient ones. It usually leads to better results than studying and mastering new tools. In our case, we need a product for small-business environment on the platform of OS Windows.

Specifications for small-business software solutions usually consist of narrow specialization on existing task, with high usage of user-friendly interface. At the same time, some usual problems, such as copy-protection, high security measures or multiplatforming are usually not so important and get lower priority in the development. This type of solutions are often not for mass-usage, require few computation resources and usually integrate inside collection, accounting, structuring, concentration, manipulation and visualization of data, and/or are used for informatization and replacement of tasks, currently executed on paper or with non-IT tools. Such requirements usually lead to the development and efficient usage of databases and maximum graphical simplification, visualization and algorithmization of all the processes for decreasing amount of workload in one point.

### **2.3 Software development planning**

The first real problem a developer facing in the beginning of the software solution development for small-business process implementation is gathering data to define customer requirements and project parameters. It is a normal and often a typical situation, when the customer does not know the final product parameters and its details and requirements, especially, if the business is not connected to IT. To find all this

information, to formalize and precisely represent it is a common task of a developer. There are different techniques of solving this problem, including data aggregation (questionnaires, brainstorming), analytics (benchmarking, modeling), visualization and the utilization of acquired data in development.

In the large development companies, company-wide normative models for software development are the rule rather than the exception. These models are typically based on software engineering approaches regarding design. They typically emphasize the following: the separation of analysis and design, design as a way to fulfil the requirement specification and hierarchical decomposition of the design work (Löwgren 1989).

The design process appears to be a process of adding formality as a design progresses with constant backtracking to correct earlier, less formal, designs. Thus, the designer starts with a very informal picture of the design and refines that by adding information and making the design more formal (Sommerville, 1989).

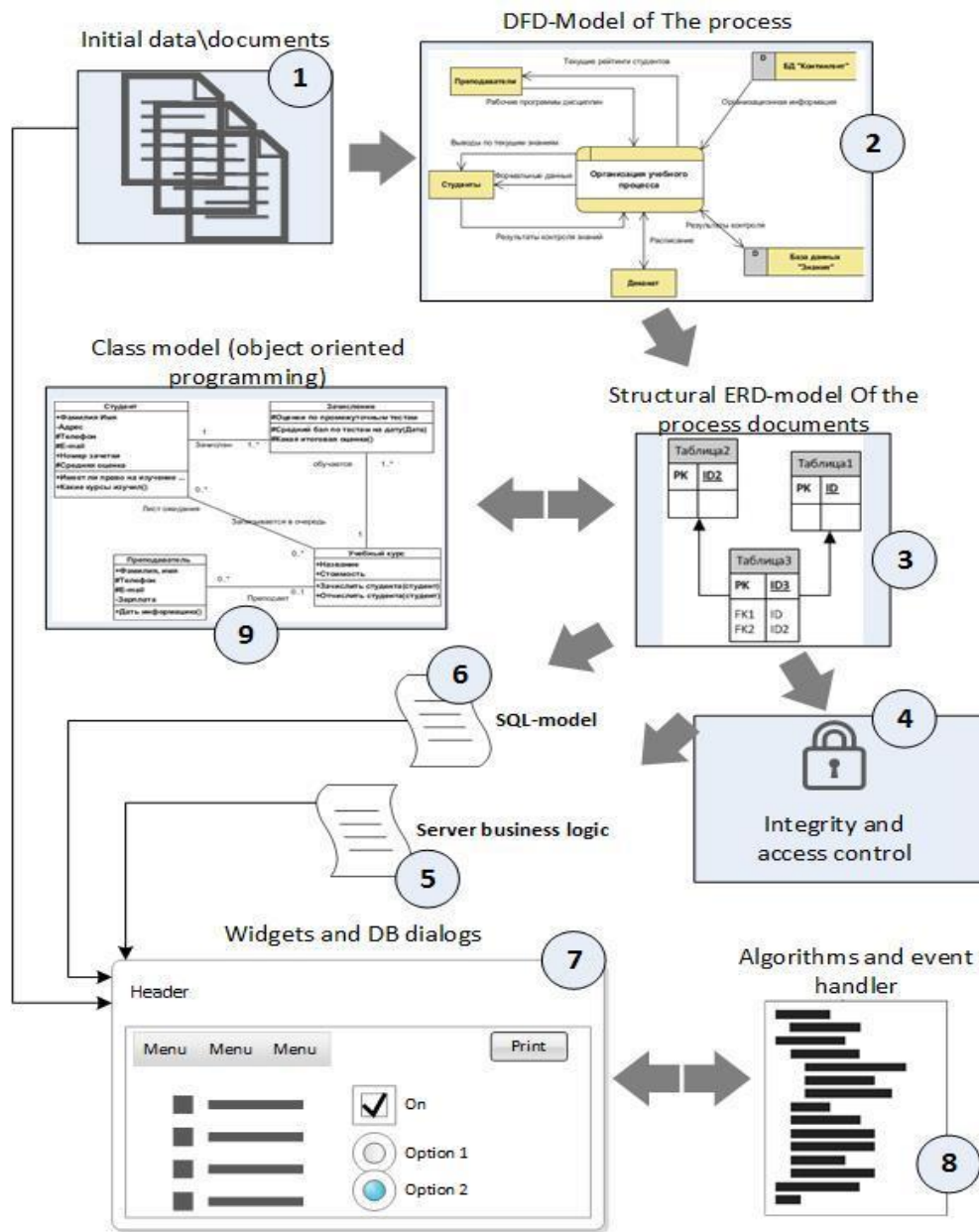
Software solution development usually consists of multiple parts whose development order is defined by development strategy. My thesis will describe the following three strategies:

- Data-driven projecting (DDP)
- Interface-driven development (IDD)
- Test-driven development (TDD)

Data-driven projecting is a strategy of development, when first the data-flow of the process is analyzed, all data connections and algorithm logic is defined. Project realization starts from database design and coding, while interfaces are usually developed in the end, under the code and DB logic.

Figure 2.1 shows the main steps of analysis and development in the DDP order: First all the data documents are gathered, analyzing their connections and manipulation. Then the database is designed for the best suiting the data flow. Then, the code is written to provide functionality for the database and data-flow operations, and in the end the

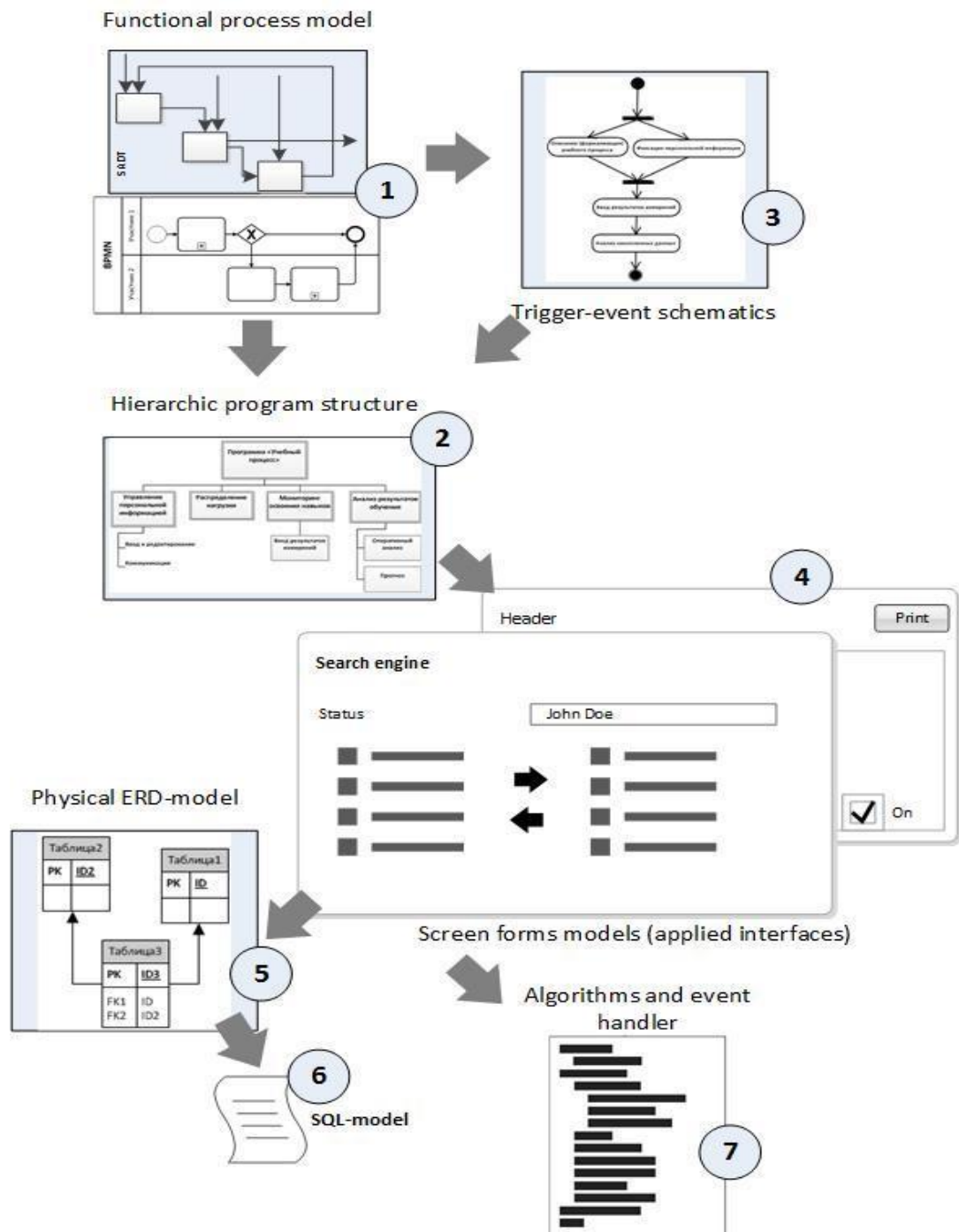
interface is designed and connected to the system. Such strategy is good for inventory systems, when the interface could be minimalistic, and the data-flow is complicated.



**FIGURE 2.1 DDP (data-driven projecting) logic** (Babenko 2016)

Interface-driven development is an opposite strategy, when the first part of analysis and research becomes an interface, which is then used as a model and direction for further development of all the other parts. Figure 2.2 describes the logic of IDD: First, there is algorithmization and the analysis of possible interface solution. Then, the interface is modelled and coordinated with the customer. The final version of the interface defines the main code logic and database structure. This strategy is better, when the interface is

in higher priority, like when a customer wants everything to be clear and easy-to-use even for untrained personnel.

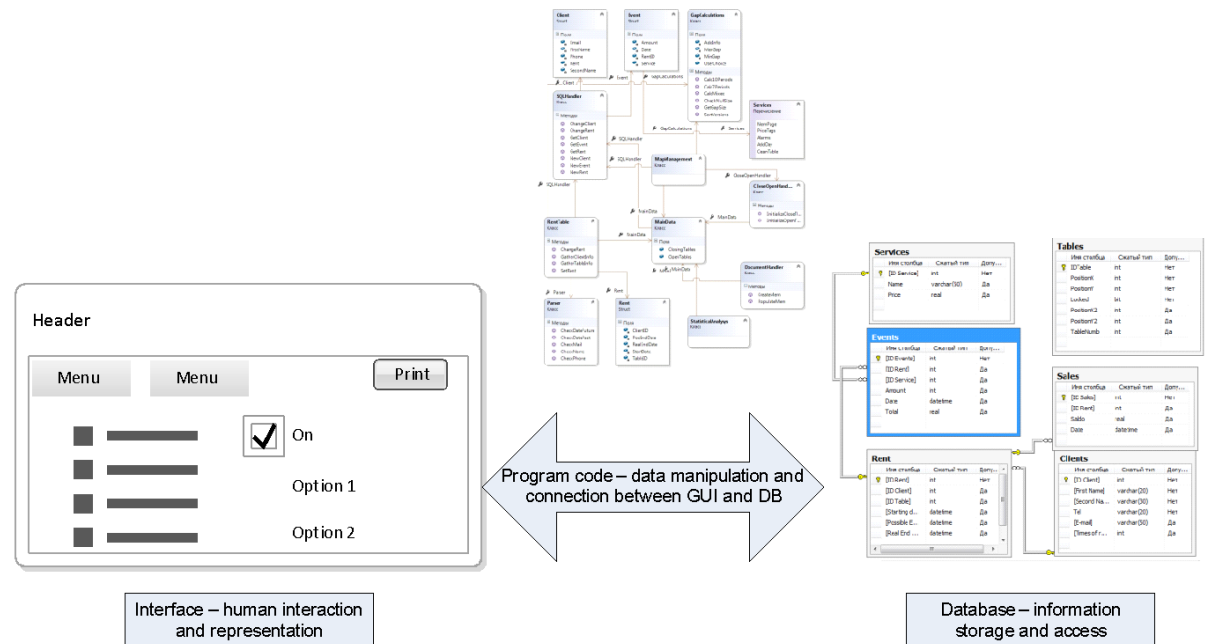


**FIGURE 2.2. IDD (interface-driven development) logic** (Babenko 2016)

Test-driven development is usually used in rapid software development and is highly popular in “agile” techniques, because it is based on development and all parts fitting in very narrow requirements. It is also based on the logic that tests and requirements are

defined and written before the actual development, which is designed for satisfying them in the most fast and efficient way. More about TDD will be in Chapter 7.

As Figure 2.3 shows, in small-business projects the body of a software solution usually could be divided into in three different parts: Interface, Code and Database.



**FIGURE 2.3. Parts of software solution project**

The first part, interface is an outer shell of the solution, responsible for interaction with the user and gathering and representing information in human-readable form. Information storage is another part of the project, responsible for storing and manipulating gathered or produced information. The last part is a code is the project’s “flesh” where most processes are done and where data is used and produced. This part is also responsible for the connection between interface and database, delivering and organizing data in both ways. Each part is developed in a different way, in logical connection with other parts. The order and methods of development depends on development strategy.

Based on K is environment and requirements, the IDD was considered the best strategy, because good interfaces were positioned as a high-important part. Close communication with client, analysis and modeling the GUI allows understanding the algorithms and the logic of all other development parts more clear, which helps in



further development. Moreover, interface research is usually faster, cheaper graphics, then data analysis.

## **2.4 Competitor analysis**

I have also performed some market research. In the small-business software solution market, the universal products are quite rare, because it is difficult to make a solution, satisfying all different environment specifications. Still, there are usually some products that try to be universal and adapt to any client need in a specified sector of business. It is important to do the competitor research to form product market strategies. Competitive marketing strategies are useful either when they position a product's strengths against competitors' weaknesses or when choose positions that pose no threat to competitors. As such, they require that the strategist be as knowledgeable about competitors' strengths and weaknesses as about customers' needs or the product's own capabilities. (Czepiel & Kerin 2009.)

I have searched for possible analogs of my project in the Internet. This is always important to do, because it may lead to the fact that exactly the same solution already exists or there are some competitors. Based on the competitors' examples it is possible to evaluate one's own product, find its strong and weak sides and compare it to them. Examining competitors' products usually can give tips to solve current problems and provide good vectors of development. The developer should understand position of the product on a market, its advantages and disadvantages, directions of improvement and priorities in management.

The number of solutions to improve a small rental business is not huge. Some solutions were made for house rental, but could be used for table renting as well. From the group of products I want to scope on three closest analogs, main competitors:

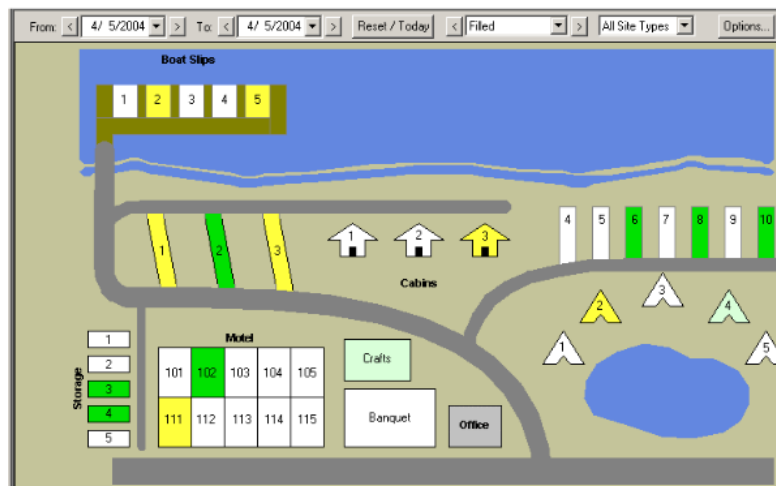
- Campground Master by Cottonwood software,
- Space Rental Tracker by Spirit Works Software Inc and
- Booth tracker by Global E-SoftSys Pvt.Ltd

In the following section I will shortly introduce them.

### 2.4.1 Campground Master by Cottonwood software

This rent management solution was designed for managing accommodations, but it should be possible to use it for the table management. It is quite complex solution, being able to manage customers and timetable for a custom map, which users should draw on their own. It can also handle transactions and sales, based on barcode reading. (Cottonwood software 2016. A.)

In Figure 2.4 we can see the main map-interface:



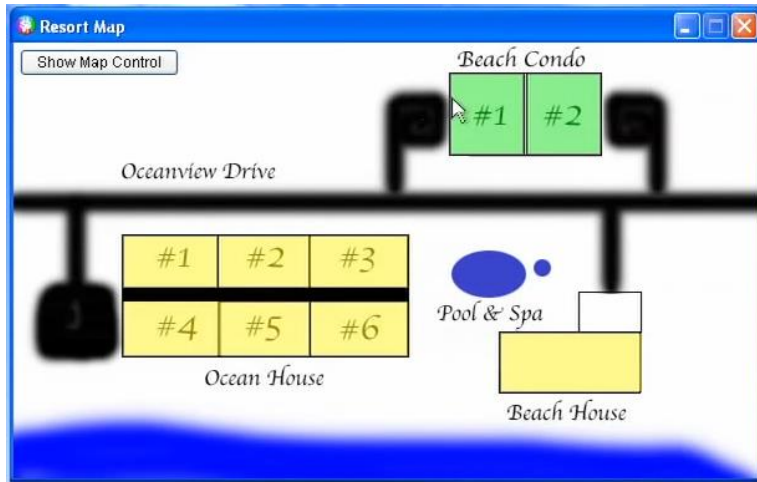
**FIGURE 2.4. User interface of Campground Master  
(Cottonwood software 2016. B.)**

More about it could be found in short product brochure. (Cottonwood software 2016. B.)

