Pedro Santos Rodrigues

Bachelor in Computer Science

Accelerating SQL with Complex Visual Querying

Dissertation plan submitted in partial fulfillment
of the requirements for the degree of

Master of Science in
Computer Science and Informatics Engineering

Adviser: Teresa Romão, Assistant Professor,
NOVA University of Lisbon

Co-advisers: Rui Nóbrega, Assistant Professor,
NOVA University of Lisbon

Tiago Simões, Principal Product Designer,
OutSystems

FACULDADE DE
CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

February, 2020

Abstract

The dissertation must contain two versions of the abstract, one in the same language as the main text, another in a different language. The package assumes that the two languages under consideration are always Portuguese and English.

The package will sort the abstracts in the appropriate order. This means that the first abstract will be in the same language as the main text, followed by the abstract in the other language, and then followed by the main text. For example, if the dissertation is written in Portuguese, first will come the summary in Portuguese and then in English, followed by the main text in Portuguese. If the dissertation is written in English, first will come the summary in English and then in Portuguese, followed by the main text in English.

The abstract shoul not exceed one page and should answer the following questions:

- What's the problem?
- Why is it interesting?
- What's the solution?
- What follows from the solution?

Keywords: Keywords (in English) ...

Resumo

Independentemente da língua em que está escrita a dissertação, é necessário um resumo na língua do texto principal e um resumo noutra língua. Assume-se que as duas línguas em questão serão sempre o Português e o Inglês.

O *template* colocará automaticamente em primeiro lugar o resumo na língua do texto principal e depois o resumo na outra língua. Por exemplo, se a dissertação está escrita em Português, primeiro aparecerá o resumo em Português, depois em Inglês, seguido do texto principal em Português. Se a dissertação está escrita em Inglês, primeiro aparecerá o resumo em Inglês, depois em Português, seguido do texto principal em Inglês.

O resumo não deve exceder uma página e deve responder às seguintes questões:

- Qual é o problema?
- Porque é que ele é interessante?
- Qual é a solução?
- O que resulta (implicações) da solução?

E agora vamos fazer um teste com uma quebra de linha no hífen a ver se a LaTeX duplica o hífen na linha seguinte...

zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz zzz zzzz comentar--lhe zzz zzzz zzz zzzz

Sim! Funciona! :)

Palavras-chave: Palavras-chave (em Português) ...

Contents

Chapter 1

Introduction

This chapter will introduce this thesis, starting with a brief contextualization about the actual technological framework, followed by a description of the motivation behind it. In addition, will be presented an overview of the problem, such as expected contributions and the structure of the document.

1.1 Context

Nowadays, Information and Computer Systems have been dominating data processing. These fields are genrators of positive impact for personal and business areas presenting a prominent place on any information system. However, well before any electronic system, since people started to count or write, they have needed to store pieces of information. [4] Thenceforth, for many years, people have used physical information, like paper, to store data, however, with the digital transformation, these resources are less and less used.

In the decades of the 1960s, Database Management Systems (DBMS) arose, and later at 1970s new management systems that use relational models, designated as Relational Database Management Systems (RDBMS). Moreover, the first Data Query Languages (DQL) appeared, like SQL [7] which was considered by the ANSI [1] and ISO [2] as the standard query language [19]. These technological evolutions have improved the effectiveness and the efficiency of the querying process. However, to find more specific and complex information on databases a higher degree of DQL understanting was required. Thus, if from one side the technological evolution and the digital transformation have optimized the data querying process, only a subset of people could use these powerful querying technologies.

[1] American National Standards Institute
[2] International Organization for Standardization

Visual Query Systems (VQSs), defined by Catarci, *et. al.* [6] as "systems for querying databases that use a visual representation to depict the domain of interest and express related requests", are used to mitigate some problems already referred. These systems use different visual representations and interaction strategies to make database queries, using a more intuitive visual approaches, instead of using textual languages, which are more difficult to learn mainly for people without programming base knowledge. In addition, even if it is not mandatory to be considered a VQS, some systems have also data visualization features which can be useful to view the query result, or even the possibility to manage the database schema in a visual way too.

However, usually, these visual languages are associated as more useful to naive users, while textual languages are associated to more expert users. Conversely, some studies have revealed that visual languages might be convenient to the expert users too. For example, the comparison made by Catarci and Santucci [5] concludes that diagrammatic languages can reduce the error rate of the queries, in comparison with those made in a textual language by expert users. These results imply that even expert users make mistakes in simple queries (e.g. they may not remember the name of the tables or the precise syntax of some language expressions). Thus, it is important to analyse how those languages could be used to optimize the querying process not only to the users with a low experience level but also for highly experienced users.

Nonetheless, the widely users' background and the diversity of the data domain made that a lot of these systems need to be modelled to a specific domain. So, it is very difficult to find an global integrated solution that covers necessities of all users on all domains being this personal or professional.

1.2 Motivation

Low-Code Development is a recent development paradigm which seeks to reduce the time and effort spent in tasks that will not have a significant impact on the final product outcome. Just as the rise of high-level languages, APIs, and third-party infrastructures have provided to developers to be more productive and spend more effort on the most valuable sections of the software they produce. Also, the goal of the Low-Code is to extend this reasoning, using visual IDEs, connectors between components and lifecycle managers, to put away some concerns as infrastructure and re-implementation of patterns and free up people to think better on things that could be more relevant to their objectives [40].

The goal of this project is to analyse and improve the OutSystems Platform [33] data querying component that can be used to create relational database queries through drag and drop interactions and simple configurations beyond visual components and multiple drag and drop features. This platform aim is to provide a complete application development environment where users with different development backgrounds can build,

deploy and manage their applications following good practices and using state-of-the-art technologies even without having to worry about those details.

However, conversely of No-Code approaches, the development through low-code systems give the possibility to use low-level code, written in textual languages like Java, .NET or SQL, in order to increase the extensibility and the power of low-code solutions [42]. Therefore, it is provided to users an alternative to perform their requests that are not supported by the low-code visual approaches. However, if the visual languages of low-code platforms are more robust, responding more thoroughly to users' requirements, there is a diminished demand to resort on these textual programming languages which are high error-prone and have a worse learning curve, requiring also, on multiple situations, previous coding experience.

Furthermore, as mentioned by Amaral *et. al.*, web and mobile applications produced on OutSystems' technology, have proven not only an increase on quality [21], but also allowed developers to increase 10.9x on productivity, when compared with IT Industry standards that do not use these rapid software development solutions [38]. These results reinforces the importance of this matter, the improvement of visual languages used on this platform.