### 2.4.2 Space Rental Tracker by Spirit Works Software Inc.

This solution is a complex of various software products, designed for usage in various situations, including over 15 separate programs for managing the whole rental business. This solution could be used for table tracking and rent management, but the complexity and unneeded additional load of software makes it to look inefficient for our task. The design of the demonstration site was not built correctly to make potential customers understand all the benefits of the product. (Spirit Works Software Inc. 2016.)

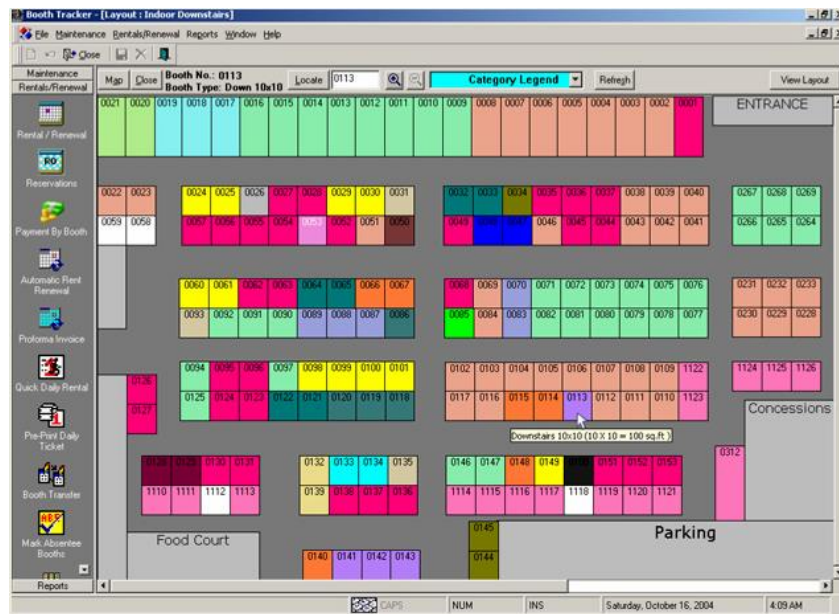
As shown in Figure 2.5, the interface and concepts of work with the program are very close to the previous competitor.



**FIGURE 2.5. User interface of Space Rental Tracker**  
(Spirit Works Software Inc. 2016.)

### 2.4.3 Booth tracker by Global E-SoftSys Pvt.Ltd

The last product in the list is Booth tracker, designed specially for flea markets’ Self Storage Places, Mini Storage Places, Tradeshow Organizers, etc. This product is in my opinion the closest to our project. It is a big and complex solution that handles the management of almost all aspects of small and medium sized businesses that work with storage and place rentals. (Global E-SoftSys Pvt.Ltd 2016. A.)



**FIGURE 2.6. User interface of Booth tracker**  
(Global E-SoftSys Pvt.Ltd. 2016. B.)

This solution is much more complex than the two previous. One the interface is full of various features and it looks quite difficult to understand, as shown in Figure 2.6.

On developers web site was mentioned, that product was made to support and handle self-organized flea markets. This makes it suitable for performing most of our tasks. More about the product could be found in their short-brochure. (Global E-SoftSys Pvt.Ltd. 2016. B.)

#### **2.4.4 Summary of the competitor analysis**

In conclusion for this short overview, I want to point out some main ideas: mentioned products all have in common some characteristics, like broad functionality, flexible adjustments, but with this also coming high complexity of the system and huge price. For example, first product is sold for 800\$ in minimal complication and require about 100\$-200\$ for some additional features, such as barcode readers. Summarizing this prices and various features, that will never have usage in our Organization, I can say, that purchasing any of this product will lead to waste of resources on unneeded features. Also, products, even if they programmed quite well, have very outdated design close to age of Windows 95. Moreover, I do not saying that design is too user-unfriendly, but it totally new and uncomfortable for workers of Reaalikirppis, which means, that they need to spent time resources for reorganization of their workflow and education, just for start to use any of this solutions. Based on that, I found it more comfortable and easy to develop new solution focused precisely on current environment. For K it will also be the cheapest solution.

### **3. BUSINESS-PROCESS SPECIFICATIONS**

Before the creation of any complex product, especially in high-tech environment, various studies should be done, to gather, analyze and produce all the needed information about the product's specifications, requirements and parameters. This process usually involves multiple methods and techniques, created to simplify, organize and order the tasks. In this chapter I will describe the methods I used for gathering all the needed data about my project.

#### **3.1. Information aggregation methods**

First of all, I wanted to get any possible detail of how the solution should look from the customer point of view. Usually, as it was mentioned, the end user has very poor understanding of all the technical details and possible results of the work. The most important in gathering information from the customer is to crystalize the requirements for the product.

User requirements capture can therefore be difficult because:

- The developers are not the users.
- Inadequate requirements information may be collected from users.
- Each individual types of users may have their individual requirements, but cannot define the overall system requirements.
- Users do not know what the particular software system can and cannot do.
- Too many nice-to-haves that wouldn't actually be used.

Capturing is usually performed through requirements identification technics such as dialogs, expert-interviews, brainstorming and questionnaires/surveys. Table 1 summarizes these findings.

**TABLE 1. Data collection methods (ATC 2007)**

<b>METHOD</b>	<b>DESCRIPTION</b>	<b>BENEFIT</b>	<b>DRAWBACK</b>
Scenarios /Use cases /Personas	Detailed realistic examples of how users may carry out their tasks in a specified context with the future platform	Personas can bring user needs to life	Scenarios may raise expectations too much. Personas may over simplify the population.
User Surveys	A set of written questions to a sample population of users. Surveys can help determine needs, current work practices and attitudes to the new system ideas	Relatively quick method of determining preferences of large user groups. It also allows statistical analysis	This method may not capture in depth comments and may not permit follow-up.
Focus Groups	This technique brings together a cross-section of users in discussion group format. A useful method for requirements elicitation	Allows rapid abstinence of a wide variety of user views	Recruitment effort to assemble groups. Dominant participants may influence group disproportionately
Interviewing	A series of fixed questions with scope for the end user to expand on his response	Interviews allow quick elicitation of ideas & concepts	Negotiate access/possible different opinions from different users
Existing Systems /Competitor Analysis	Comparison of expected product with existing systems	Effective in identifying current problems, possible new features and acceptance criteria	This method may lead to including too many new functions or make system too similar to a competitor's.
Dialogs/experience backlogs	Informal communication with clients before/during development	Allows fast data gathering with small resource costs for brief corrections and improvements.	Small and uninformative amounts of data, sometimes incorrect and leading to wrong decisions. Difficult to formalize
Delphi method	Reaching consensus among participants about an issue of concern through a survey consisting of a series of rounds with questionnaires (Amal 2005)	Allows the participants to freely express their opinions, refine their views in light of the progress of the group's work, informs the participants of the other participant's perspectives (Skulmoski, Hartman, Krahn 2007).	Complicated, not suitable for small-business environment, useless in small groups of developers of in direct client-developer communications.
Brainstorming	A tool used by teams to bring out the ideas of each individual and present them in an orderly fashion to the rest of the team	Encourages creativity. Rapidly produces a large number of ideas. Equalizes involvement by all team members. Fosters a sense of ownership (Air University).	Overloading with creativity may generate too many unneeded ideas, which will lead to a wrong development decisions.

## 3.2 Data collection and description in the work

### 3.2.1 Data collection methods

Now I want to scope on the methods I used in my work.

Expert-interview is usually done by straight dialog between a person who knows all the details needed about development process and environment (“Expert”) and the person who knows the best what is needed from the product (“Client”), and usually based on pre-made questions, done by the expert. Most of the questions and answers are “open”. They do not have a predefined answer “Yes” or “No”, and help to describe the project through the customer point of view.

Both dialogs and interviews were done in the beginning of analysis, but they are quite difficult to save as documents, except in some handwrites, used to fix key points in the memory. It is sometimes possible to save dialogs and interviews on records, but it must be clear and agreed by both parties of the dialog.

Questionnaire: The questionnaire form of the gathering information is more formal and allows keeping all data stored in physical form. It also provides anonymity, if it is needed. It also helps to already visualize and organize data, if the questionnaire is not made of open questions. A fixed variety of answers allows forming statistical analysis, which is very helpful and convenient with massive numbers of interviewers.

In our case, the questionnaire was used to find out some information that was still unclear after interviews and to save information that was already discuss. Because of a small number of workers in the Reaalikirppis, the questionnaire was given only to three people who will most probably be the main users of the product.

The results of the small-scale survey can be summarized as follows. Some points were chosen unanimously, which usually means, that they are really important for the client group. The unanimous issues were:

- Program should be single-user.
- It should keep all client data, including a name, phone, e-mail, table number, rent dates, history of rents.

- Time, given to the customer for choosing is very short, around 5-15 minutes.
- The solution should include these features: table map, calendars, rent history, client data storing tools.
- Automation is not important in normal booking procedures.

Also a tendency could be found that an intuitive and user-friendly interface is highly important, while future expanding and development are not needed.

### **3.2.2 Visualization of the collected data**

The next step in the research process was to create SADT (IDEF0) diagram based on gathered data to visualize and analyze factors of business process workflow.

SADT was created long ago by Ross as a result of ongoing work (1969-1973) in problem solving dating at Softech. SADT is a graphical language and was used extensively for describing complex systems in communicative designs, military planning and computer-aided manufacturing (Dickover, McGowan, and Ross 1977). It was also used in problem analysis and functional specifications. It was found, that this technique is very effective in the requirements definition phase for software design (Ross and Schoman 1977). SADT was adopted as Icam DEfinition for Function Modeling (IDEF0) by US Air Force Integrated Computer Aided Manufacturing (ICAM) in 1980s. There were several other IDEF versions mainly, IDEF1, IDEF2, IDEF3, and IDEF1x.

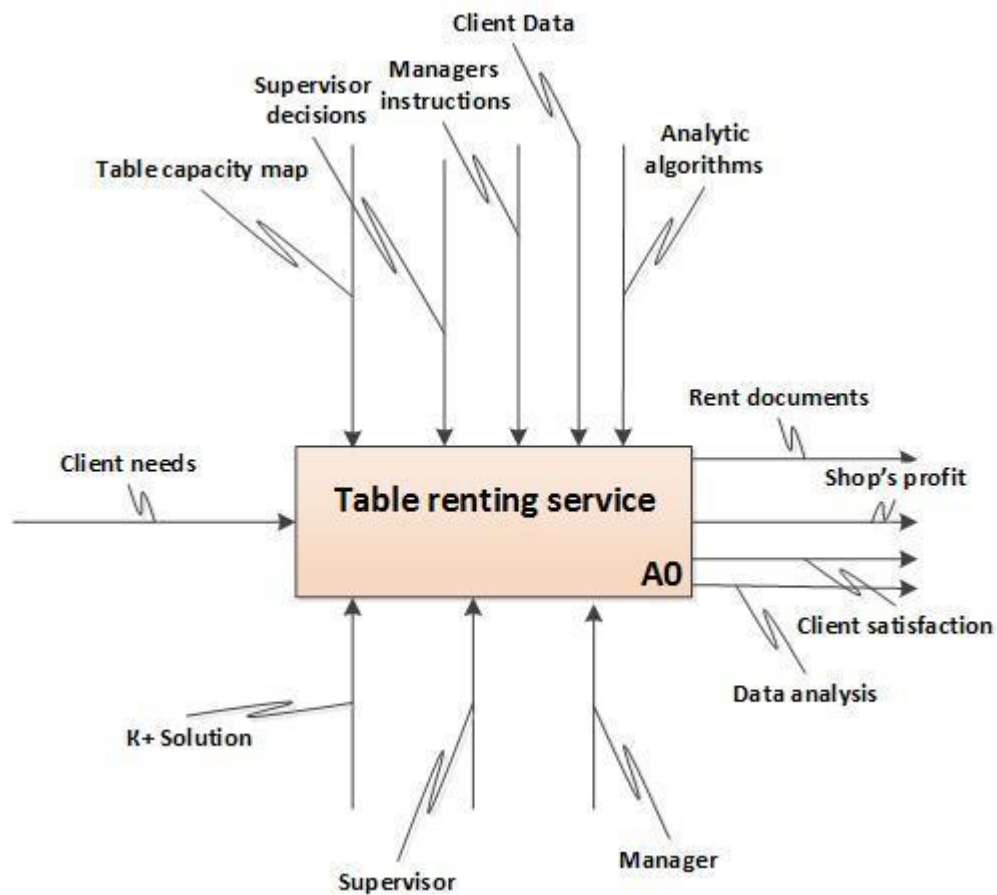
SADT notations consist of box-arrow diagrams (blocks), with four arrows on each side defined as: input, output, control and mechanism and one activity in the middle as shown in figure 3.1 (Fahim, Robinson, Tako 2014). Their definitions consist of the following:



- Activity: An activity is any function or process that serves to transform inputs into outputs
- Input: The data/information required by an activity to start the transformation process
- Output: the data/information produced by the activity as a result of this transformation
- Control: Any constraint that affects the behavior of activity in some way
- Mechanism: Persons, resources, or any means that are required to run the activity

As any process could be viewed as a logical “factory”, this model show, how various inputs connect in processes to process and convert initial need into desired results. Inputs are logically groped as “resources” – data, objects and sub-products, used in processes, “controllers” – persons and controllers who ally the actions. The logic of this model is based on fact, that almost any process could be split into many smaller processes and each process connect inputs and give results. This allows simplifying the algorithms and braking them into smaller parts, which are easier to solve.

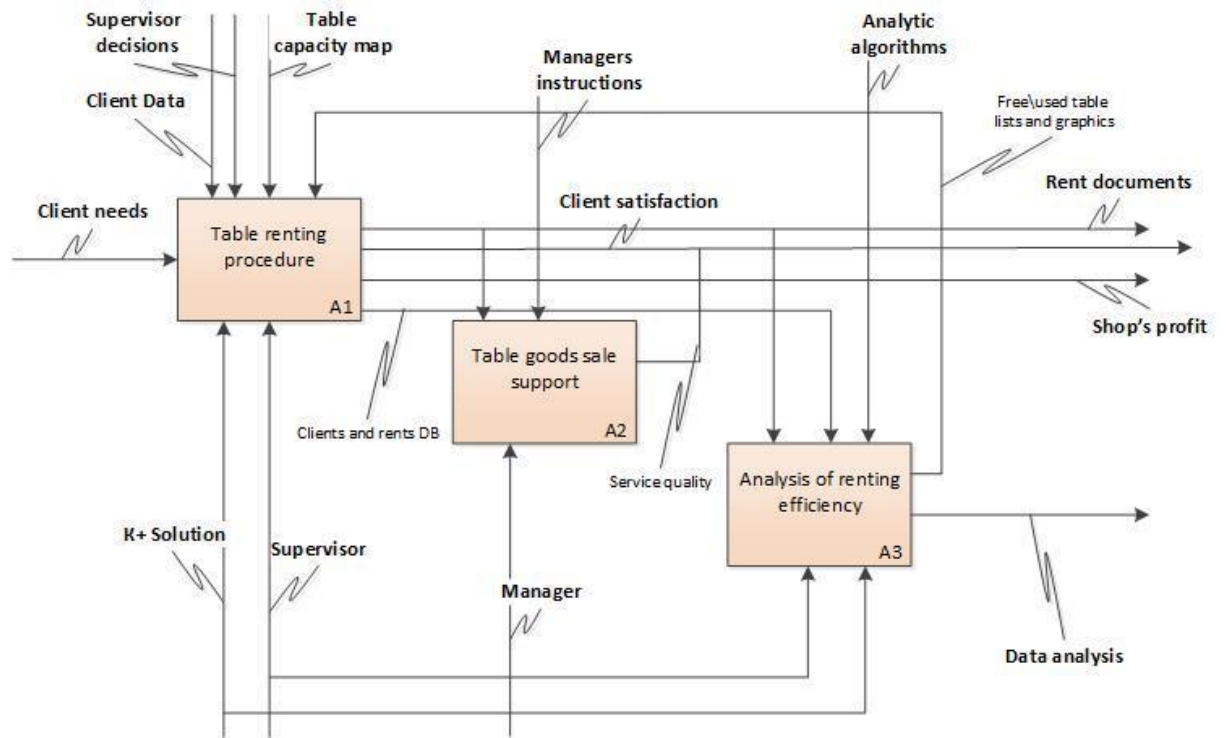
In our case, there is client, who needs to rent table, as a main input. By the process “table renting service” as a result we should get successfully rented table, document about it, shops profits, client satisfaction and some data analysis. The process is controlled by two persons: supervisor and manager and also by this software solution. Also process requires some sub-inputs: Client data, supervisor and manager actions and instructions, map of all tables and some analytical algorithms provided by the solution. All this information is displayed in Figure 4.



**FIGURE 3.1. Main view**

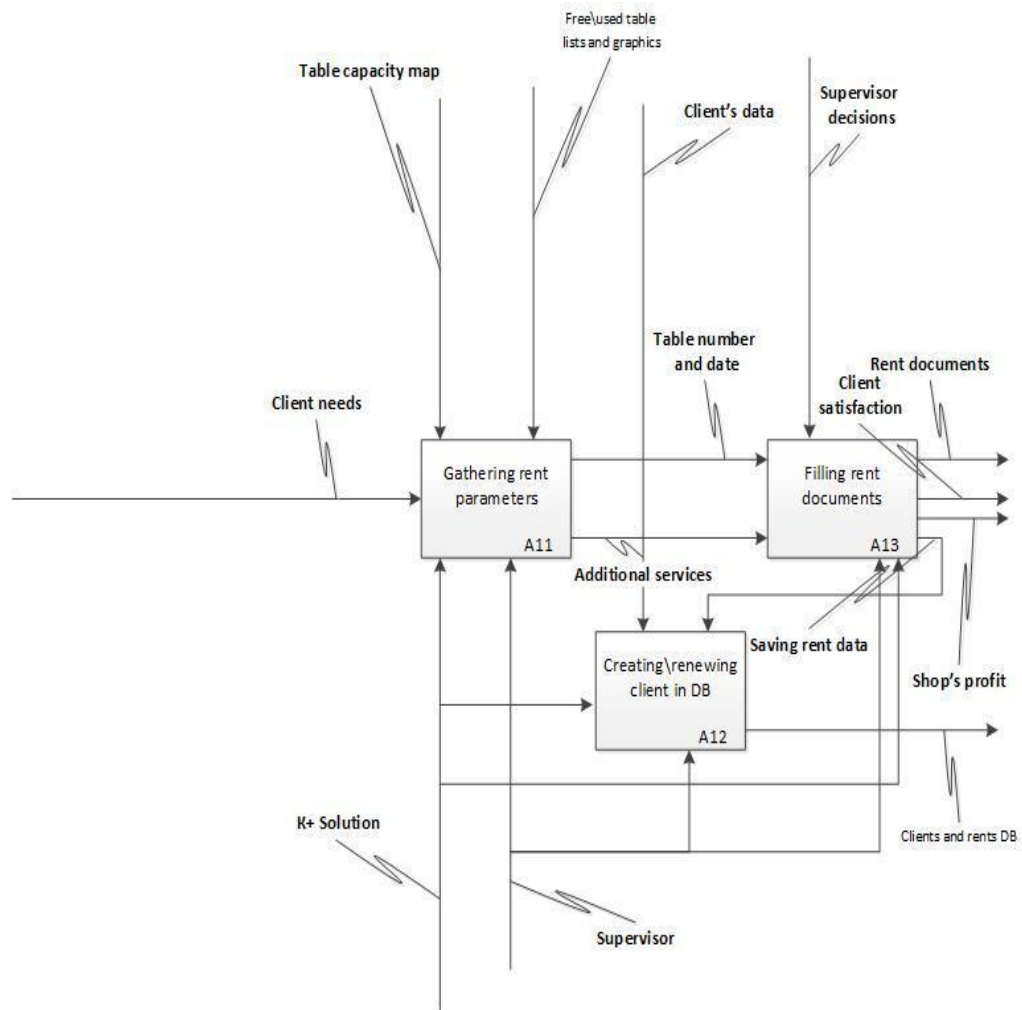
But the process “table renting service” is very broad and unclear, so it is possible to get inside of it and model sub-processes it consists of. In Figure 5 we can see, that table renting service includes three main operations: Renting procedure, sales supporting and final analysis. Already on this stage we can see, that manager handles sales support and this solution is not used in it. We also can see, that sub-processes are parallel, as table renting procedure needs free table information as input, and this data is a product of rent analysis process. We also can see, that main profit and client satisfaction are produced by renting, and there should be more attention on this step.

In this modeling structure is very important, that all inputs and outputs are same in main view and sub-view.



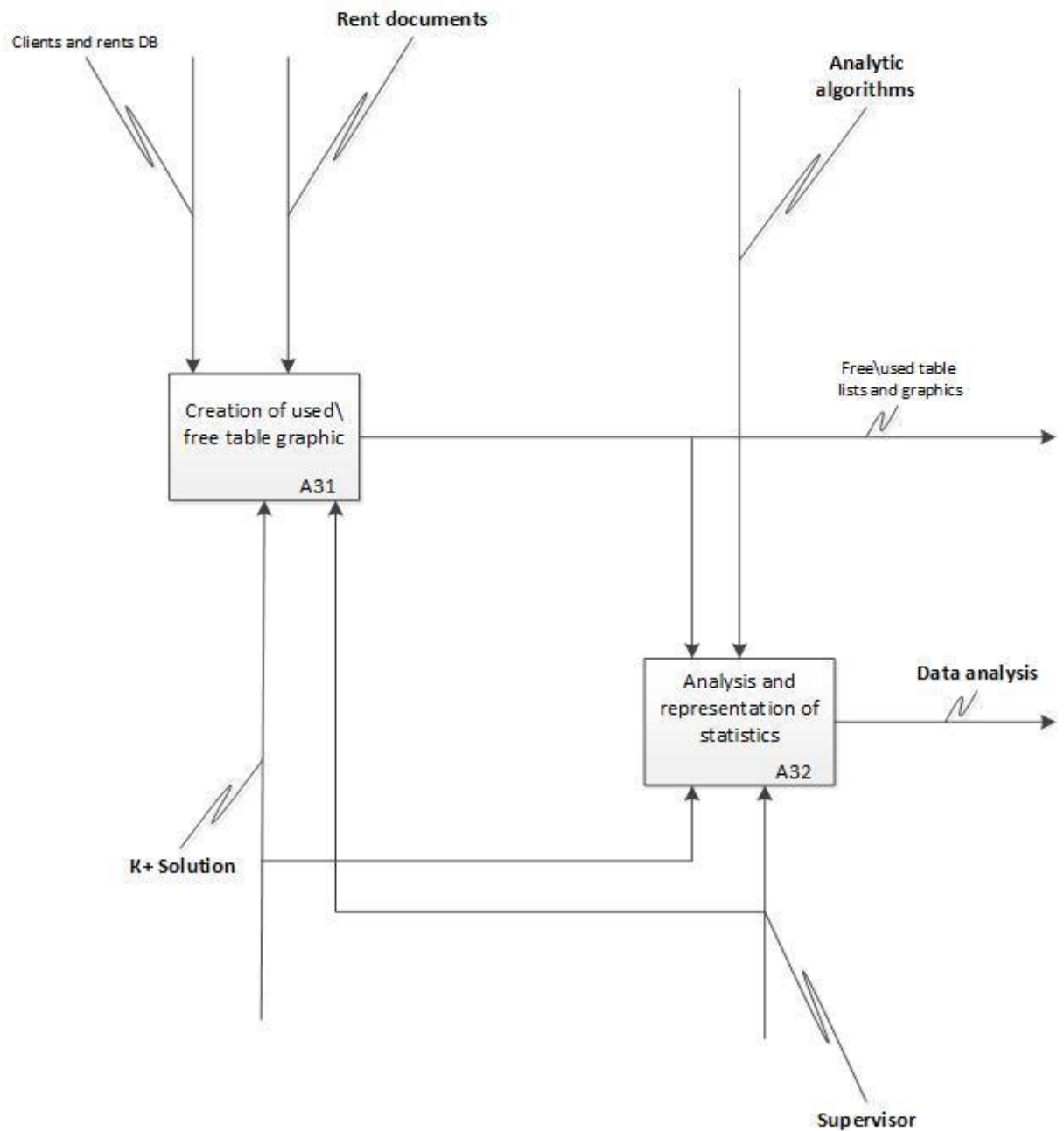
**FIGURE 3.2. Content of Table renting process**

SADT allows to model in many levels, so any sub-process could consists of other sub-processes. This helps to take even closer look on specific processes to define it precisely. In Figure 6, we can see detailed view on sub-process “Table renting procedure”. It also includes three sub-processes: Gathering data, Applying data to the system and Filling rent documents. All processes are controlled by software solution handled by supervisor. Based on table usage data and table map, correct rent details are gathered and submitted to database, and then table is rented and used, during this period client may use some additional services, then rent time expires, final documents are filled and client receives profits (Mylopoulos 2004).



**FIGURE 3.3. Renting procedure close-up**

Another important sub-process which needs close-up view, is Analytics. Analytic process could be split into two main parts: dynamic table environment maps forming and statistical data analysis. The maps forming sub-process gathers all needed data about rents on demand and produce full or focused view on table environment for further usage. Analytic process collects all data stored and performs different graphs and conclusions for end-user.



**FIGURE 3.4. Analysis process close-up**

Problem analysis decompositions blocks 1 and 3 (. Figure 5, 6 and 7) shows that the process is characterized by an online update important control information - the current state of employment tables, tables scheduled release date, etc. Also, in my opinion, process optimization is possible by connecting an operational analysis of the entire history of the leases. The store manager must establish operational management decisions taking into account the answers to the following questions:

- What tables are most in demand?
- Whether there is a pattern in terms of rent claimed?
- One of the most active customer in the lease?

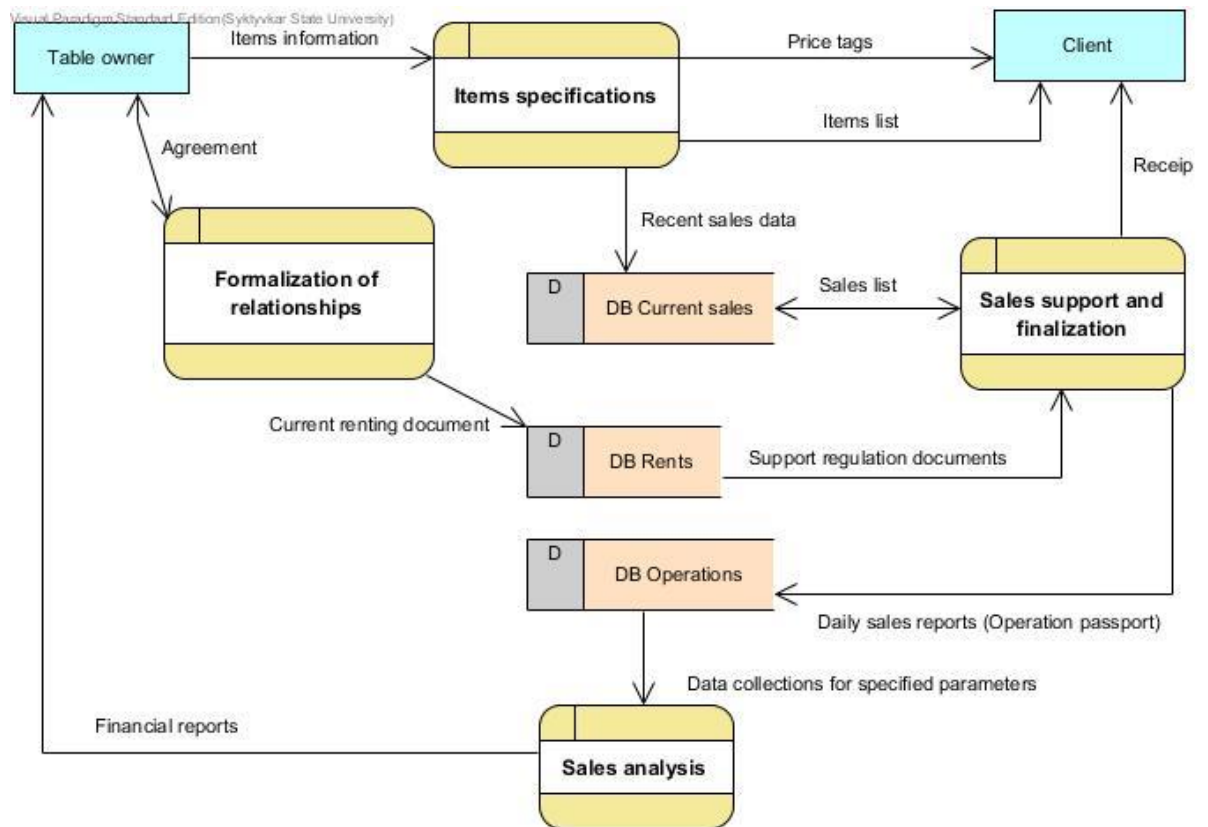
The most effective solution optimizes the creation of a specialized transaction-analytical application that allows to store the data in the desired format and provide the ability to manage data samples and analytical tools.

### **3.2.3. Data-Flow Diagrams**

Next step in business modeling is Data-Flow Diagram, which describes all data (documents) created and changed through business process.

Data flow diagrams (DFDs) reveal relationships among and between the various components in a program or system. DFDs are an important technique for modeling a system's high-level detail by showing how input data is transformed to output results through a sequence of functional transformations. DFDs consist of four major components: entities, processes, data stores, and data flows. The symbols used to depict how these components interact in a system are simple and easy to understand; however, there are several DFD models to work from, each having its own symbology. DFD syntax does remain constant by using simple verb and noun constructs. Such a syntactical relationship of DFDs makes them ideal for object-oriented analysis and parsing functional specifications into precise DFDs for the systems analyst. In short terms it shows what data is inserted in process and sub-processes, what data is in output, what data is created and how. All data is shown as "document" – abstract representation of the data.

DFDs help system designers and others during initial analysis stages visualize a current system or one that may be necessary to meet new requirements. Systems analysts prefer working with DFDs, particularly when they require a clear understanding of the boundary between existing systems and postulated systems (Le Vie 2010).



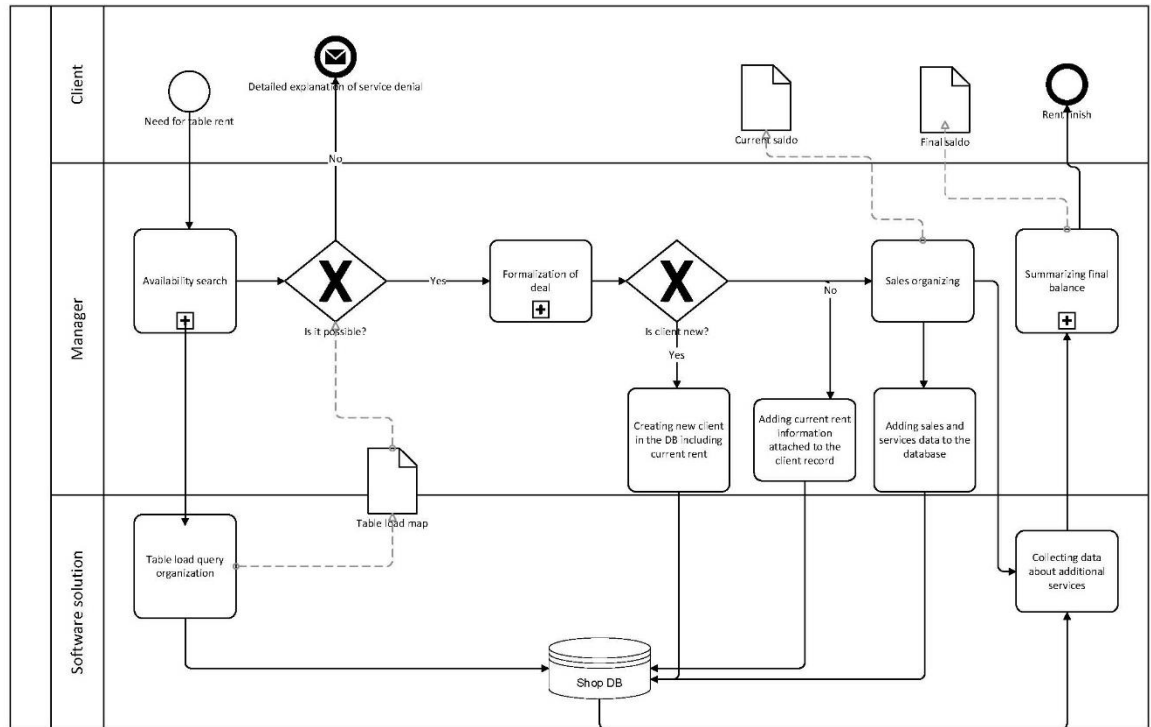
**FIGURE 3.5. Data-flow diagram**

On Figure 3.5 we can see the actual data flow in the company. We can see, that some documents, produced during rent process are saved in the Database.

### 3.2.4. Business Process Modelling

The last research method I want to mention is BPMN (Business Process Modeling Notation) modeling technique. The Business Process Modelling Notation (BPMN), is a visual notation for business process flows modelling, composed of graphical elements, semantics, attributes and properties, which visually compose a Business Process Diagram (BPD) (OMG 2007). Both business analysts and technical developers conceive it to provide a readily understandable formalism. BPMN fits well with Service-Oriented paradigms and Web service technology since it can be almost finally mapped to process execution languages (Owen 2005) in order to obtain an executable process from a BPMN diagram, providing a notation for them. BPMN supports modelling a business process flow using activities, events, gateways for business decisions and flow branching (Pintus, Paternò, Santoro 2010). In the Figure

3.4 we can see the summarization of the process. Deriving from the diagram, it is possible to say, that the main task of the solution is gathering, aggregation and analysis of process information.



**FIGURE 3.4. BPMN diagram**

### 3.3 Functional specifications of the solution

In the analysis and modeling use-case diagram takes a conclusive place and is common and popular, as it specifies most requirements the final product should have. The Use Case Model describes the proposed functionality of the new system. A Use Case represents a discrete unit of interaction between a user (human or machine) and the system. A Use Case is a single unit of meaningful work; for example login to system, register with system and create order are all Use Cases. Each Use Case has a description which describes the functionality that will be built in the proposed system. A Use Case may 'include' another Use Case's functionality or 'extend' another Use Case with its own behavior. Use Cases are typically related to 'actors'. An actor is a human or machine entity that interacts with the system to perform meaningful work.

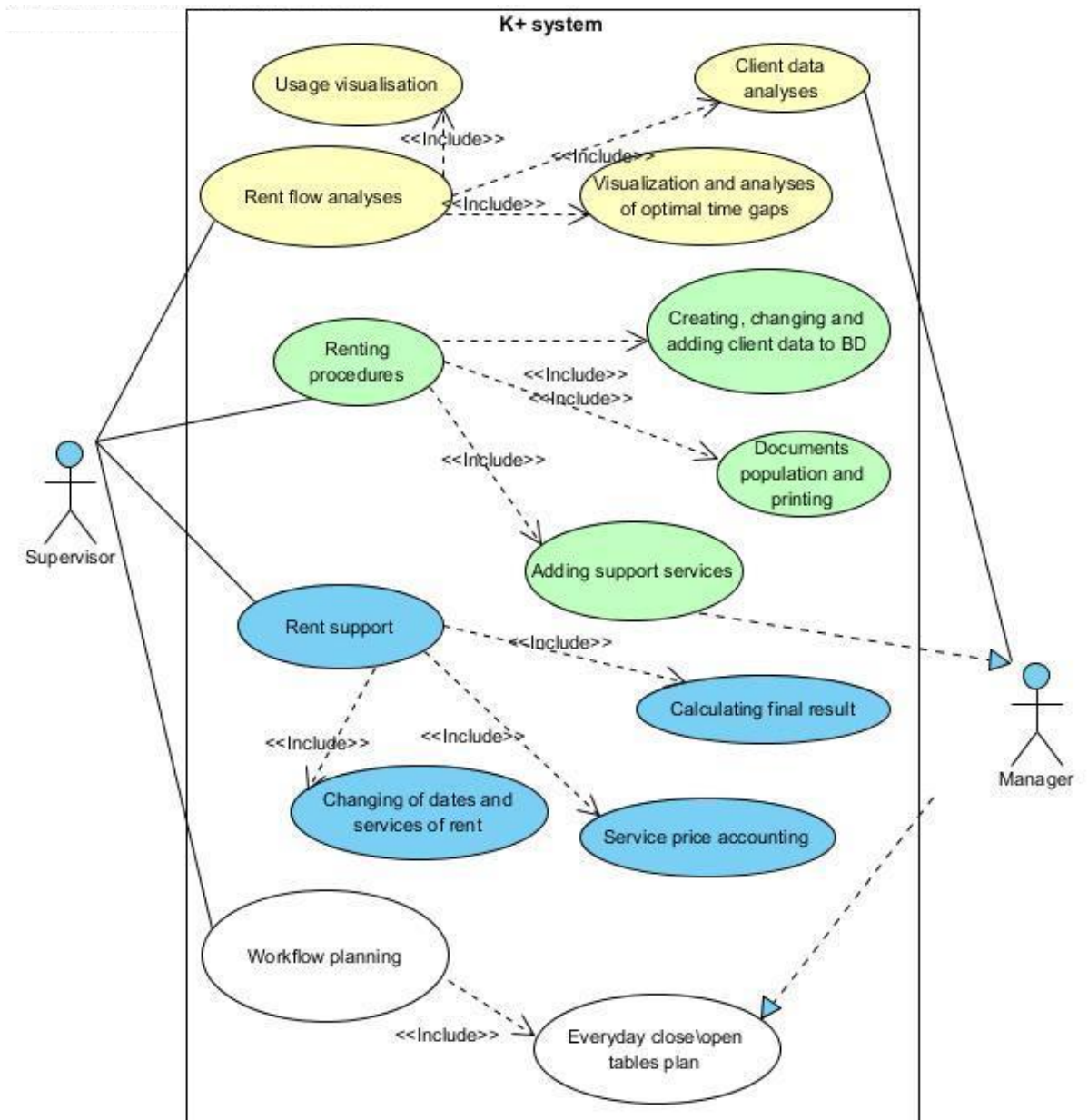
In shorter explanation it allows to point out persons, who perform actions, specifies actions, its dependences and interactions with other processes and users. It shows



complete list of actions, that final product must perform. Some actions can include sub-actions or dependences, variants and connections to other items in the model (Sparx Systems 2004).