1.3 Problem Description

The OutSystems platform provides a visual query language that allows users to retrieve data from databases by simple processes. Besides, with this language, it is possible to perform some operations that are usually supported by textual Data Querying Languages (DQL), namely join, filter, sort and group operations.

Currently, the OutSystems' solutions have been applied on digital transformation processes in multiple industries that lead with high quantities of data, in order to accelerate the application development processes, unlocking its value and growth.

To this extent, the already implemented querying tool is not able to deal with a set of scenarios, because it might be not accurate when the domain has a lot of tables involved, or does not support essential advanced constructors. Are part of these constructors not supported clauses like IN, NOT IN, EXISTS, NOT EXISTS, DISTINCT, UNION and the possibility to use subqueries.

Under the above mentioned circumstances, the goal of this project is to design and evaluate a new and more powerful Language and User Experience that allows developers to do all these complex data queries in a very easy way without using SQL.

1.4 Research Questions

- What query features are supported by the existing OutSystems visual query language?

- Why do OutSystems developers often use SQL to perform database queries?
- Why are some queries that can be built visually written though SQL instead?
- What are the main causes that users point out to use SQL?
- What users are more unsatisfied with the current provided approach to retrieve data?
- Can we enable OutSystems developers to easily do all kinds of database queries without ever using SQL?

1.5 Main Expected Contributions

This work aims to provide a set of contributions not only as a scientific research, but also with a perspective of creating the most value added to OutSystems. Thus, are presented below a summary of the main expected contributions for this project:

- A synthesization of the design concepts, including relevant interaction and conceptual models, usability definitions, guidelines and principles, and a description of the processes to evaluate the human-computer interaction in the context of a software analysis;
- Provide a state-of-the art about what are the most significant Visual Query Systems, presenting also a comparison of visual representation techniques and interaction strategies used, as well as other important features proper for this study;
- A description of the analysis made to identify what are the most impactful problems regarding the existing visual query language, which includes user interviews and data analysis;
- Design and implementation of a new graphical user interface prototype that tries to improve the existing solution to make queries visually. This prototype expects to fix some predominant problems selected as the most relevant to solve;
- An usability evaluation of the prototype developed through the use of user tests, confronting these results with the first obtained.

1.6 Structure

The remaining chapters of this thesis are organized as follows:

- Chapter 2 - Related Work: presents a short description of the OutSystems Platform, as well as a description of the existing techniques that already exist on the context of the main topic of this thesis - data visualization and visual querying. Besides,

other commercial applications will be enumerated which can have relevant content for this study;

- Chapter 3 - Proposed Solution: describes the proposed solution, starting with a requirement analysis, followed by a more detailed explanation about the problem, and finally with a definition of the development scope to understand what problem will be tackled on detail;
- Chapter 4 - **??**: includes a planning of the inherent total work. Thus, will be presented an overview of the tasks that were done on this dissertation plan, together with the preview of the work which will be the focus of the second phase of the thesis, the elaboration.

Chapter 2

Background

After the presentation of the problem and the motivations behind it, in this chapter will be introduced key concepts of Human-Computer Interaction, as well as, a briefly contextualization of the OutSystems Platform Background that are indispensable for the comprehension and progression of this study.

2.1 Human-Computer Interaction

Since the goal of this project is to improve and extend the visual language that gives to the user the possibility to construct queries through manipulation of visual components of graphical user interface. Human-computer interaction concepts, rules and techniques are fundamental to design and evaluate any possible solution to this problem. Thus, following, will be presented a summary of human-computer interaction topics taken into account throughout the design and development of this project.

Although computer systems have been designed by humans, these two parts of human-computer interaction do not communicate through the same language. Nonetheless, these type of systems were created to support, in a transparent way, human's tasks and requirements, forgiving careless mistakes. [18] Thus, human-computer interaction fields aim to study the relationship of users and computer systems, in the context of the users' desired tasks, in order to "unfold and reveal challenges and insights, and to instrument appropriate solutions for alleviating the current obstacles to the access and use of advanced information technologies" [43].

2.1.1 Interaction Models

Therefore, have been proposed interaction models to represent the intention of a user to perform a certain task on a system. One of the most influential is Norman's model,

that characterize the interaction of a user with a system beyond execution-evaluation cycles. [18] Thereby, the execution is a flow when the user interacts with the system, and the evaluation phase comprise the presentation and the interpretation of the system output. However, this model only represents the interaction from the users' point of view. Thereupon, Abowd and Beale [1] proposed an extension of this model where it is possible to view how the system communicates through the interface.

In this approach, represented in Figure 21, after the user has defined his goal and respective tasks to achieve it, he transmits his intention (articulation) through input language, and after this, that information is translated to system language (performance), ending the execution phase. Secondly, the result of the action executed by the system is presented (presentation) and the user observes it (observation), ending the evaluation phase.

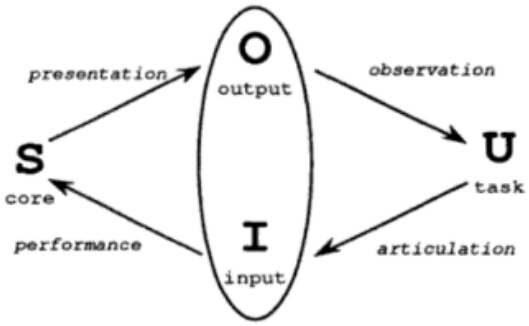


Figure 21: Interaction Model proposed by Abowd and Beale [43] (Based on Norman's Model)

These models are useful to understand two concepts that cannot be forgotten when a system is being designed, since these are two effects that designers want to reduce as much as possible, in order to optimize the effectiveness of the human-computer dialogue. Following, these concepts will be described respecting *Dix et al.* [18] definition:

- **Gulf of execution**: "Difference between the user's formulation of the actions to reach the goal and the actions allowed by the system. If the actions allowed by the system correspond to those intended by the user, the interaction will be effective."

- **Gulfs of evaluation**: "Distance between the physical representation of the system state and the expectation of the user. If the user can readily evaluate the presentation in terms of his goal, the gulf of evaluation is small."

2.1.2 Main Concepts

The **Usability** of a system is one of the most important concepts in human-computer interaction, that can not be forgotten on the design process, since its attributes must be taken into account performing also a guidance function through all this process. This

concept was standardized in ISO-9241 [22] as "extent to which a system, product or service can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use".