Regarding to all data I gathered and analyze, the Use Case diagram of the project was done and is mentioned as Figure 3.5.

In this diagram we can see, that product requirements could be grouped in four main fields: rent handler, data analyses, rent support and planning. Moreover, supervisor will handle most of the actions with the product. The program should be able to perform a rent procedure, which includes manipulations with client data, interaction with support services and printing filled documents; then product must provide support in rent workflow, changing dates and data, if it is needed, adding services and finally calculating all results. Based on gathered data, product must be able to provide valuable information to the supervisor about workflow and correct statistics, and also help on the rent planning stage, giving helping hints on “choosing rent date”, for most optimal rent date placement. For managers solution also should provide tips for everyday management: lists of tables, which start rent today and list of tables, which will close rent today, to perform basic cleaning and check table conditions.



**FIGURE 3.5. Use Case K+ solution specification**

Based on Use Case diagram it will be very useful to make a MoSCoW plan. The MoSCoW plan is a prioritization technique, not very widely used in IT, but still very helpful in any management and planning. MoSCoW is a prioritization method and assists teams to organize storycards according to the value from the customers' perspective (Agile Academy 2016). The idea of the MoSCoW plan is to group all the features possible for the solution into four groups: M – Must be, S – Should be, C – Could be and W – will not be implemented. The letters “o” are just used to make the abbreviation more memorable and human-readable. Obviously, all features in the group “M” are core features. Without them project cannot be used and released. The “S” are features which will be implemented immediately the core is built and are a bit less essential for the project, could be changed or adjusted, but are still important.

Features in “C” group are usually plans for the future development and features that are not critical and could be implemented, if there would be enough resources. The “W” group gathers ideas and suggestions which are possible to implement, but will not be implemented. This group is also important, because it helps to scope on a task without spreading resources on additional things. (Qiao 2009.)  
The MoSCoW plan of our case is shown in Figure X.

## M

- Table map with navigation and displaying of used\free tables
- Booking system
- Client data storage and interface to it

## S

- Gap handler
- Document former
- Workflow planner
- Service handler
- Understandable GUI

## C

- Analytic systems.
- Integration with the old system
- Sales statistics processing

## W

- Web interface
- Cloud integrations
- Remote booking
- Client Photo base

**FIGURE 3.6. MoSCoW plan**

### 3.4 Post-planning changes

As it usually happens in a real environment, many things are not possible to predict and plan. And it is a normal situation, when a plan gets post-additions during the development stage. In our case, as an example could be used an occasion with an old system, when there were two persons with the same First, Second and Middle names. This led to a harmful situation when one customer got all money from two tables and the other got nothing. As a conclusion from this situation, I modified the project and added features that might help in warning if such situation will happen ever again.

This required slightly modifying the database, adding two new fields to the customer line, and adjusting the code to handle this new information.

#### 4.     **PROTOTYPING**

Modern software development strongly depends on graphical interfaces. One system itself might not be new or perfect, but if the interface is done well, it might have a huge commercial success. The importance of clear information displaying and process controlling is well estimated and a lot of resources in the software development projects are usually spent on design, especially, if the product is made for a huge market group.

There are still many software solutions that have no graphical interface (controlled through a console) or with the interface that was done “just for make it done”. With the huge popularization of IT, PC utilization almost everywhere, the tendency is clear: typical users want to see something that they understand from the beginning, which is fast and convenient to use and clear in representation of resulting data.

Based on GUI development helps to understand the main requirements for the project, get customer point of view and to effectively increase the code development. I considered these benefits to be very important, and this was the reason for me to choose Interface-Driven development as the main strategy.

Modeling a graphical part of the software solution is called prototyping. A prototype is an early sample or model built to test a concept or process or to act as a thing to be replicated or learned from. Prototyping serves to provide specifications for a real, working system rather than a theoretical one.

The GUI itself is a good description of the business process. It is possible to describe the document\data flow based on GUI, understand the main operations, get the idea, how this GUI could help in the process and what is possible to improve.

The design process of any GUI could be divided into two parts: sketches and mockups. Sketches, also known as “Wireframes” are schematic and low-detailed models of the interface, usually showing just the basic object positioning, features and mechanics usually described in words. Usually wireframes are done to get the basic view and to predict the best solution, discuss concepts with customers and to create the basic code and database logic. Sketches do not need to be precise and are usually used for outlining positions and constructions. The Wireframe depicts the page layout or arrangement of the website’s content, including interface elements and navigational systems and how they work together. The wireframe usually lacks typographic style, color or graphics, since the main focus lies in functionality, behavior and the priority of content. (NEAUG 2013.)

#### 4.1. Sketches of the user interface

There is big variety of tools suitable for drawing sketches, starting from most of the graphical redactors and finishing with special tools created only for GUI design or hand-drawings. Sometimes first sketches are done by hand on paper, sometimes there are created directly as digital files.

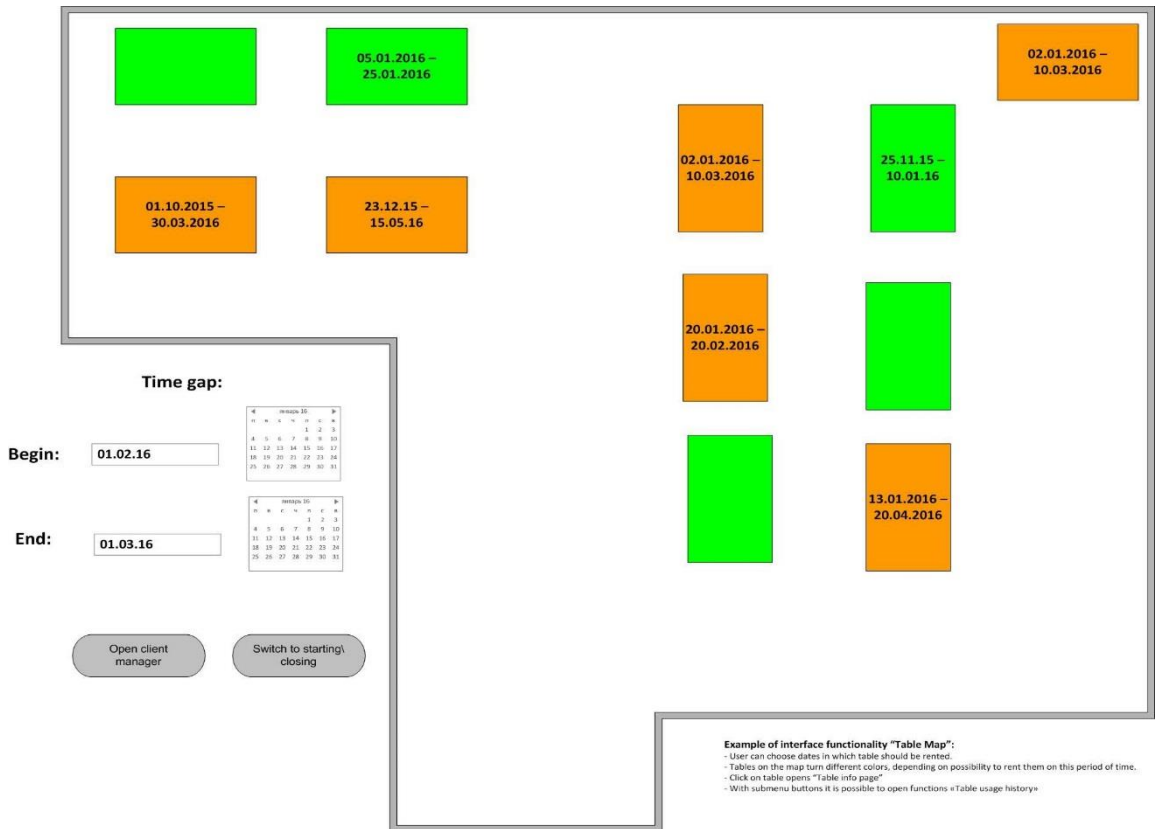
In this project most sketches are done in MS Visio as well as most of the diagrams. This tool allows building primitive graphical layouts and is suitable, familiar and convenient. It may lack some features, other professional programs may offer, but they were not considered important. MS Visio is an object-oriented drawing program, which allows drawing simple models, charts and schemes through graphical primitives (Quesenbery, Bachmann 2004).

The basic building blocks:

- Shapes and stencils: ranging from basic shapes and flowcharting to Windows UI and UML
- Templates: supporting consistent use and the reuse of elements throughout the prototype
- Backgrounds: containing common elements applied across screens

Figure 4.1 shows the first concepts of the program outlook, the example of the map with the tables, date manipulation and mode switching buttons. Color differentiation helps to add more information to the screen without adding more text: the green shows that the tables are free on the chosen date, orange means that tables are occupied. On booked tables there are the dates of their occupation. It was later decided to disable this feature, as the full map was not suitable for holding such amounts of text. One of the buttons refers to the client management menu, described later. Another was planned to switch color to differentiate the opening\closing the tables – another important feature. But, in discussions with client we found that client want this feature to look different. The client asked for pushing the opening\closing list into separate window. Each table also works as a button, opening a chosen table management window.

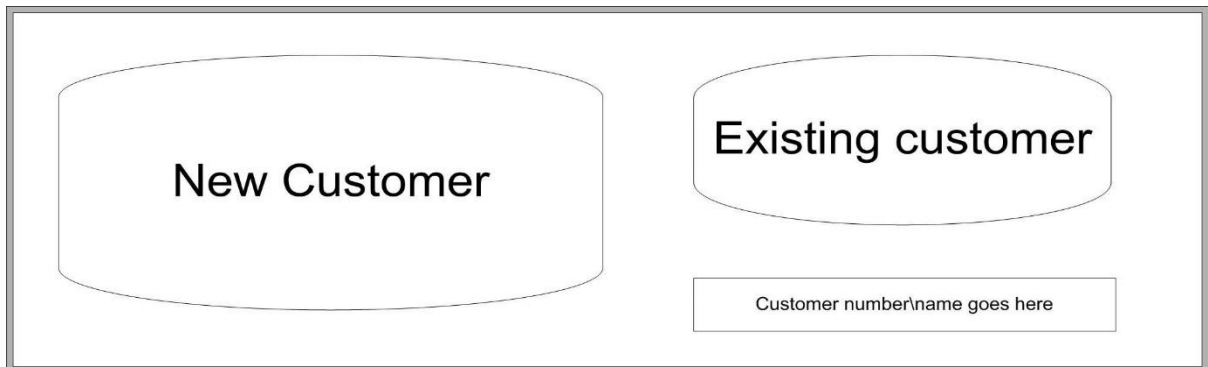
Later the client asked for additional color to differentiate long-term customers who rent a table for more than one month.



**FIGURE 4.1. The first concept**

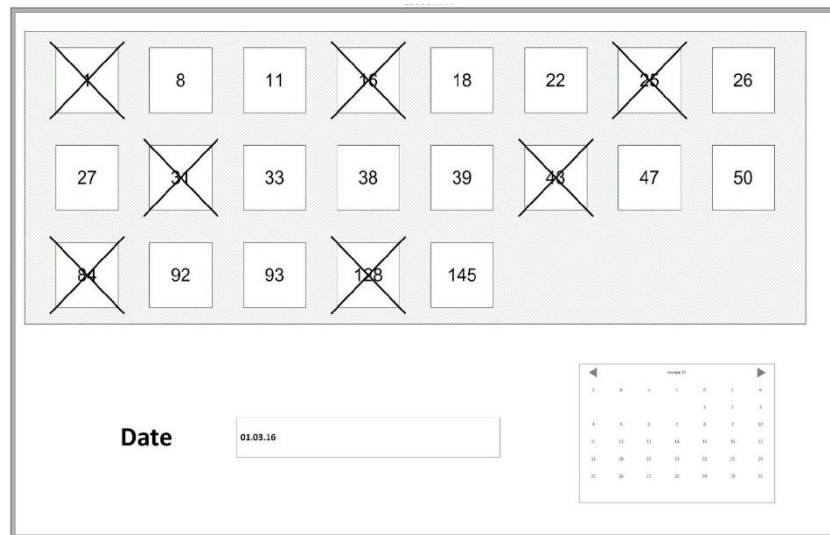
In our environment, as it was discovered, it is very important to make the system as simple, user-friendly and intuitive as possible. It was mentioned that it is better to make the appearance of the solution to look similar to the current used on paper mechanics. This influences many design decisions that may look a bit inconvenient or illogical in other situations.

I decided to make some parts of GUI which have no relations to current procedures and mechanics, minimalistic, so that it would difficult to get lost in them. The customer control screen is very simple and includes two options: to add a new customer or to change an existing one, finding the customer by name or number.



**FIGURE 4.2. Customer menu 1**

The client prefers to see the closing\opening table as a plate with table numbers in rows which can be clicked by adding a cross to them and mark as “done”. The same mechanics could be done right above the main map, but this design is a copy of the current procedure, used for managing table service workflow, so it was chosen as more preferable.



Closing tables window

**FIGURE 4.3. Open\close list**

The click on the table button on the main map opens the table management window, shown in Figure 4.3. In this window I placed a calendar of the table booked. Later it was decided to expand the calendar into three months, because one month is not enough to understand the situation. Here is also used color differentiation: orange – occupied days, green – free and blue – the chosen ones.

The customer defined a problem that requires a solution: gap manipulation, also mentioned in the text as “gap logic”. The current functionality allows customers to book a table only for seven or ten days, and also to prolong it for one day multiple times for additional costs. This leads into creating some kind of mechanics, which will help to select booking dates, so that the free days’ gap between the previous and next rents allows utilizing these days in the most efficient way, either with booking or with prolongation, living minimal number of days to waste. This “gap logic” will be described more detailed in next chapters.

It is also possible to open another window from the table window, showing all the previous customers who rented this table with their contact data. This might be useful, if someone needs to contact the current\previous customer in order to discuss some issues (Figure 4.5).



If dates are chosen and agreed with supervisor, it is possible to book a table from the table window. It should lead to the previously mentioned window customer adding where it is possible to choose a customer or create a new one.



FIGURE 4.4. Table menu

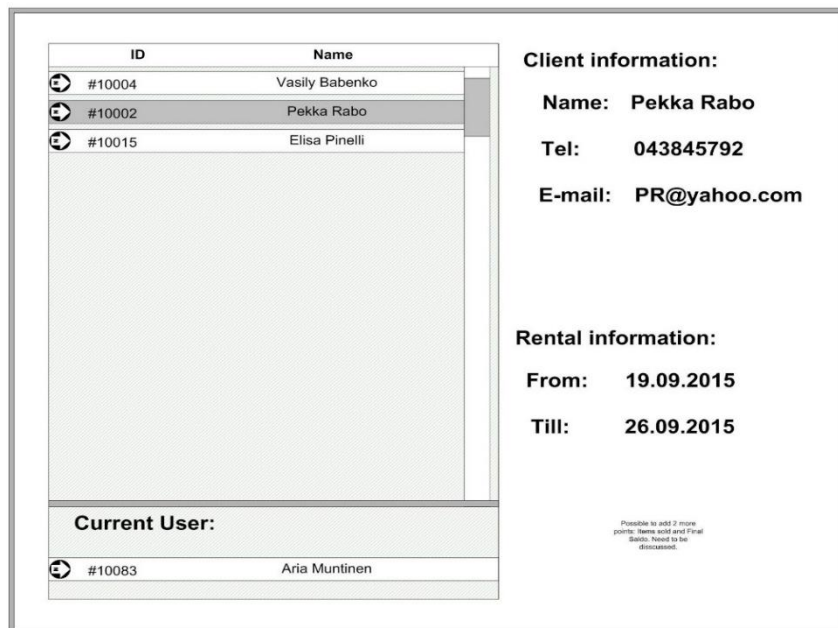


FIGURE 4.5. Customer menu 2

The “Create Client” window, shown in Figure 4.6, is the same as the client modification window, with the only difference that in the modification window data is populated (filled) automatically, and the client number is not generated. After adding\modifying data and its validation, it is possible to save it into the database or discard it by closing the window without saving.

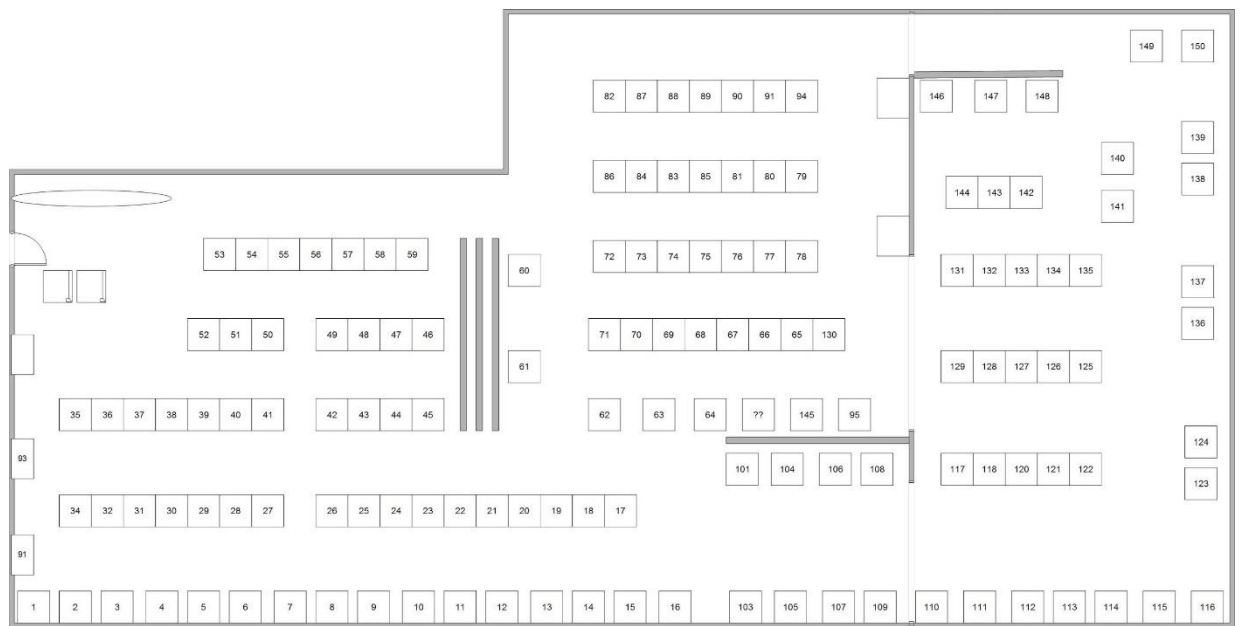
The image shows a software window titled "Client #10009". It contains five text input fields and two buttons. The fields are labeled as follows:

Field Label	Value
First Name	Vasily
Second Name	Babenko
Telephone	+7 898 09283741
E-mail	Temp.Pemp@gmail.com
Number of previous rents	0

At the bottom of the window, there are two buttons: "Save" and "Cancel".

**FIGURE 4.6. Adding a new customer menu**

The shop table map was designed the last, its appearance was taken from the current printed map. There was a decision to divide the map into three sectors to make all the views bigger, but the client denied this idea, preferring the full-map view as more convenient. The problem is in its size: it is big, including a huge number of elements (150 tables), which makes each element small, and allow only the table number to be placed inside the table space.



**FIGURE 4.7. The full map**

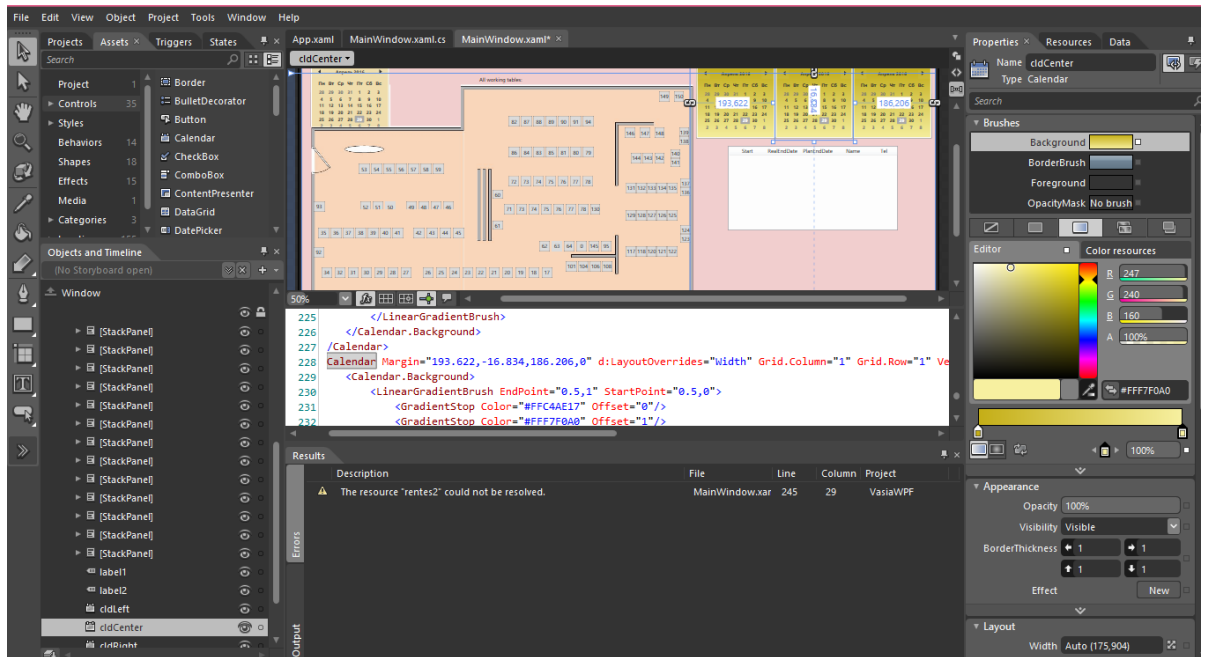
## 4.2 User interface mockups

After the wireframes were done and the main look was discussed and agreed, the more precise GUI model should be done. These models, also known as mockups, are mostly the closest to the final version of the GUI and contain all the details. Usually they are already working prototypes, with animations and transitions.

A huge number of projects, developed in the Microsoft environment, Visual studio and C# are done on build-in framework WinForms. WinForms is convenient and handy, but not flexible and beautiful enough. There is a huge number of various limitations in the design possibilities, so I have chosen another MS tool, developed exactly for the GUI design. This tool is MS Expression Blend, and it is a part of the MS Expression suite (Troelsen 2012). It is a powerful designer tool, utilizing declarative language XAML for graphic notation. XAML will be introduced in the next chapter.

Microsoft Expression Blend 4 is Microsoft's newest interactive design tool. It's intended for designers and developers who need to create user interfaces for rich Internet, desktop and mobile applications. And it offers tools that support the design of such applications, from the conception to completion

With Expression Blend, developer can even integrate graphics created in other design tools, such as Microsoft Expression Design, Adobe Illustrator, and Adobe Photoshop. The easiest source for integrating external graphic assets is Expression Design (Kosinska and MediaCarbon Inc. 2011).



**FIGURE 4.8. Expression Blend Gui**

Figure 15 shows the GUI of Expression Blend 4. As we can see, it combines elements, usually included in graphical redactors, like Adobe Photoshop and IDEs like Visual Studio.

#### 4.2.1 The actual versions of GUI

As planned in the wireframe prototype Interface was developed in the series of separate windows that are quite close to ones planned in wireframes. Figures 4.9-4.12 shows the most functional solutions.



FIGURE 4.9. Table map

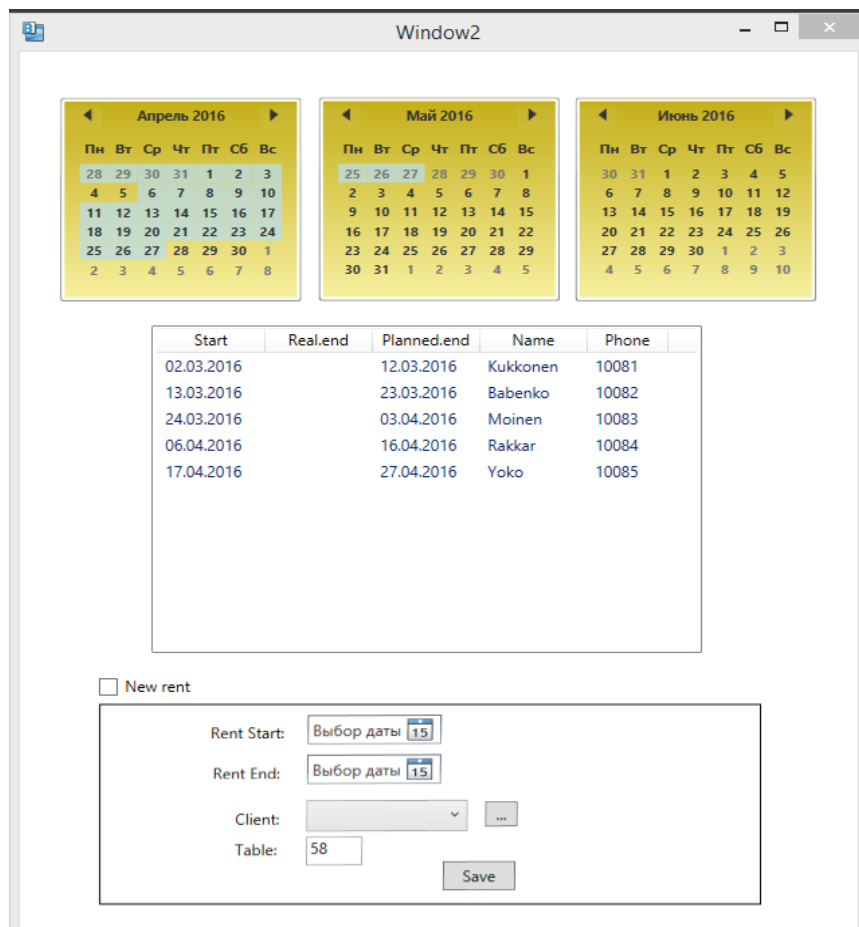
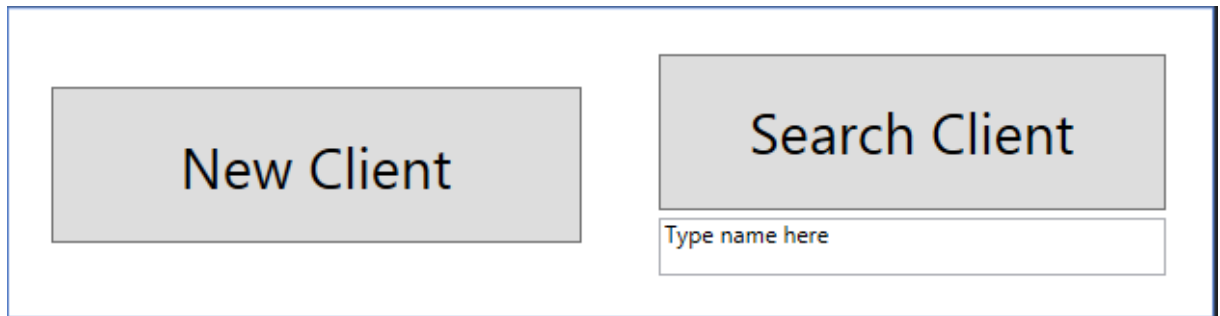


FIGURE 4.10. Table Rent manager



**Figure 4.11. Client manager start**

**Figure 4.12. New/change client window**

I have already discussed table map, table rent manager, client change\new windows and close\open planner.

The time and knowledge resources of the project were insufficient for deep and detailed mastering of the design techniques, so the GUI lacks fancy animations and graphical complexity. Moreover, as it was pointed in planning and prioritization phase, many other features should be realized first, and the minimalistic, but user-friendly GUI suits well without unneeded complications.

## **5. DATABASE DEVELOPMENT**

Term “Information Technology” point out, that subject is based on various technics of manipulation with information. Despite the fact, that information is abstract and immaterial, it can be described and understood in the same way, as any valuable material resource. Software solutions need data for operations, as machinery needs fuel, except data is not destroyed upon consumption. The information source for IT solution could be “outside” and come directly from the physical world – from user, sensors, other services, etc., but it also could be inside the solution, in information storage.

Storing information is a very important feature of any software solution. Information could be stored in different ways, serving different needs. Saved data could be stored for user, in human-readable, for internal usage, understandable for software or in combination. Any long-term stored information could be stored in roughly two different ways:

- Independent files (logs, documents, save files) – this files are independent, completed and have strict data format. They could be easily copied, read and used separately.
- Structured file systems (database, tables, libraries) – usually data in multiple files depends and connected with other data in files, information could be renewed, added or deleted.

### **5.1 Database in theory**

As an example, in database everything focused in manipulation with data, fast and easy access to a specific part, searches, derivations and operations with it. Control over database is usually done by special service – database server, and all other processes cannot get straight access to the stored data.

Database - is any collection of related data. In more strict way it is a database is a persistent, logically coherent collection of inherently meaningful data, relevant to some aspects of the real world (Robbins 1995).

In our case, the project is based on storing, manipulating and analysis of always changing and growing lists of customers, rents and events. Storing all data in files would be very ineffective, so database is the most useful and logical solution for it. The database part in this project, according to our strategy, is the last part of the software solution body.

As it will be mentioned, project is based on environment and tools provided by Microsoft. Selection of database server environment is also following this strategy, so the MS SQL Server 2008 was chosen. MS SQL Server – A database management system (DBMS), a collection of programs that enables users to create and maintain a database. A database management system (DBMS) is an aggregate of data, hardware, software, and users that helps an enterprise manage its operational data. The main function of a DBMS is to provide efficient and reliable methods of data retrieval to many users (Simovici 2011).

Data in a database of this type is usually stored in tables, aka entities. Each line in the table is an object, which is composed by number of attributes with predefined logical and variable type. For example: any line in table “Clients” must have attributes “ID”, “First name”, “Second name”, ”Middle name”, “Additional info”, “tel” and can also have attribute “e-mail”. Each attribute has defined logical meaning, so in all lines this attribute will contain same information meaning, and also each attribute has constant data variable type, like number or amount of characters, which will help programs to operate with data.

Many tables could be connected with each other in connection one-to-many, by same attribute. This mechanics is called Primary-Foreign Keys. Primary Key (PK) is an attribute is one table, which must be unique in all lines. Usually it is done by special ID attribute which is incremented and assigned automatically to each line, when it is created. Foreign Key (FK) is an attribute in any other table, which has same logical



meaning and data type. Many tables with Foreign Keys can be connected to one with Primary Key, and this connection is used, when there is a need to find more information about the object. Then FK attribute is used as identification to search through the PK table to a one line, containing data about the object.

For example: Table “Rent” contain attribute “ClientID”. Table itself contain no data about clients, who rent table, except their ID. ID could be same in different lines, as one client can rent multiple times. So “ClientID” is FK and it is connected to the table “Clients” where there is same attribute, but unique in each line, and used as PK. Then when there is a need to know more data, about who is actually renting, this connection allows to gather needed data from another table.

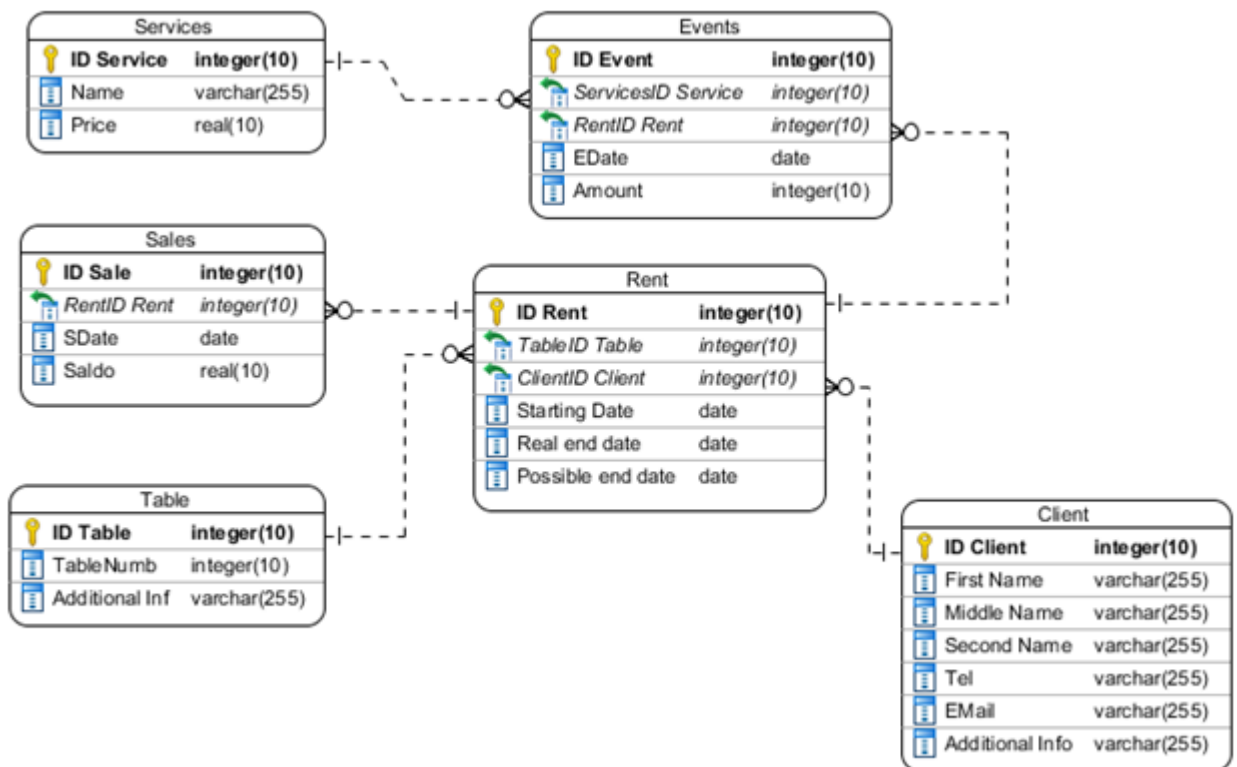
There are two main types of structuring data On-line Transaction Processing(OLTP) and On-Line Analysis Processing (OLAP) which are different in their purpose: OLTP is mostly used for collecting data and accessing, when OLAP is designed for analysis and processing. Official explanation of OLTP is a class of systems that facilitate and manage transaction-oriented applications, typically for data entry and retrieval transaction processing (Lakshmi, Razia 2013) and for OLAP is computer processing that enables a user to easily and selectively extract and view data from different points of view (Indstuds.com 2015). These terms are not strict and database can be used more or less in both ways. In this project database is closer to the OLTP logic, because main feature are based on collection and structuring data for its further easy access.

As it was mentioned, database is maintained and operated by the server, and only server has access to it directly. All operations with the server are done through special language. Most of operations are done through executing short scripts – “queries”. The language is usually SQL or its derivatives, in our case it is T-SQL (Transact-SQL)

SQL – (Structured Query Language) is a database computer language designed for managing data in relational database management systems (RDBMS). SQL, is a standardized computer language that was originally developed by IBM for querying, altering and defining relational databases, using declarative statements.

Also Database server environment can be used as a tool for DB planning, as it allows to create DB graphical models, illustrating DB structure and all entity's connections, which helps to understand and optimize project.

## 5.2 Project database implementation



**FIGURE 5.1. Database model**

In this project the table structure was rather compact and using four main tables: “Clients”, “Rent”, “Events” and “Services”. The first models of DB also included table “Tables” (Figure 5.1), which contained attributes “ID”, “Table Number”, “Locked” and 4 attributes with coordinates X and Y for button positioning. Because table number was found unique, even not in straight logical order, and positioning gesture control system was considered ineffective, this table was deleted from final model. Also model contains table “Sales” which is currently not fully used in project, but was designed for further development and implementation of some analytical features, mentioned in MoSCoW diagram in position of C-could.