However, usability is not a single-dimensional property, being always associated with its attributes, that characterize the user accessibility when is using the system into five different points, such as referred by Nielsen [30]:

- **Learnability**: How easy is the learning process until a novice user (has not used the system before) has some high-level of proficiency using the system [49]. The learnability is higher as the learning process is faster, and the user has to spend less effort to reach his goal. Also, it depends on tutorials and training provided to users. A system that requires less training has a higher learnability than a system that requires more training. The time that a novice user requires to perform some specific tasks can be used to measure learnability. Learnability can be improved using tips while a novice user explores the system doing his first tasks.

- **Efficiency**: Refers to the productivity level of a user which have already learnt how to use the system. Efficiency can be measured analysing the time that expert users spent to do specific tasks on the system. This attribute can be improved, for example, adding shortcuts to accelerate the interaction process.

- **Memorability**: Defines how easy is for a user, that was using a system before but did not use it for a time period, to do his desired tasks on the system. So it's related to how many time the user has not used the system and the time that the user needs to remember how the system works. Therefore, if a system has a good memorability the user does not need many time to remember it, even if it has stopped use it for a long period of time. Memorability can be measured, for example, analysing the interaction process of a user who has been away from the system, while he uses the system again. The use of visual components and metaphors with real-life objects helps, sometimes, the users in this process.

- **Errors**: A system not only must have a low error-rate but only if an error occurs, the user should be able to recover from that. Since there are multiple types of error with different severity levels, it is important that catastrophic errors must not occur. This attribute can be measured evaluating the error-rate, taking into account the severity levels of the errors. Furthermore, if a system has errors, that can be reverted and do not have a negative impact on the final result, cannot be forgotten that these errors also harm the efficiency of the system.

- **Satisfaction**: The most subjective attribute of the usability that is related to the overall satisfaction of the user when uses the system. Could be measured by asking the users about the experience while they are using the system, always searching for subjective answers.

Nonetheless, as mentioned by Nielsen [30]:

"it is not always possible to achieve optimal scores for all usability attributes simultaneously".

Thus, when a system is designed it is necessary to prioritize what are the most important attributes for the users and the domain where the system will be used and applied. These trade-offs are one of the most challenging tasks of the design processes because it depends on user expectations and their backgrounds, as well as, the problem domain and what are the main focus of the system use.

Accordingly, it is fundamental that the design process can focus on target users of the systems. Therefore, following will be described some main concepts, processes and techniques for a design process centered on the users.

2.1.3 User-centered Design

Before user-centered design principles and methodologies were adopted, the Waterfall model was commonly adopted as a software development process. This model comprises five sequential phases, from the requirements specification phase to the operation and maintenance phase, and has a good quality control since documentation and planning are a major concern of this methodology.[2] However, the stages of this model are not overlapping stages, so other development methodologies and philosophies arose to mitigate this problem and include the user on the design process, due to their impact on the usability of the system.

Consequently, it was necessary a new model that has the users included on the development process to verify, along the development, if the approaches adopted are positive and what is the users' acceptability. Nielsen [29] reinforces this saying that "user interfaces should be designed iteratively in almost all cases because it is virtually impossible to design a user interface that has no usability problems from the start. Even the best usability experts cannot design perfect user interfaces in a single attempt, so a usability engineering lifecycle should be built around the concept of iteration".

The Spiral model of iterative design arose as an iteration through design, implementation and evaluation phases where the cost and accuracy increase on each iteration, since first iterations should use low-cost resources, like paper prototypes, and when the results are positive the accuracy is incremented, changing to computer prototypes, for example. [23] An overview of this model is represented in Figure 22.

2.1.3.1 User and Task Analysis

Regarding the concept of usability presented above and the importance of the users on the design process, it is important to define the users and their desired tasks of the system in order to find the best solution as possible to the usability attributes trade-offs. Just a

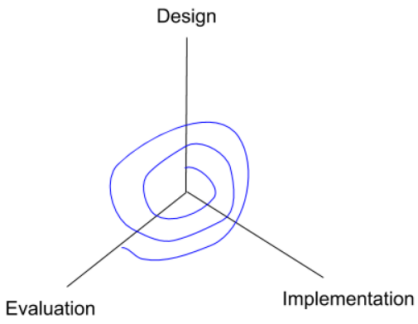


Figure 22: Sprial model of Iterative Design (adapted from [23])

good description of the users and the tasks of the system leads the designers to the best choice of what are the usability attributes most important for the system.

Accordingly, it is important to make a **User Analysis** to understand all users' characteristics that could have an impact on the acceptability of the system. The expected result of this analysis should be a set of structured information that characterize the users of the system in terms of technological expertise, knowledge of business domain, application experience, educational background, gender and age, as other aspects that might be useful to comprehend, depending on the system's users and domain. [17] The more traditional process to gather this information is through questionnaires or interviews, but that can also be obtained by conducting a market analyses or observational studies. [30]

Furthermore, it is essential to enumerate and analyse the tasks the users should perform using the system. The **Task Analysis** process aims to aggregate information about the tasks that should be performed on the system, starting from the system overall goals and break down these to obtain individual tasks. [30] Moreover, the goal of this analysing process is to obtain more structured information about: how the tasks are performed using the existing systems, what are the pre-conditions and the requirements of each task, why the users need to perform this tasks, and others that might be useful to characterize the tasks of the system.

The techniques used, to extract information to this analysis, tries that users specify how the tasks should be done instead of how they would perform them, resorting on examples as well as possible, in order to understand what type of strategies are used, what type of exceptions from their normal workflow are occurring, and other aspects that can be observed where the communication with users is on a concrete level. [30] In addition, Nielsen [30] points out that "The users' model of the task should also be identified, since it can be used as a source for metaphors for the user interface", which reinforces that these dialogues with users to obtain analysis content, can be useful also to find relevant solutions to latter design process phases.

Therefore, the outcome of this analysis should contain a list of the entire tasks that users what to perform in the system, the information that is required to complete them, the steps and the dependencies between tasks, all the outputs that must be generated,

and how is the communication process between the users associated with the system's tasks. [30]

2.1.3.2 Sketching and Prototyping

After the user and task analysis process, designers must start sketching and prototyping ideas and approaches, in order to think about how can they solve the problems. Nevertheless, this phase of the design process should start with sketching techniques, as these are not only a good and inexpensive starting point to communicate ideas, but also help to develop structure and enrich the reasoning, leading to the perception of other details as well as of other approaches to solve the related problems.

While sketching techniques are more plentiful, and have a low detail level, being mainly based on suggesting and exploring, rather than retrieving results, the prototyping phases, have more refinement approaches and are used to test the design choices made.

However, there are prototypes with different thoroughness degrees, presenting different advantages and disadvantages. Thus, it is important to start the prototyping process with low fidelity prototypes, as paper prototypes, since the objective of these is to evaluate the conceptual model (if the users understand the system), the functionalities presented, the navigation, the screen components distribution, and the terminology used. After the evaluation of these prototypes presents good results, high fidelity prototypes should be used, like computer prototypes. There are a set of available tools to assist in the building process of these prototypes, such as Balsamiq [3] and Mockingbird [28]. These different prototype types can detect different issues when tested with users, so it is very important to test prototypes with different granularity levels.

Furthermore, there is another relevant aspect of the prototype designing, that is the scope definition of the prototype. It is important to define what features the prototype undertakes and what is the inherent detail level. Nielsen [30] describe this as two dimensions of prototyping: horizontal prototyping and vertical prototyping, as demonstrated in Figure 23. A vertical prototype is characterized as a prototype to test a restricted part of the system but with real users and circumstances. A horizontal prototype is presented as suitable for test the entire system but in a less realistic approach.

Finally, regarding the methodology about how to use the prototypes built, Dix et al. [18] refer that are three main approaches:

- **Throw-away**: After the prototype is built and tested, it is used on the final system development, but after that, the prototype is discarded and the rest of the design process continues without relaying on the prototype previously built;

- **Incremental**: First, the system is separated into different parts, and each part is built, one at a time. So the prototypes are developed separately, regarding its correspondent part, and finally are combined together to build the final system;

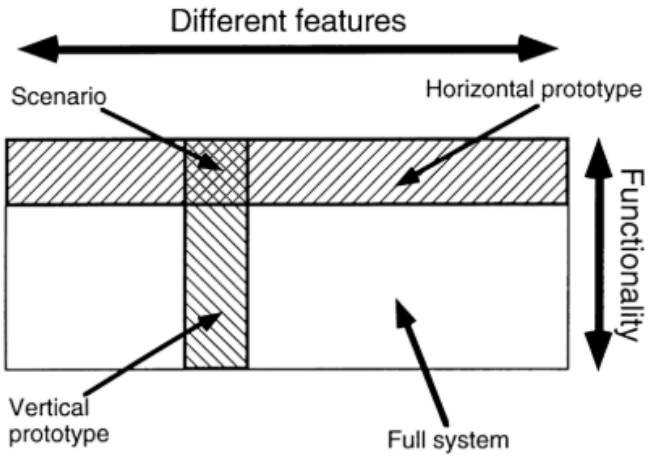


Figure 23: The two dimensions of prototyping: Horizontal prototyping keeps the features but eliminates depth of functionality, and vertical prototyping gives full functionality for a few features (extracted from [30])

- **Evolutionary**: Contrary to the throw-away approach, the prototypes developed are used as the basis for the next iteration of design.

2.1.3.3 Evaluation Techniques

The evaluation phases are crucial in the design process, since they allow designers to understand the systems' specific problems and the impact of the interfaces on users. So, the expected outcome of these processes is a list of usability issues ordered by priority level, referring what usability principles and guidelines are not being accomplished and what solutions can be applied to solve the problem. [30]

However, there are different approaches to evaluate interfaces. First, one important topic that distinguishes two types of techniques is who performs the evaluation. There are techniques where only the designers and specialists are involved on the process, and are another type of evaluation where users participate in the evaluation. [18] Thus, there are two types of evaluation: evaluation through expert analysis and evaluation through user participation.

Evaluation through expert analysis

In this type of evaluation, designers, or other specialists, evaluate the system, supporting their analysis on cognitive principles, to preview usability problems likely to occur. Moreover, this evaluation not only is cheaper because it does not involve users, as also can be applied on any phase of the design process, from the design specification to the high fidelity prototypes. [18]

Regarding the approach presented above, these two methods are one of the most used:

- **Heuristic Evaluation** [31]: The system is thoroughly analyzed in order to find problems that do not follow important and recognized usability heuristics. After a

problem has been detected, it should be reported, not only with a description of the problem but also the indication of the heuristics that have not been accomplished, the severity level of the problem and possible solutions to solve it. [30]

- **Cognitive Walkthrough** [39]: On this method is used a sequence of actions as the principal resource to guide the evaluation process. For each action, the evaluator tries to understand if all steps are clear and visible, as well as, if the system gives a clear feedback, confirming if the action has been completed. Usually, the main focus of this method is to analyse the learnability of the system. Mainly, to understand if the system provides a good learn mechanism through exploration, rather than using manuals, training or other types of *a priori* learning processes. [18]

A study made by Desurvire *et al.* [13] concluded that Heuristic Evaluation made by specialists is better to predict some problems before the user testing process than Cognitive Walkthrough. The reason pointed out for this result is that heuristic evaluation can often help to remind the designers of problems since this method analyses more dimensions of the system than Cognitive Walkthrough.

Evaluation through user participation

Although there exist methods that do not need the users to evaluate the system usability, it is difficult to predict all the behaviours of the users when they interact with a system. Therefore, there are multiple methods to make usability tests with people that are the system's target. Then, following will be presented some methods of user testing which can be resourceful on the context of this work, respecting the terminology used by Dix *et al.* [18]:

- **Observational Methods**: the main principle of these methods is observing users using the system to conclude important usability information about it. Usually, it is requested to the user to 'thinks aloud' since it might be possible to obtain more insights that can be useful, not only to understand why the user might have made an error, but also it can be a good strategy to find starting points for other possible solutions. Moreover, although usually these tests have a set of predetermined tasks, since it is easier to find the user reaction to the system part the need to be tested, also can be executed tests only by evaluating the normal tasks of the users on their work;

- **Experimental Methods**: starting from a properly defined test hypothesis, it is selected a set of users to perform an experimental test to verify if the test hypothesis proves to be true or not. Thus, it is necessary to define previously all the experimental environment, which includes: the test hypothesis that wants to be verified, the users that will perform the test, and the independent and dependent variables. The dependent variables are the ones that express the result of the test in function of the independent variables, such as the task execution time or the number of the

errors that occurred. The independent variables are chosen by the test designer to produce different conditions for comparison. Examples of independent variables can be the size of an interface component or the use of an interaction technique. This method is very useful to verify through a test hypothesis which of the possible solutions presented (independent variables) have a better performance for the intended application context;

- **Query Methods**: these methods focus on what the user thinks about the system, collecting information from interviews or questionnaires to analyse this opinion. One advantage of this method is that it might reveal issues not observed previously complained before, but contrary a lot of cases are not tested, since in the interaction field, many times, the user only finds a problem when it occurs for the first time. So it is not possible to extract concrete information from users that never have passed for this situation.