Main data-flow is based on four mentioned tables:

“Clients” – Collecting data about all unique clients, that have done any rent.

Attributes: “Client ID” – auto assigned and incremented, “First Name”, “Second Name”, “Middle name”, “Additional info”, “Tel”, “E-mail”. From all attributes, only ID is integer number and it is automatically assigned increment, used as PK, other are number of symbols (varchar), and only “E-mail” could have value “Null” e.i. no value – all other must be filled in order to save object. Telephone was done in symbolic way to allow spaces and “+” signs.

“Rent” – Collection of all table rents. It includes attributes “Rent ID”, “Client ID”, “Table ID”, “Starting date”, “Planned End Date”, “Real End Date”, where “Rent ID” is automatic increment, used as PK, “Client ID” is used as FK to table “Clients”, and all fields except “Real end date” are required.

“Services” – Fixed collection of all additional services, available to the customer. This table is not designed to be changed, as there is no plans to add any new services. Each object “Service” consists of “Service ID”, which is a PK, “Name” and “Price”. There are 6 objects in this table.

“Events” – Collect any data about used services, mentioned in “Services” table.

Consists of “Rent ID”, “Service ID”, “Amount”. “ID” fields are FK connecting table to table “Rent” and “Services” accordingly.

### **5.3 Stored procedures**

Besides storing data, DB server allows to store inside user procedures for remote calling. A Stored Procedure is a precompiled collection of SQL statements. In a stored procedure you can use if sentence, declare variables, etc. (Halvorsen 2016).

This is very important feature, as SQL language is declarative and it is not flexible in creating queries as most programming languages and many features in query, such as loops and logical operations are difficult to organize. Procedures allow to solve this limitations, packing inside multiple queries and organize data-flow with logical

functions. As it allows to manipulate and work with data directly inside server, it also significantly improves traffic usage, security and solution operation speed, because only few data is transferred between endpoints – program and server. Program calls remote procedure on a server and sends parameters for this procedure to operate, server calculates and derives data from database and sends only the answer. There is no need to transfer all the data all the time and calculate it on program machine – this can save resources and protect data from massive sniffing, if there is someone trying to steal data.

In our case this is not so important, because both program and database will be operated on same machine, but this way of organization still helps to make project more clear and stable.

To design and analyze stored procedures, I am creating “SQL procedure model” – a list which combines the main planned operations with database and possible solution with SQL procedure.

**TABLE 2. Project SQL procedures model**

Query line	Query variables (parameters)	Description (Business Logic)
select count(*),count(distinct [id table]) from rent where [id client]=@client	Int @client	Counts all rents and counts of different tables for the client
select distinct * from clients order by [second name]	–	Returns all objects from table Client, sorted by second name (list of all clients).
select * From Clients where [ID Client] = @client	Int @client	Returns full information of client with matching ID
select * From Rent where [ID Rent] = @rent	Int @rent	Returns full information of rent with matching ID

<pre>select services.name from rent,services,events where rent.[id rent]=events.[id rent] and services.[id service]=events.[id service] and rent.[id rent]= @rent</pre>	Int @rent	Returns list of services are provided during rent matching ID
<pre>select sum(price*amount) from rent,services,events where rent.[id rent]=events.[id rent] and services.[id service]=events.[id service] and rent.[id rent]= @rent</pre>	nt @rent	Counts total for prices of additional services.
<pre>select services.name, sum(price*amount) from rent,services,events where rent.[id rent]=events.[id rent] and services.[id service]=events.[id service] and [starting date]&gt;=@dt1 and [starting date]&lt;=@dt2 group by services.name</pre>	Date @dt1 Date @dt2	Returns list of services and their total price in selected time interval.
<pre>select [Starting date],[Real end date],[Possible end date],[Second name],[id rent] from rent, clients where rent.[id client]=clients.[id client] and [id table]=@table</pre>	Int @table	Return renting periods and names for selected table.
<pre>select * from Rent where [Starting Date]&lt;=@dt and [real End Date] is null and [id Table]=@table</pre>	Date @dt Int @table	Checking, whether table is occupied for selected date, or not.
<pre>select tablenumb from rent,tables where tables.[idtable]=rent.[id table] and [real end date] is null and [possible end date]&lt;= @dt</pre>	Date @dt	Get a selection of objects from table “Tables”, which should close today.
<pre>Insert into Clients ([First Name],[Second Name],[Middle Name],[Additional Info],[Tel],[E-mail]) Values (@FName,@SName,@MName,@Note, @Tel,@Mail)</pre>	All parameter s are nvarchar (string)	This will create new object in Clients table, with input parameters.

Insert into Rent ([ID Client],[ID Table],[Starting date],[Possible End Date]) Values (@Client,@Table,@Start,@End)	@Client, @Table – int, @Start, @End – date	This will create new object in Rent table, with input parameters.
Insert into Events ([ID Rent],[ID Service],Amount,Date) Values (@Rent,@Service,@Amount,@Date)	@Rent, @Service, @Amount – int @Date - date	This will create new object in Events table, with input parameters.

SQL-queries in this format are invariant in the meaning that they could be implemented in any programming language and any design paradigm (for example, on the WEB-platform when coding in PHP, or Silverlight). On the other hand, they determine the database visualization application interface and respond to the main question of design: “What data and in what format the program should provide the user?”

The most important queries from the table were used in the code, but some show the database capacity and focused on the following releases of the program. Not all SQL queries were converted to the procedures, as it was easier to use without procedure calls. Procedure storing might also cause resource wastage, as it makes code development less obvious.

To test procedures and prototype’s functionality, I populated database with some test data. The challenge in this process is that there should be big amount of data in database to test most of the features and it is not trivial task to generate it for a single person, as data should be variative and cover most possible situations.

Currently populating the database with testing data is shown in Figure 5.2.

ID Rent	ID Client	ID Table	Starting date	Possible End Date	Real End Date
10005	10004	4	2015-09-23 00:...	2015-10-25 00:...	2015-10-26 00:...
10006	10000	10	2015-10-01 00:...	2015-10-20 00:...	2015-10-20 00:...
10007	10003	6	2016-01-03 00:...	2016-01-16 00:...	NULL
10008	10002	15	2016-01-05 00:...	2016-01-20 00:...	2016-01-20 00:...
10009	10002	5	2016-01-21 00:...	NULL	NULL
10010	10005	4	2016-01-13 00:...	2016-01-27 00:...	2016-01-28 00:...
10011	10001	3	2016-01-08 00:...	2016-01-22 00:...	2016-01-29 00:...
10012	10003	13	2016-01-15 00:...	NULL	2016-01-21 00:...
10013	10003	13	2016-01-29 00:...	NULL	2016-02-04 00:...
10014	10002	8	2016-01-18 00:...	NULL	2016-01-27 00:...
10015	10000	42	2016-01-16 00:...	NULL	2016-01-29 00:...
10016	10001	21	2016-01-15 00:...	2016-01-21 00:...	2016-01-23 00:...
10017	10000	144	2016-01-02 00:...	2016-03-08 00:...	NULL
10018	10005	65	2016-01-23 00:...	2016-02-01 00:...	2016-02-03 00:...
10019	10001	21	2016-01-27 00:...	2016-02-02 00:...	2016-02-02 00:...
10020	10006	8	2016-02-01 00:...	NULL	2016-02-07 00:...
10021	10005	21	2016-01-27 00:...	NULL	2016-02-02 00:...
10022	10007	42	2016-02-03 00:...	NULL	2016-02-12 00:...
10023	10006	65	2016-02-03 00:...	NULL	2016-02-10 00:...
10024	10003	13	2016-02-09 00:...	NULL	2016-02-15 00:...
10025	10004	8	2016-02-10 00:...	NULL	2016-02-16 00:...
10026	10005	21	2016-02-14 00:...	NULL	2016-02-20 00:...
10027	10007	13	2016-02-16 00:...	NULL	2016-02-25 00:...
10028	10006	65	2016-02-18 00:...	NULL	2016-02-24 00:...
10029	10002	8	2016-02-26 00:...	NULL	2016-03-03 00:...
10030	10007	42	2016-02-25 00:...	NULL	2016-03-02 00:...
NULL	NULL	NULL	NULL	NULL	NULL

**FIGURE 5.2. Example of populated data**

## 6. CODING

This chapter is dedicated for the coding part of the project, the “body” of the solution, its main logic and mechanics. The main code of the program is written on two main programming languages: C# and XAML.

### 6.1 Programming environment

C# is a modern programming language for the development of software applications, created and developed by Microsoft, together with the .NET platform. There is highly diverse software developed with C# and on the .NET platform: office applications, web applications, websites, desktop applications, mobile applications, games and many others. C# is a high-level language that is similar to Java and C++, and, to some extent, the languages like Delphi, VB.NET and C. All C# programs are object-oriented. They consist of a set of definitions in classes that contain methods, and the methods contain the program logic – the instructions which the computer executes. (Nakov 2013.)

XAML (short for Extensible Application Markup Language and pronounced “zammel”) is a markup language used to instantiate .NET objects. Although XAML is a technology that can be applied to many problem domains, its primary role in life is to construct WPF user interfaces. In other words, XAML documents define the arrangement of panels, buttons, and controls that make up the windows in a WPF application.

#### **Programming paradigm in WPF (Windows Powerful Foundation).**

The Windows Presentation Foundation (WPF) is a modern graphical display system for Windows. It is a radical change from the technologies that came before it, with innovative features such as built-in hardware acceleration and resolution independence. Application Development Technology at the WPF platform, is, in my opinion, the most suitable to achieve the goals (MacDonald 2012). Its main features are as follows:



- Programming Tools are completely based on the functionality of .NET framework.
- Application (screen forms) interface are developed and implemented by means of a declarative language XAML, a derivative of XML, and it has the ability to Data Binding and means of interaction with object-oriented languages.
- The implementation of algorithms and business logic is achieved by means of C # (or VB.NET).
- Full implementation of the charts is based on DirectX technology, and uses hardware acceleration that is independent of the screen resolution.
- Lock screen visual components is not coordinate a "container", which greatly simplifies the layout and scale.

The interaction between the markup codes (XAML) and algorithmization (C #) in the following manner by the use of uniform named classes and Data Binding.

## **6.2 Coding process and code examples**

XAML is not usually redacted by hand, in my solution I use for XAML redaction\generation MS Expression Blend tools. But the code itself is easily readable and could be redacted manually. Person, who knows HTML (very popular basic web-site language) may find some similarities, as both languages are declarative and use tags for placing elements.

Some sample code from the project, for representing table, containing DB query result information is shown in Code 1.

```

<ListView x:Name="lv1" Margin="16,48,16.288,108" SelectionChanged=
    "ListView_SelectionChanged">
    <ListView.View>
        <GridView>
            <GridViewColumn DisplayMemberBinding="{Binding SName}"
Width="80"
                Header="Second Name"/>
            <GridViewColumn DisplayMemberBinding="{Binding FName}"
Width="80"
                Header="First Name"/>
            <GridViewColumn DisplayMemberBinding="{Binding Tel}"
Width="60"
                Header="Tel"/>
            <GridViewColumn DisplayMemberBinding="{Binding Email}"
Width="120"
                Header="E-mail"/>
            <GridViewColumn DisplayMemberBinding="{Binding IDClient}"
                Width="50" Header="ID Client"/>
        </GridView>
    </ListView.View>
</ListView>

```

### CODE 1. XAML representation of data grid

As XAML is used for the front-end of the solution, representing the GUI, the C# code forms the main logic and data operations inside solution. The code for previous XAML table on C# which represents data manipulation is shown in Code 2.

```

public class Client
{
    public String SName { get; set; }
    public String FName { get; set; }
    public String Tel { get; set; }
    public String Email { get; set; }
    public int IDClient { get; set; }
}

```

### CODE 2. C# equivalent of table object

And some code is needed to connect these two parts together to make the table display the Clients class correctly. It is also on C# and can be found in Code 3.

```

while (reader.Read())
{
    Client c1 = new Client
    {
        IDClient = (int)reader[0],
        FName = reader[1].ToString(),
        SName = reader[2].ToString(),
        Tel = reader[3].ToString(),
        Email = reader[4].ToString()
    };
    lv1.Items.Add(c1);
    mit++;
}
reader.Close();

```

### CODE 3. Reading data from DB to object

This code will read data stream from the database answer and write it inside the table which will then represent it. It is also possible to read backwards from the table, when one of the elements is chosen by the user. The code on C# for handling such event can be found in Code 4.

```

private void ListView_SelectionChanged(object sender,
                                     System.Windows.Controls.SelectionChangedEventArgs)
{
    Label1.Content = "Number of rents for Client " +
        (lv1.SelectedValue as Client).SName.ToString() + ": " +
        GetAllRents((lv1.SelectedValue as Client).IDClient).ToString();
}

```

### CODE 4. Reading user's input from data grid

This code fragment writes all the data from the table line in the instance of the class Clients (the GetAllRents function which counts all rents for one client).

The part of the code realizing business-logic and written in Visual Studio is mostly related to algorithms and the data handler, because some parts of the code which declare screen forms are also written in Expression Blend. The important part of the program is the SQL handler class which is responsible for working with database and exists as a mediator between the interface and DB. As it can be seen in the class

diagram, this class includes methods that look similar to each other and perform same tasks: Get, Change and New. I have written a base methods for each task that could be used with each element (Client, Rent, etc.) with minimal alterations.

Each method requires opening a connection to a database and uses the fragment of code shown in Code 5.

```
public SqlConnection Connect()
{
    SqlConnection conn = new SqlConnection(connStr);
//connStr is defined from outside
    try
    {
        //try to open
        conn.Open();
    }
    catch (SqlException se)
    {
        if (se.Number == 4060)
        {
            //Error message
            Console.WriteLine("No such DB");
            //close
            conn.Close();
        }
    }
    finally
    {
        Console.WriteLine("Connected")
    }
    return conn;
}
```

#### **CODE 5. Database connection handler**

To get any needed data I use the code shown in Code 6 (based on getting Rent)

```

public Rent GetRent(int RentNum)
{
    Rent Rent = new Rent();
    SqlConnection conn = Connect();
    SqlCommand cmd = new SqlCommand("Select * From Rent where [ID Rent] = @ID",
conn);
    SqlParameter param = new SqlParameter();
    param.ParameterName = "@ID";
    param.Value = RentNum;
    param.SqlDbType = System.Data.SqlDbType.Int;
    cmd.Parameters.Add(param);
    SqlDataReader dr = cmd.ExecuteReader();
    dr.Read();
    Rent.ID = int.Parse(dr[0].ToString());
    Rent.ClientID = int.Parse(dr[1].ToString());
    Rent.TableID = int.Parse(dr[2].ToString());
    Rent.StartDate = DateTime.Parse(dr[3].ToString());
    Rent.PosEndDate = DateTime.Parse(dr[4].ToString());
    Rent.RealEndDate = DateTime.Parse(dr[5].ToString());
    if (conn != null)
    {
        conn.Close();
    }
    conn.Dispose();
    return Rent;
}

```

### **CODE 6. Getting an object from the table "Rent"**

As Code 6 shows, I use the parameter handler, not the SQL string modification, as it works faster and in a more stable way, but also require longer code writing.

For creating a new entity the following logic shown in Code 7 is used (based on creating a new rent).

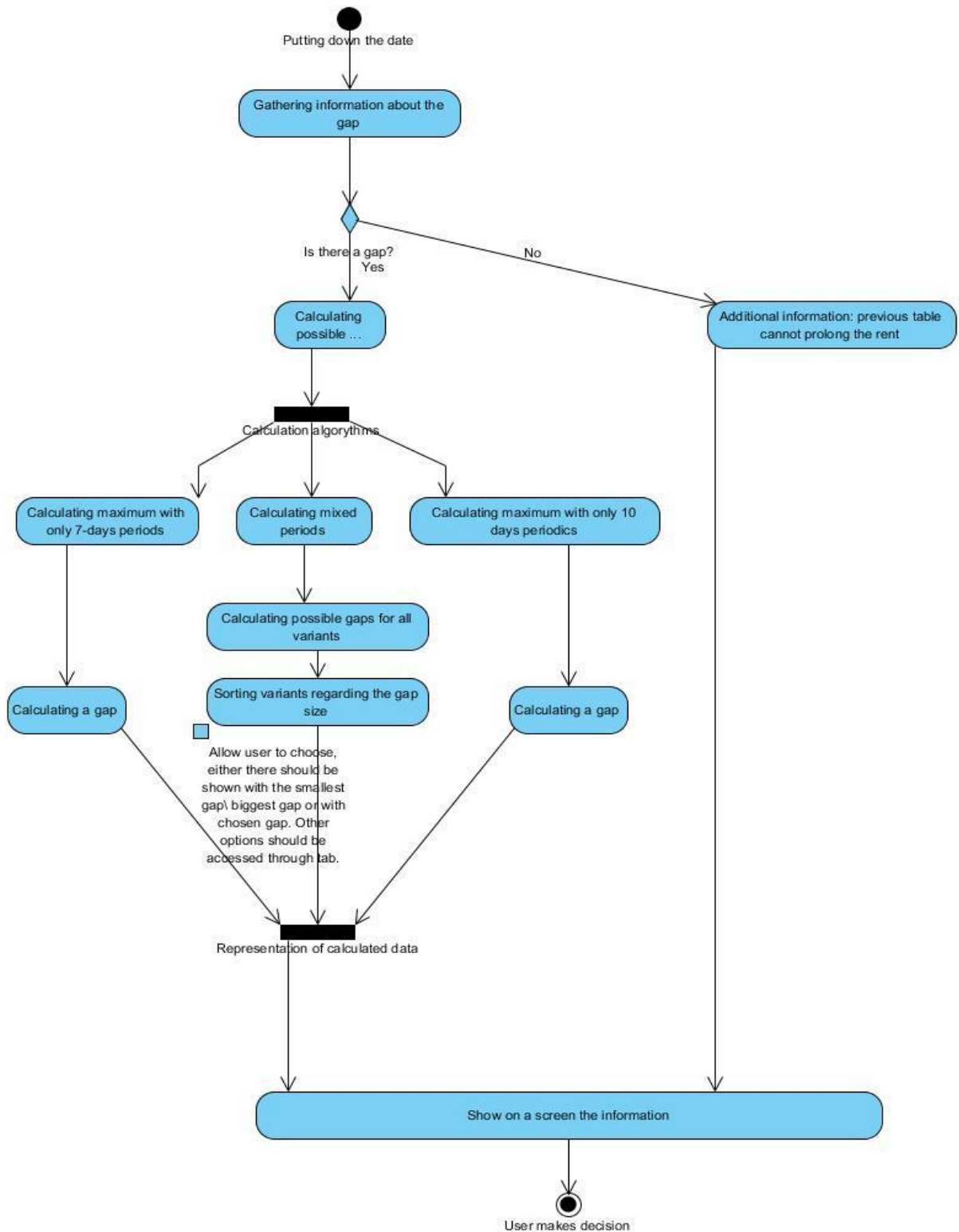
```

public void NewRent(int IDClient, int Table, DateTime StartDate, DateTime PlannedEndDate)
{
    SqlConnection conn = Connect();
    Console.WriteLine("Creating dataset");
    SqlCommand cmd = new SqlCommand("Insert into Rent" + "([ID Client],[ID
Table],[Starting date],[Possible End Date]) Values (@Client,@Table,@Start,@End)", conn);
//new object of class SqlParameter
SqlParameter param = new SqlParameter(); //parameter name
param.ParameterName = "@Client"; //parameter value
param.Value = IDClient; //parameter type
param.SqlDbType = System.Data.SqlDbType.Int;
//sending parameter to SqlCommand
cmd.Parameters.Add(param);
param = new SqlParameter();
param.ParameterName = "@Table";
param.Value = Table;
param.SqlDbType = System.Data.SqlDbType.Int;
cmd.Parameters.Add(param);
param = new SqlParameter();
param.ParameterName = "@Start";
param.Value = StartDate;
param.SqlDbType = System.Data.SqlDbType.DateTime;
cmd.Parameters.Add(param);
param = new SqlParameter();
param.ParameterName = "@End";
param.Value = PlannedEndDate;
param.SqlDbType = System.Data.SqlDbType.DateTime;
cmd.Parameters.Add(param);
try
{ cmd.ExecuteNonQuery(); }
catch
{ Console.WriteLine("Error on writing"); }
if (conn != null)
{ conn.Close(); }
conn.Dispose();
Console.WriteLine("Operation complete");
}

```

### CODE 7. Creating a new object in the table "Rent"

One difficult algorithmic problem was related to the gap calculation helper which should warn users about possible gap usages and gap day wastages. The graphical model of the algorithm can be seen in Figure 6.1.