Finally, independently of the evaluation process adopted (through expert analysis or through user participation), severity ratings should be attributed to the problems identified in order to define the main priorities and understand which might have a larger impact on the system acceptability. However, although these levels should be attributed by specialists, if they use not only cognitive principles, but also use the results observed from user testing phase to sustain the classification of the problems detected, the result can be more accurate.

Besides, the Figure 24, extracted from [30], displays two influential factors that should be taken into account to attribute a severity level to a problem: how many users are experiencing this problem and what is the impact level on the user.

		Proportion of users experiencing the problem	
		Few	*Many*
Impact of problem on the users who experience it	*Small*	Low severity	Medium severity
	Large	Medium severity	High severity

Figure 24: Severity Levels of the Problems based on their impact on the users (extracted from [30])

2.1.3.4 Errors Classification

The errors occurred when a users interacts with a system are an excellent indicator to designers, because understanding the reason that led to error situations is a good strategy

to classify them. Therefore, errors can be classified as slips and mistakes, as will be presented below according to Dix *et al.* [18]:

- **Slips**: in these type of errors, the user knows how to do the intended task on the system, however, he presses a wrong button or closes one window accidentally. So, he understands the action, but a misaction does not allow that he reaches his goal;
- **Mistakes**: these errors occur when the user does not understand the system, formulating a wrong goal. An example of an Mistake is when the user does not understand the action associated with an icon, performing a not intended action.

Therefore, the strategy to mitigate the problems associated with these two types of errors could be different, as mentioned by Dix *et. al.* [18]: "Slips may be corrected by, for instance, better screen design, perhaps putting more space between buttons. However, mistakes need users to have a better understanding of the systems, so will require far more radical redesign or improved training, perhaps a totally different metaphor for use."

2.2 OutSystems Background

The OutSystems Platform has the mission of simplifying and accelerating the development and management of digital enterprise solutions, no matter the dimension and domain of the applications. It covers the entire development lifecycle which aims to promote rapid development and integration, to facilitate and speed up the deployment stages, to keep track the status and health of the applications produced, and to expedite the management of daily operations and configurations on the final products. [32]

2.2.1 Visual Development Environment

Service Studio is the low-code development environment of the platform, which allows the developers to create complete applications using visual elements to perform drag and drop actions. Figure 25 presents an overview of the IDE, which highlights the different areas of the workspace.

Using the widgets and icons provided in the toolbox, the main area is dedicated to design the applications' interface and logic. So, in the main area, there are visual elements, which can be set using the properties editor, placed on the bottom right corner of the screen.

Also, it includes other sections whose main purpose is not the product development, but are related to key actions of the software development process. Therefore, on the window's top, there is a toolbar which has shortcuts to some of the most common operations, and a green circle button, denominated "1-Click Publish button", to start the automated deployment process provided. In addition, the bottom of the window is dedicated not only to the presentation of messages, errors, and warnings but also to debugging the application.

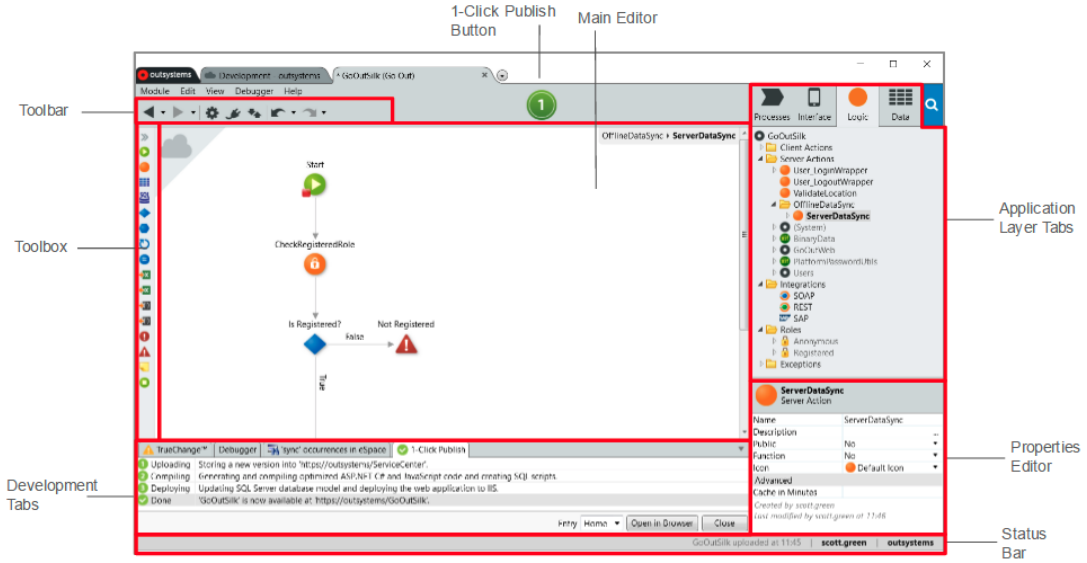


Figure 25: Main areas of Service Studio (extracted from[34])

Since the development environment can be used to develop a complete full-stack application, the elements, which can be manipulated in the Service Studio, can be related to different parts of the application. Thus, following will be described the purpose of each one of the Application Layer Tabs, that contains in that section, a tree view of the elements related:

- **Processes**: in this section, it is possible to create and manage business processes of the systems through a flow that can be composed by human or automatic activities, time waits, conditional decisions and indications to execute processes. In addition, on this section can be configured the timers of application. Then, it is possible to indicate when a timer should start, what is it period and what action should be performed when it is triggered;

- **Interface**: the purpose of this section is the management of the components related to the visual interface of the final application. It is possible to observe all applications' screens, as well as, the variables and the actions related to each one. Moreover, flows between the various screens can be also defined. Accordingly to the overview presented above, if one screen or action is selected, it will be possible to add new visual components to the interface or assign new elements to the action flow in order to define all the client-side logic of the screen;

- **Logic**: in this area is where the core logic of the application is defined. It includes not only the server actions of the application but also the exceptions specification, the existing user's roles and also the integration with external services. Although there is a data section on the application layer tabs, which will be described below, the actions which require data querying are managed into this section.