**FIGURE 6.1. Gap analysis algorithm model**

The class diagram is one of the methods to model code structure and basic code logic it is very useful, as it could be converted to a working code canvas, if done in VS. The class diagram of the project is shown in the following Figure 6.2.

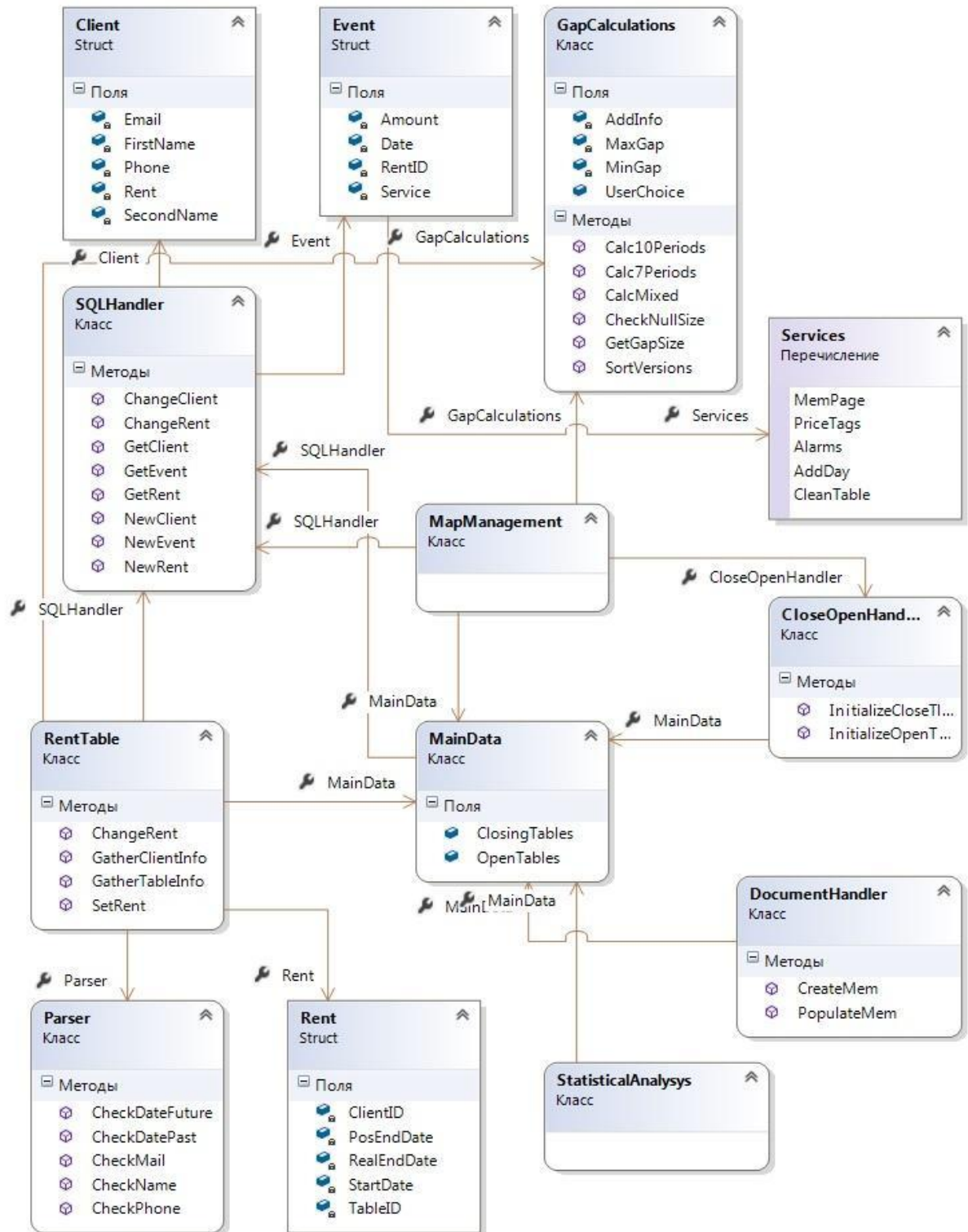


FIGURE 6.2. Class diagram



## **7. TESTING AND IMPLEMENTATION**

Software development is a complex and long, creative process which cannot be fully automated. Because of that, there is always a place for human-factor and chance for error. Many errors are made in the coding part, but they could also be in any other parts. It is very important to make error-free parts that are anyhow connected to storing and securing data, as failure in the parts may cause the biggest losses to the business.

### **7.1 Modern software testing**

To prevent or find and eliminate errors in the project developers use various methods and tools during the whole development process. Good planning and modeling is one of the first error-preventive methods, allowing predicting and correcting some realization errors, before they are done. During the coding phase a huge numbers of errors could be highlighted or corrected by IDE, if it is complex enough. Also, IDE is a perfect too for continuous error correction, also known as debugging. The main method of finding and correcting errors and mistakes during development phase is testing. There are many testing techniques that could be used during the different phases of development.

Some people may think that testing is done afterwards, when the project is done and it is ready to use. This is not correct, as it is possible to test a solution continuously during development. Many complex IDEs provide developers various tools to make the testing process easier, faster and more detailed.

Roughly, testing can be divided into three main parts: White-box testing, Black-box testing and Gray-box testing. White-box testing (later WBT) is when the tester has access to the internal data structures and algorithms including the code that implement them. (Williams 2006 A.)

White box testing methods include:

- API testing (application programming interface) - testing of the application using public and private APIs

- Code coverage - creating tests to satisfy some criteria of code coverage (e.g., the test designer can create tests to cause all statements in the program to be executed at least once)
- Fault injection methods - improving the coverage of a test by introducing faults to the test code paths
- Mutation testing methods
- Static testing - White box testing includes all static testing

Black-box testing (later BBT) treats the software as a "black box" without any knowledge of the internal implementation (Williams 2006 B.). Black box testing methods include:

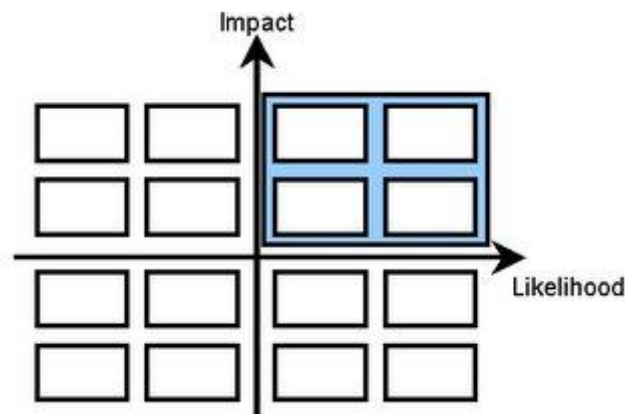
- equivalence partitioning,
- boundary value analysis,
- all-pairs testing,
- fuzz testing,
- model-based testing,
- exploratory testing and
- specification-based testing.

Grey Box Testing involves having knowledge of internal data structures and algorithms for the purposes of designing the test cases, but testing at the user or black-box level. The tester is not required to have full access to the software's source code. Grey box testing may also include reverse engineering to determine, for instance, boundary values or error messages. In this chapter I will mostly write about WBT and BBT, and how they were used in the project.

Moreover, one of the development strategies is based on testing. Test-driven development is based on writing tests before the project itself. This strategy requires the most strict and accurate planning, as all features, requirements and code logic should be defined precisely. Tests are developed first, based on plans, with the presumption that if a project satisfies all tests, it is ready. Then, the project is developed in the framework of tests, in the shortest way, just to make it go through all tests. Everything is working with the idea that if it works fine with tests, there is no

need to make more. (Quality Tree Software, 2008) This strategy is difficult, but can bring results with fewer resources, if the planning was good. Test-driven development is well known in theory, but in my experience, I have seen no project based on it.

In the software testing planning is considered as much important, as in all other parts of the project, because it should be done cost-effectively. Badly-planned testing can cost an enormous amount of resources and give useless results. The most popular strategy of testing planning is Risk-driven testing. Following this strategy, developers should characterize before testing all the parts of the project with two characteristics: probability of error and severity of error. The probability describes the chance of error in any part, and the severity describes the area of impact and losses, if an error occurs. On Figure 7.1 we can see the analytic model of the process, when each item is evaluated by mentioned two criteria, and placed on the two-axis diagram. The evaluation is empiric, and if Impact can be predicted logically, based on code needs, main tasks and business relations with different testing items, the Likelihood usually could be predicted only empirically based on evaluation of code developer. It is possible to predict likelihood based on code size – the bigger class of method is, the more chance to make mistake, but usually it is evaluated by developer work experience.



**FIGURE 7.1. The Risk analysis.**

Obviously, first in this approach will be tested items, placed in right top corner, the more right and top, the higher priority. This allows finding and gathering most of possible critical errors. After testing the most critical part, the strategy either might go to prioritizing of testing the most likelihood items or to most critical items left. It all depends on the project specifications and project manager preferences.

The code should be tested all along the development, not in the release states, as the basic formula of error correction cost tells, that correction costs of error multiplies by two each stage of development, that was after making error. That means that error, made in the beginning will be very difficult to resolve in the end of development, as many other parts of the project will be affected.

The code itself could be tested in ongoing coding process by several methods. First – reviews and inspections, either by developer, or by colleagues and third party professionals. Self-evaluation is normal and essential, but usually ineffective, as it is very difficult to see mistakes in one's own code. Peer reviews and inspections may vary in parameters such as anonymity, amounts of code viewed, communications between peers etc. Second method is testing pre-versions in test environment. Such as inserting testing console for method execution and debugging info displaying. This method is popular, but also has low-efficiency, as it allows only briefly test usability, based on empirical feeling “working\not-working”. Usually debugging console is still left in the project helping to debug and monitor processes in application real-time.

Another ongoing testing method is Unit Testing. Unit testing is one of the levels of testing which go together to make the “big picture” of testing a system. It complements integration and system level testing. Unit testing environment and tools are usually provided by modern IDE and their add-ons, including automatic testing construction, execution, result gathering and other analytical support. Unit test is a small script, usually written in the same language that takes a part of the program, executes it separately, with predefined inputs or range of inputs, and evaluate it based on outputs. It is possible to make Unit tests react on exact correct output, range or approximation, or negative reaction of method, such as throwing exception. If test gather output, which was expected, it returns information that test was passed with success. In any other situation, the test will fail. Based on unit tests, it is possible to improve, accelerate and automatize testing process. (Parkin 1997.) However, despite the fact, that unit testing can be used with any code, Unit testing is not convenient to use with GUI testing.

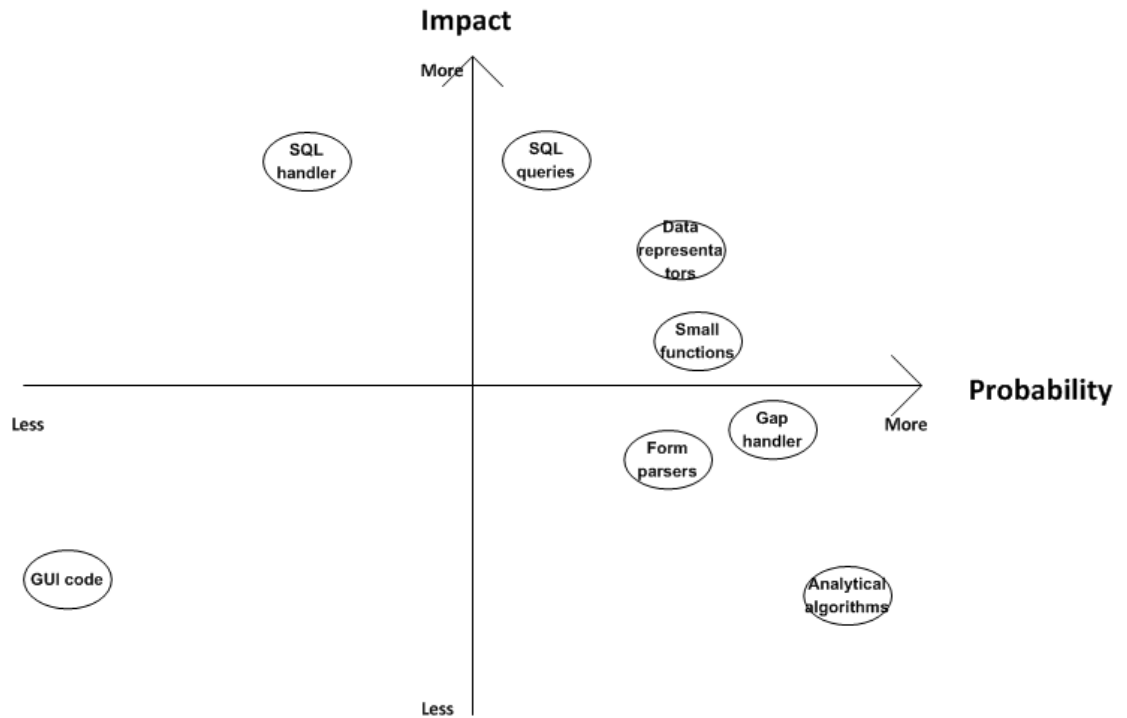
One of the important value of testing is code coverage. It means how many lines of code was used on testing. Theoretically it should be 100%, as unused lines of code are garbage and should be deleted or revised. However, even 100% of code coverage not giving 100% warranty from mistakes, as it may cover only some entries on loops or use not the complete logical paths. More adequate, but also more resource consuming is path coverage, which means covering all possible paths of code. The automatic testing environment calculates both measurements. In real testing situation, the 100% path coverage is not needed, as code usually contain loops and reused parts of code that are too difficult to test with all possible inputs. That leads us to the idea of balance of maximum useful coverage with minimum resource costs. This could be achieved by usage of different input-aggregation methods, such as boundary value analysts.

Boundary value analysis is a testing method focusing on manipulating and analyzing procedure input variables and applying testing limitations. In other words, it helps to test the most critical things with least resource costs, leaving all similar test inputs, which theoretically should work as tested inputs, untested. The idea of this method is not very complex. Tester needs to find a solid line of input variables from main pool of all possible inputs, where each variable is logically same as the others in line. For example incremental arithmetical sequence, or group of strings with same length. Then tester finding boundaries of this variable group and test only few inputs of boundaries and one randomly chosen input inside group. It is assumed, that all other inputs in the group will proceed with the same or predictable results. This brings huge resource savings with little efficiency loss. (Blake 2007.)

## **7.2 Project testing methods**

In my work, I faced huge time and people resource insufficiency, which led me into shortening and suppressing full testing process. As I worked in a team of one person, there was no possibility to make reviews and inspection in other way as by myself, and as it was mentioned, such inspections give lower results, as usual.

I made risk analysis diagram, which could be seen on Figure 7.2. It is fully empirical, I assume, that solution, developed for handling data must have no errors in code related to data storing and retrieving, because incorrectly stored data in this case can cause value losses. I am also confident with code for GUI as it was mostly generated by Visual Studio and Expression blend. I am finding complex analytical algorithms difficult to write and analyze, so I gave them very high probability of error, but as they were mentioned and additional and supporting feature with low priority, I assumed that errors in that part will make very small impact. Form parsers are usually very difficult to test even with boundary analysis methodic, as they usually contain huge variety of combinations and fields, and usually do not critical. For example, in my project fields for phone number and e-mail are just “strings” or text fields, as I wanted to allow more space for maneuvers, do not plan them as proceeded somehow else, except human-readable form and do not wanted to waste resources on testing their parsers. In fact, I choose as a strategy to test the most critical features – SQL related and small work functions.



**FIGURE 7.2. The Debug Console.**

I used testing console for brief testing and bug tracking.

The Debug Console is a separate project with references to the code library, and the whole program can be easily written in five minutes or less like that:

```
namespace DebugConsole
{
    class Program
    {
        static void Main(string[] args)
        {
            string Name;
            Console.WriteLine("Enter client Name");
            Name = Console.ReadLine();
            ProjectKLib.SQLHandler SQL = new SQLHandler();
            int ID = SQL.getClientID(Name);
            Console.WriteLine("Client ID is " + ID.ToString());
            Console.ReadLine();
        }
    }
}
```

**Code 7.1. Console class with getClientID method testing.**

Just a few lines of code allows test a full functionality of a method without building any specific GUI or applying it to existing. The console itself with the rests is on Figure 7.3:

```

Enter client Name
Matti
Connected
Select correct person and write correct number
0. Name: Matti First
1. Name: Matti Second
2. Name: Matti Third
3. Name: Matti Fourth
2
Client ID is 10018

```

**FIGURE 7.3. The Debug Console testing client search.**

The SQL queries and procedures could be also easily tested in SQL manager tool just by executing them. Both methods are not very formal and professional, but quite usual and help developer to make sure there is no critical mistakes in the code.