- **Data**: the database modelling aspects are covered in this section, making possible the creation of diagrams to represent the schema of the database. Moreover, in the tree view of this tab is allowed to establish new entities and static entities in order to define the data model of the application.

2.2.2 Visual Data Querying

The main topic of this work is the improvement of the visual data querying process on the low-code development of applications, using OutSystems. Consequently, the headway of visual querying components of the platform is an essential factor to properly understand the entire project. Then, regarding the last version of the OutSystems platform, will be described, the actual visual data querying tool that is the starting point of this study.

2.2.2.1 Previous Work

Since 2002, which is the release date of the first OutSystems low-code development environment, the Hub Edition 1.0, it has been provided two distinct manners to create database queries [41]:

- **Simple Queries**: visual query builder that allowing the creation of some less complex queries, interacting with a graphical user interface;
- **Advanced Queries**: feature to specify queries textually, using a language based on SQL, but includes some extra syntax to reference variables used of the application development;

Then, since the first versions of the IDE, it is provided two ways to build queries. The first uses the visual language, which does not need so many learning requirements but also diminishes the necessity to remember the entities name or the language syntax. The second provided relies on SQL, which is the standardized textual query language, known for the developers' majority.

The first simple queries versions have accelerated the query building process finding automatically the relationships between the entities chosen. The main idea is that the developer only needs to select the entities intended and the respective join conditions would appear automatically in the query view interface. After this, it will be possible to change the join type, as well as, to add, edit and remove filtering or sorting conditions. Figure 26 presents a simple query example in Hub Edition 2.0 (2003) when the developer was changing the join type to an Outer Join.

This visual querying approach continued in the next versions, adding some minor improvements, such as the support to structures in order to store temporary information without changing the entity definition [36], and the inclusion of a properties pane to view and change the properties of all query elements in a single window [37].

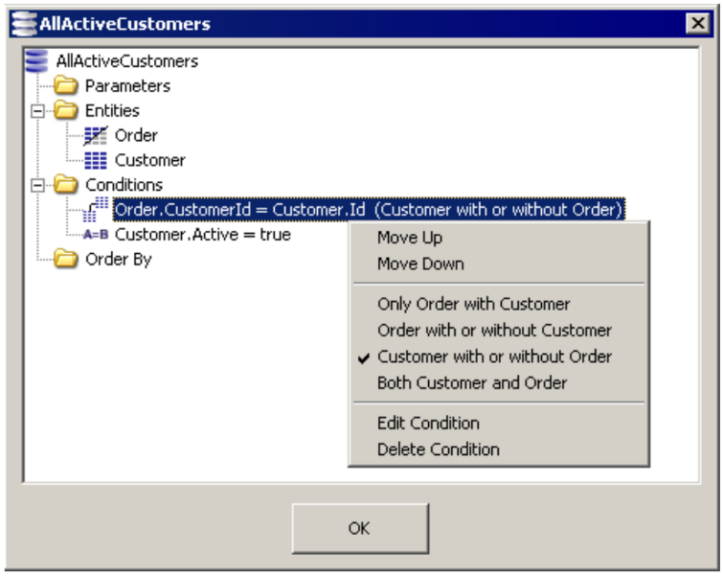


Figure 26: Simple Query example in Hub Edition 2.0 (extracted from [35])

However, at the launch of the OutSystems Platform 9 (2013), has been released an entire new way to manipulate data and express database queries. These new components of the system, called **Aggregates**, have replaced Simple Queries, promoting a new interaction strategy to query databases, where the main focus is the data, instead of query design. Also, new features have been introduced to improve the expressiveness of the visual query language, namely grouping functions and the ability to add calculated columns easier.

2.2.2.2 Current Progress

Since the implementation of Aggregates, the query process without textual languages is more visual and more focused on the query outcome. Aggregates can be used to query data from the server or mobile local storage, and can be created in the following ways:

- If someone is designing the user interface and wants to present some data gathering from the database (server or local storage), he can right-click on the screen where data will be displayed and select “Fetch Data from Database”;
- When an action flow is designed, it is available an Aggregate icon in the toolbox that can be dragged and dropped to the main area;

When the Aggregate is created, the first step is the data source selection in order to specify what entities should be included in the query. Then, the developer should click in the main area and choose the entities he wants or drag and drop the entities to add them to the query. When an entity is added, all its attributes are automatically included in Aggregate. Since an Aggregate can include one or more entities, added on the beginning or later, if it has more than one, it will be analysed to verify if there are relationships between them. Unless there is a relationship that links them, the Aggregate will request

the condition to join them. Otherwise, the entities will be included automatically using an inner join.

Hereupon, the Aggregate has already been created and the data source of it specified, so the visual querying process can be started, in a progressive way, seeing at the same time the query output. Figure X demonstrates an Aggregate on the referred state, to be possible to understand the structure of the interface.

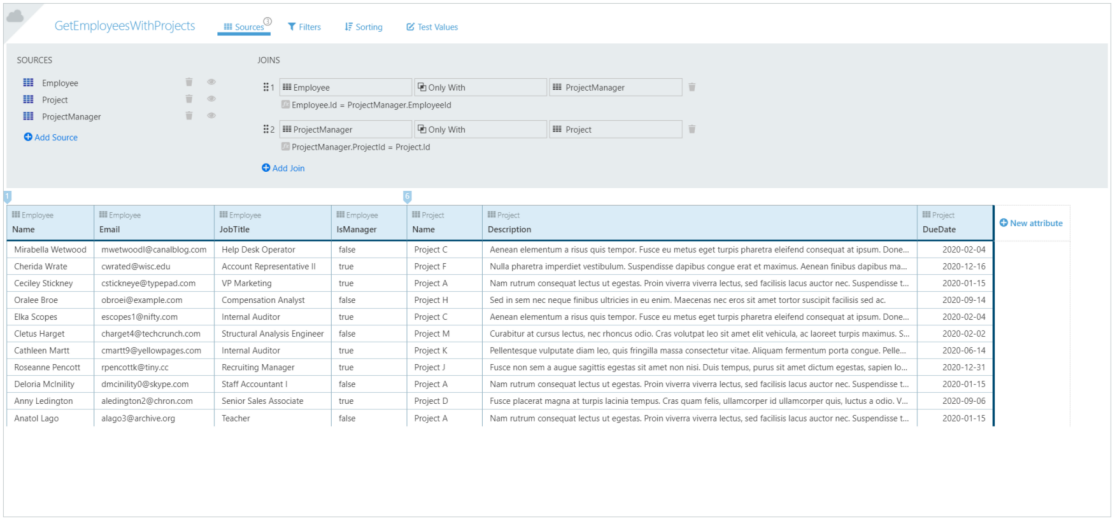


Figure 27: Aggregate Example

First, this component of the OutSystems Platform presents two main capabilities, represented visually in two distinct areas: the query design area, and the viewer of the query result. The former, located at the top of the window, is composed of a set of four tabs, where each selection action changes the grey area to the respective form-based interface. The latter is a table-based interface, which is similar to spreadsheets applications, such as Microsoft Excel [25] of Google Sheets [20] and its principal aim is to provide a direct and visual approach to show the query output.