Another fast-testing method is IDE debugger. When developer build the solution, compiler check the code on possible mistakes and show warnings. Some syntaxes mistakes could be highlighted even without compiling. When error is found, it is possible to run code line-by-line until it reach the error or any specified point, to see it path and variable values. This is difficult, but very correct method of error correction.

Unit testing in MS Visual Studio could be done in a very convenient way, by installing a small add-on “Unit Test Generator” in Extensions and Updates. After completing a method in a class developer can just right-click on the method and choose “generate unit test”. This will create a new project called the same as the class library plus name “Test”, create a new test class in it and a test framework inside.

As an example I will take a small function in Gap Calculations class called GapSize():

```

public int GapSize(DateTime last, DateTime starting)
{
    int size = 0;
    if (last.Year == starting.Year)
        size = starting.DayOfYear - last.DayOfYear - 1;
    else
        size = 365 - starting.DayOfYear + last.DayOfYear;
    return size;
}

```

**Code 7.2. Gap size calculation code.**

Very simple code, which is returning the actual gap size based on two dates input. All it does is converting dates to the numerical value in the year, checking, if both dates are in the same year and calculating the size.

The random unit test could be written like this:

```
namespace ProjectKLib.Tests
{
    [TestFixture()]
    public class GapCalculationsTests
    {
        GapCalculations gap = new GapCalculations();
        [Test()]
        public void GapSizeTest()
        {
            Assert.AreEqual(gap.GapSize(DateTime.Today,
            DateTime.Today.AddDays(5)), 4);
        }
    }
}
```

**Code 7.3. Unit test of previous method.**

This test is not systematic and takes any current day as first border date, add 5 more days to it, use it as the second border date and execute code with assuming, that the result should be four days in gap. Way that is more systematic is to analyze and create a boundary table to specify data, which should be used as input.

**TABLE 7.1. Boundary analysis for unit test.**

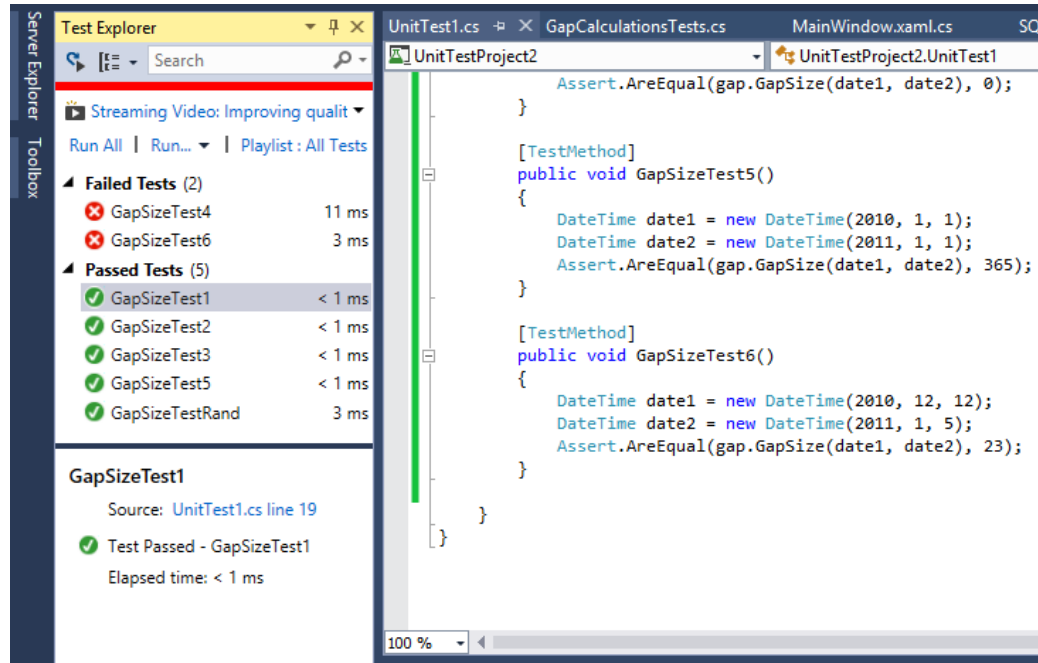
Logical path 1	Logical path 2	Boundary of first path	Boundary of second path
Date 1 and date 2 years are same.	Date 2 is in next year.	Date 1: 01.01.X to (Date 2 – 1 day) Date 2: (Date 1 + 1 day) to 31.12.X	Date 1: 01.01.X to 31.12.X Date 2: 01.01.X+1 to 31.12.X+1

**TABLE 7.2. Input data for unit test.**

Test #	Date 1	Date 2	Expected result
1	1.01.2010	2.01.2010	0
2	1.01.2010	31.12.2010	363
3	12.05.2010	23.05.2010	10
4	31.12.2010	1.01.2011	0
5	1.01.2010	1.01.2011	365
6	12.12.2010	5.01.2011	23



The derivative data is used in unit testing. Here I took two input variants for each border and one randomly chosen input for each group. Then results are formed in unit tests, the environment could be seen on Figure 7.4:



**TABLE 7.4. Testing environment, unit test examples and results.**

In results we can see, that last tests were unsuccessful. That was a mistake of algorithm. Also, this testing is not uncovering the mistake of 366-day years, but on paper it gives 100% code and path coverage.

The last and final testing is user testing and feedbacks. Big projects usually can afford a testing group that get the early versions of the solution and using it, trying to find mistakes in real-life situations. In our case, this is not possible, but the project was given in beta-version to the client. Unfortunately, the time resources did not allow me to gather enough information for full feedback analysis and the solution is still tested in the field. It will take a while, as the real client database need to grow in size till it can be used in full load.

### 7.3 Injection in the workflow

The solution has reached the beta-release state and was presented to the client at the office. The integration process is going slowly, but easy, as currently client using both old and new systems. The full moving to the new system will require the full client database, currently stored in the paper text format, to be translated to the electronic database, which requires huge amounts of time. Also, current database should grow enough to operate current clients, which also need some time.

The solution has many vectors of development and improvement, the closest planned are translation of the solution on the clients convenient language (Finnish - Suomi) and adding some features, that was not released in beta-version.

## 8. CONCLUSION

In a conclusion I want to summarize the work done, results acquired, point out challenges and pitfalls encountered during the work, describe possible ways of future improvements and developments.

The amount of work that was done in limited time period is unusual for me, also the software production from its entire start till the end is not common for the developer team consisting of only one person. It usually includes very different parts and require huge variety of skills and information used. In such limited amounts of time and human resources, as well as lack of previous experience in field, that is not realistic to expect full completion of the plan and various challenges, changes and failures may occur.

However, I can admit, I succeed in reaching the main goals and can describe my results as satisfying my expectation. Even the project is not 100% complete, it was released from development state to early-release. I am also satisfied by the skills and experience I get during project work and expect them to be very useful in my future work life.

Once again, the theoretical aim of the study was spotlighting of possibilities and advantages of creating IT solution for business from beginning and to the release phase by methods of interface-based developing. It also was aimed for formation and explanation of step-by-step guide, detailed sequence of development software solution for a small business. The practical aim was studying of several products, environments, technologies and methods of software development as well as production of ready-to-use program and implementation of it in current workflow for achieving improvements.

I researched the work environment and possible strategies of solving the specified problems and have chosen one, most suitable for current project in my opinion, then follow it during the work. I successfully studied IT project analytical phase, learned how to perform main analysis and preparations for the development, acquired needed skills in data collecting and processing, which could be proven by models I have done and logical results I have got. I also acquired valuable experience of usage of aggregated information and experience of real life work compared to the theoretical investigations. I have done few corrections of plans and prioritization according the limitations of environment and resources, pointed out the most essential parts of the project.

During development stage I successfully designed and implemented all three parts of the project: interface, database and code. This required me to study and use new tools and getting valuable experience in the field. I have described essential relationships between parts and pointed possible ways of solving some issues. After development I

researched and performed error check and testing based on chosen testing strategy and methods and implemented the solution.

The biggest challenge for me was and keep being the limitation of resources. The typical software development project, even for small-business solution, require more than one person in a team and time period longer, than I had. In resource limitations I was forced to correct project requirements and implement less ideas and features, which I wanted or planned from the start. For example, the analytical features was planned but implemented only as starters, giving background for future development. It was also not possible to inject the solution in the working environment properly, with full integration of database into the existing database server and creating connections between existing software solutions. Some pitfalls were not expected or planned at all, like the problem of clients with identical names and required immediate reactions in the project.

Because of limited amount of features was released from the list of planned features, the project has long plan of developed and will be developed in future. I am planning to keep communicating with client and implement more features further as well as providing help and support for any possible issues. As it was mentioned, the user-testing phase is not completed and there is still some chance of minor errors occurs, that should be immediately fixed. I also expecting various feedbacks from users, to understand best path of improvements. However, the closest improvement that is planned will be translation of the solution in the other language.

In the end of the work, I want to admit, that this work has a high influence in my future professional career and possible employments, as I examined one of my possible fields of work. I am willing to make use of my acquired skills in future and going to develop them further.

## APPENDIX

### 9.1 References:

Babenko, Viktor 2016. Business-oriented informational systems development strategies. Syktyvkar.

IBM Global Business Services 2013. The software edge, Executive Report. GBE03545-USEN-01. WWW document.

[https://www.ibm.com/smarterplanet/global/files/se\\_sv\\_se\\_products\\_the\\_software\\_edge\\_.pdf](https://www.ibm.com/smarterplanet/global/files/se_sv_se_products_the_software_edge_.pdf)

Ben-Ari, M. 2006. Understanding Programming Languages. Weizmann Institute of Science.

<http://pages.towson.edu/aconover/Documents/Programming%20and%20Algorithms/Understanding%20Programming%20Languages.pdf>

Kim, Larry 2002. XML Integrated Development Environments. Altova, Inc. & Altova GmbH. PDF document. [http://www.altova.com/whitepapers/xml\\_ide.pdf](http://www.altova.com/whitepapers/xml_ide.pdf)

Löwgren, Jonas 1995. Applying Design Methodology to Software Development. Department of Computer and Information Science Linköping University.

Sommerville, I. 1989. Software engineering. Wokingham. England: Addison Wesley, Third edition.

Czepiel, John; Kerin, Roger 2009. Competitor Analysis. PDF document.

<http://pages.stern.nyu.edu/~jczepiel/Publications/CompetitorAnalysis.pdf>

ATC 2007. Requirements Identification and Analysis. PDF document.

<http://www.gredia.eu/upload/GREDIA-D1.2%20Requirements%20Identification%20and%20Analysis.pdf>

Amal, Ali 2005. The Qualitative Report Volume 10 Number 4. Brock University, Canada. PDF document.

<http://www.nova.edu/ssss/QR/QR10-4/ali.pdf>

Skulmoski, Gregory; Hartman, Francis; Krahn, Jennifer 2007. The Delphi Method for Graduate Research, Journal of Information Technology Education Volume 6. PDF document. <http://jite.org/documents/Vol6/JITEv6p001-021Skulmoski212.pdf>

Air University. Basic Tools for Process Improvement, Module 2 BRAINSTORMING PDF document. [http://www.au.af.mil/au/awc/awcgate/navy/bpi\\_manual/mod2-brainstm.pdf](http://www.au.af.mil/au/awc/awcgate/navy/bpi_manual/mod2-brainstm.pdf)

Dickover, M. E., McGowan, C. L. & Ross, D. T. 1977. "Software Design using: SADT" Proceedings of the 1977 annual conference. ACM.

Ross, D.; SCHOMAN, E. 1977. "Structured Analysis for Requirements Definition". IEEE Transaction on Software Engineering

Fahim, Ahmed; Robinson, Stewart; Tako, Antuela 2014. Using The Structred Analysis And Design Technique (Sadt) In Simulation Conceptual Modeling.

Mylopoulos, John 2004. Structured Analysis and Design Technique (SADT). PDF document. <http://www.cs.toronto.edu/~jm/2507S/Notes04/SADT.pdf>

Donald S. Le Vie 2010. Understanding Data Flow Diagrams. Information Development Director Integrated Concepts, Inc.. PDF document. [http://ratandon.mysite.syr.edu/cis453/notes/DFD\\_over\\_Flowcharts.pdf](http://ratandon.mysite.syr.edu/cis453/notes/DFD_over_Flowcharts.pdf)

Object Management Group (OMG) 2007. Business Process Modelling Notation (BPMN)1.2. PDF document. <http://doc.omg.org/formal/09-01-03.pdf>

Owen, M. 2005. BPMN and business process management, an introduction to a key process modelling tool. WWW document. [http://researchlibrary.theserverside.net/detail/RES/1215708509\\_251.html](http://researchlibrary.theserverside.net/detail/RES/1215708509_251.html)

Pintus, Antonio; Paternò, Fabio; Santoro, Carmen 2010. Modelling user interactions in web service-based Business processes. WEBIST - 6th International Conference on Web Information Systems and Technologies. PDF document.

<http://giove.isti.cnr.it/attachments/publications/WEBIST.pdf>

Sparx Systems 2004. The Use Case Model. Sparx Systems UML Tutorials. PDF document.

[http://www.sparxsystems.com/downloads/whitepapers/The\\_Use\\_Case\\_Model.pdf](http://www.sparxsystems.com/downloads/whitepapers/The_Use_Case_Model.pdf)

Agile Academy 2016. Agile Practices Help Sheet. PDF document.

<https://www.agileacademy.com.au/agile/sites/default/files/aboutpdfs/AgilePracticesMoSCoWAgile%20Academy.pdf>

Qiao, Ma 2009. The Effectiveness of Requirements Prioritization Techniques for a Medium to Large Number of Requirements: A Systematic Literature Review. School of Computing and Mathematical Sciences. PDF document.

<http://aut.researchgateway.ac.nz/bitstream/handle/10292/833/MaQ.pdf?sequence=3>

NEAUG 2013. Wireframes, Prototypes, and Comps. PDF document.

<http://neaug.org/files/Wireframes-prototypes-comp.pdf>

Quesenbery, Whitney; Bachmann, Karen 2004. Prototyping and Usability Testing with Visio. WQusability. PDF document. <http://www.wqusability.com/handouts/visio-prototyping.pdf>

Troelsen, Andrew. 2012. Expression Blend with C# examples.

Kosinska, Elena and MediaCarbon, Inc. 2011. Microsoft Expression Blend 4 Step by Step.

Robbins, Robert 1995. Database Fundamentals. Johns Hopkins University. PDF document. <http://www.esp.org/db-fund.pdf>

Simovici, Dan 2011. Introduction to Database Concepts, PDF document.

<http://www.cs.umb.edu/cs630/hd1.pdf>

Lakshmi, Priya; Razia, Sultana 2013. OLAP (Online Analytical Processing). PDF document. <http://www.srmuniv.ac.in/sites/default/files/files/OLAP.pdf>

Indstuds.com 2015. WHAT IS OLTP. PDF document.

<http://indstuds.yolasite.com/resources/OLTP.pdf>

Halvorsen, Hans-Petter 2016. Structured Query Language. University College of Southeast Norway. PDF document.

<http://home.hit.no/~hansha/documents/database/documents/Structured%20Query%20Language.pdf>

Nakov, Svetlin 2013. Fundamentals of computer programming with c#. Sofia.

MacDonald, Matthew 2012. Pro WPF 4.5 in C#, Apress.

Williams, Laurie 2006. White box testing. (A) PDF document.

<http://agile.csc.ncsu.edu/SEMaterials/WhiteBox.pdf>

Williams, Laurie 2006. Black box testing. (B) PDF document.

<http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf>

Hendrickson, Elisabeth 2008. Driving Development with Tests: ATDD and TDD. Quality Tree Software, Inc. PDF document.

<http://testobsessed.com/wp-content/uploads/2011/04/atddexample.pdf>

Parkin, Rodney 1997. Software Unit Testing. IV&V Australia. PDF document.

<http://condor.depaul.edu/sjost/hci430/documents/testing/UnitTesting.pdf>

Blake, Neate 2007. Boundary Value Analysis. University of Wales Swansea. PDF document.



<http://www.cs.swan.ac.uk/~csmarkus/CS339/dissertations/NeateB.pdf>

Cottonwood software. Campground Master. (A) Official webpage.

<http://www.campground-master.com/index.html>

Cottonwood software. Campground Master. (B) Short product brochure. PDF document. [http://www.campground-master.com/zip/brochure\\_2\\_pages.pdf](http://www.campground-master.com/zip/brochure_2_pages.pdf)

Spirit Works Software Inc.. Space Rental Tracker. Official web page.

<http://productivity-software.com>

Global E-SoftSys Pvt.Ltd. Booth tracker. (A) Official web page.

<http://www.boothtracker.co.uk>

Global E-SoftSys Pvt.Ltd. Booth tracker. (B) Short product brochure. PDF document.

<http://www.boothtracker.co.uk/downloads/BoothTracker.pdf>

## 9.2 Additional documents

### Questionnaire:

#### Questionnaire

#### Questions for specification (determining functionality) of program K +

How the program should be run?

- Single-User (one local installation, one same operator).
- Controlled single user (a local installation, but at different time, different operators can work with different access).
- Group single-user (several local installations, synchronization is done in chosen time periods).
- Multi-user, networked (one server with multiple simultaneous clients).

Is there a need to keep records of information about customers (who rent tables)?

- Yes
- No

If yes, choose data needed to be kept:

- Customer name
- Customer address
- Customer tel.
- Customer e-mail
- Other Customer contact details
- Please specify: \_\_\_\_\_
- Customer rent time
- Table number
- Other \_\_\_\_\_
- Customer rent history
- If yes, choose:
  - Table numbers
  - Dates
  - Time periods
  - Final saldo
  - Units sold
- What features do you expect to see in new program? (also specify value of features. 1 - highest)
- Table map
- Table calendar
- Table history
- Client data storing tools
- Statistics and analyzing tools

Automatization of processes (one click auto booking)

Value the specified functional and operational features of the program in order of importance (1 - the most important):

Intuitive work with the program

Ability to perform multiple tasks with program

Protect information from unauthorized access

Ability to analyze the history of sales with various methods

Personalization of work (bind any operations with the goods to the store manager, who produces them)

Possibility to expand program or add new features

Compatibility with different workstations

**Questions to clarify the characteristics of the analyzed business process**

How much time client has to decide the table rent period, table number and start renting?

\_\_\_\_\_

In what form the contract between the store and the client is done? (the template of the contract)

Do you use any program for accounting? Which?

\_\_\_\_\_

Is there a form of an information report on sales for the period to the customer-seller? (attach form)

Are some of the client information stored in other organization programs?

Yes

No

If yes, do they need to be stored again in K+?

Yes

No