Focusing on the query design area, the sources tab, illustrated in Figure 28, it can be used to add, change or remove the entities of the quey, defining also, the join types between them. There are three join types available: "only with", "with or without", and "with". The respective joins in SQL are: "inner join", "left join", and "full outer join". Additionally, each one appears with a visual representation similar to Venn Diagrams [24] to be easier to identify which data is selected on each join type. Moreover, it is possible to edit the join condition textually, using the platform expression editor.

The filtering tab is can be used to apply filters in the query likewise the where statements in SQL. These filters can be defined through boolean conditions inserted in the expression editor. Despite the fact that they are provided shortcuts, listed in a tree view and also others in a list of buttons, the conditions are specified using only textual language.

The sort conditions can be added in the sorting tab choosing the add sort or the add

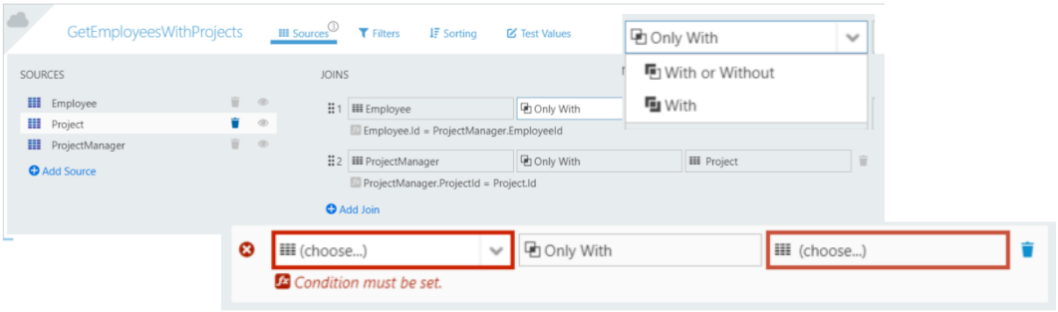


Figure 28: Aggregate - Defining Sources

dynamic sort options available. The difference between them is that dynamic sort relies on a variable of the system, contrary to the other that depends only on an entity attribute. To add a sort, the user has to select what entity wants to sort and specify what are the sort criteria, for example, ascending or descending. Moreover, more than one sort can be inserted for different entities. Also, the sorts inserted can be ordered to define the priority between them.

Finally, the last tab has a different behaviour when compared with the rest of the options available in this interface area. The main goal of this feature is the testing of query output when it is assigned concrete values to the variables referred in the query. Thus, it does not contribute to the query design in the same way as the other tabs, presenting only a test purpose in the context of the query result visualization.

Figure 29 presents an usage example of the last three sections referred, to create a query to filter and sort dates, regarding a value of a variable.

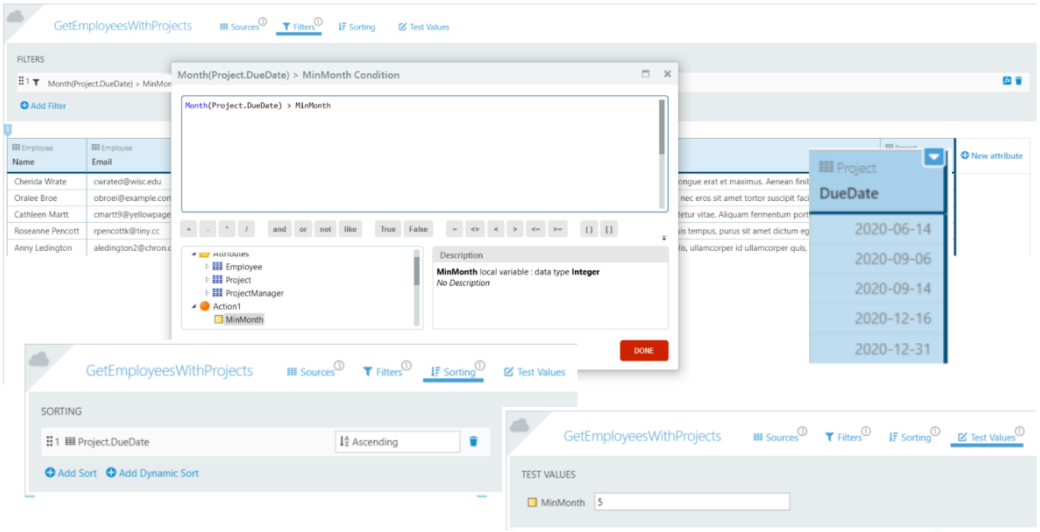


Figure 29: Aggregates - Filtering, Sorting and Test Values in an example of querying DueDates after a month indicated in a variable

On the other side, the area dedicated to viewing the query output provides also provides some functionalities to design the query while the user is interacting and exploring

the query result. Therefore, as represented by an example in Figure 210, the user can change the query when he performs a right-click on a column or when clicks on the new attribute. The only options in the list presented that do not change the query are the hide options since they just change the result in the presentation layer. So, if the user hides a set of columns, he will not see them in the result table, however, the query did not change. In addition, the user can add other attributes based on the existing, so Figure 211 shows an example of this functionality.

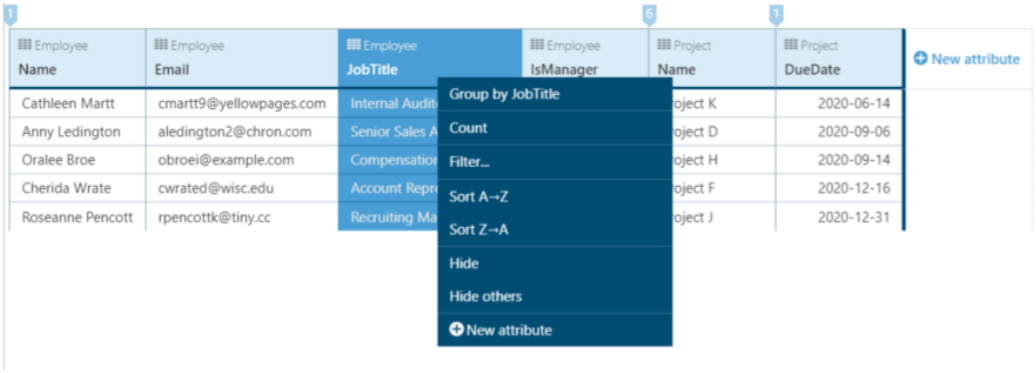


Figure 210: Query Design functions while interacting with Query Result

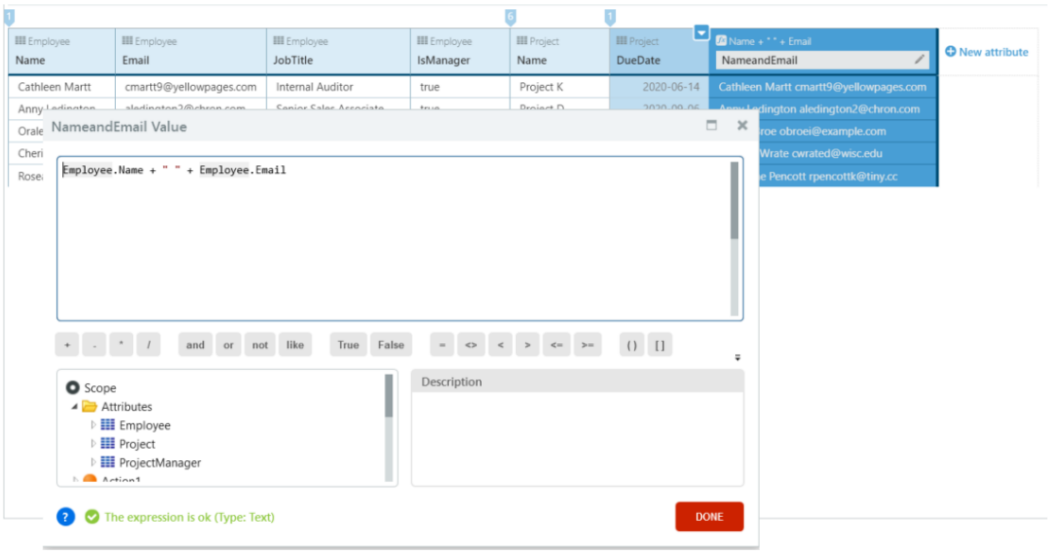


Figure 211: Calculated Attribute Insertion

Chapter 3

Related Work

In this chapter, will be described and compared a set of technologies and techniques, which could be useful to explore solutions, and understand different points of view to manage problems, regarding the project scope. As the project is focused on the improvement of an interface that combines the query design with the query output viewer, some relevant technologies, regarding the former will be compared, and, in a different section, the related work that exists for the latter shall be presented. Moreover, since the users are a fundamental piece of this study, will be presented some information about how is its relationship with data and what are its expectations to these query systems.

3.1 Visual Query Formulation

The first relevant topic to analyse is the visual representation approaches, which can be applied to design an user interface capable to build queries through a visual language. Catarci *et al.* [6] presented an interesting classification according to the visual formalism which the interface is based on:

- **Form-based**: based on forms, which can be seen as a rectangular grid of other components (subforms, groups of cells, a combination of cells, etc.) that group objects in a named collection regarding its structure. Forms and tables are similar, but contrary to the tables, forms allow nesting. Thus, forms can be seen as a generalization of tables. In this approach, the relationships can be represented among cells, cells subsets, or even the overall set, providing to the user three information levels;

- **Diagram-based**: usage of graphical representations, such as graphs, charts and diagrams to better transmit the relationships among data. The aim is to use visual

representations to help the understanding of the relationships between concepts which are represented by textual labels;

- **Icon-based**: as the opposite of the diagram-based, this type of interfaces tries to facilitate the understanding of the concepts instead of relationships. So, are used Icons, which are visual segmented objects to transmit a message or information, using analogies and metaphors with the real-world objects, or even conventions that are used to express no tangible objects, as computer processes;

- **Hybrid**: these approaches can combine the previous visual formalisms in order to select the best combination of advantages to the application usage domain.

Following, it is essential to specify what are the visual query language requisites, to understand and compare the different parts of the systems. Accordingly, Table 31 presents the query creation required specifications, comparing them with the respective indication in SQL.

Table 31: Query Language Requisites

Specification	Description	SQL Indication
Data Source	Entities and attributes which will be presented in the query	Using SELECT and FROM statements
Merge Type	Define how will be merged attributes of different entities	Using JOIN clauses
Filtering Criteria	Criteria that can be used to filter records, presenting in the result only those that fulfil a set of conditions	Using WHERE clause
Sorting Criteria	Define what are the criteria to sort the records of the result	Using ORDER BY or HAVING clauses
Aggregation Functons	Group a set of records by comparison or using mathematical functions	Using GROUP BY statements or SQL functions, such as MIN, MAX, COUNT, AVG and SUM
Calculated Attributes	Attributes added, based on existing ones	Using SELECT statement
Distinct Values	If only different values will be considered in the result (removing duplicated values)	Using SELECT DISTINCT statement
Unions	Combine the result of two different queries	Using UNION operator
Subqueries	Defining a query that uses other queries, for example, to filter the result	Nesting SELECT statements

Nevertheless, there are two relevant aspects, according to the last requisites presented: the interaction process to indicate the query specifications, the interface feedback about what the current query. Both are fundamental since a good visual query language aims to simplify not only, the design process but also, the query readability, promoting a faster and easier recognition of what are the desired data.

Data Source:

Chartio [8] has two components to query databases visually: using the Data Explorer [9] or using the new Visual SQL [11]. Regarding the data source specification, these two systems use different strategies to select and present the entities and attributes related to the query. In the Data Explorer, there is a list of tables in a fixed, scrollable and searchable tree view, where the user can expand each one and choose the desired attributes. Also, above this list of collapsible items, there is a search component that can be used to select the desired attributes. This system divides the attributes into two different types: Measures and Dimensions. Usually, measure refers to quantitative data and dimensions to categorical data. So, to insert the attributes in the query, users can drag and drop the required attributes to the form-based interface that contains the Measures, Dimensions and Filters of the query (Figure 31a) [9].

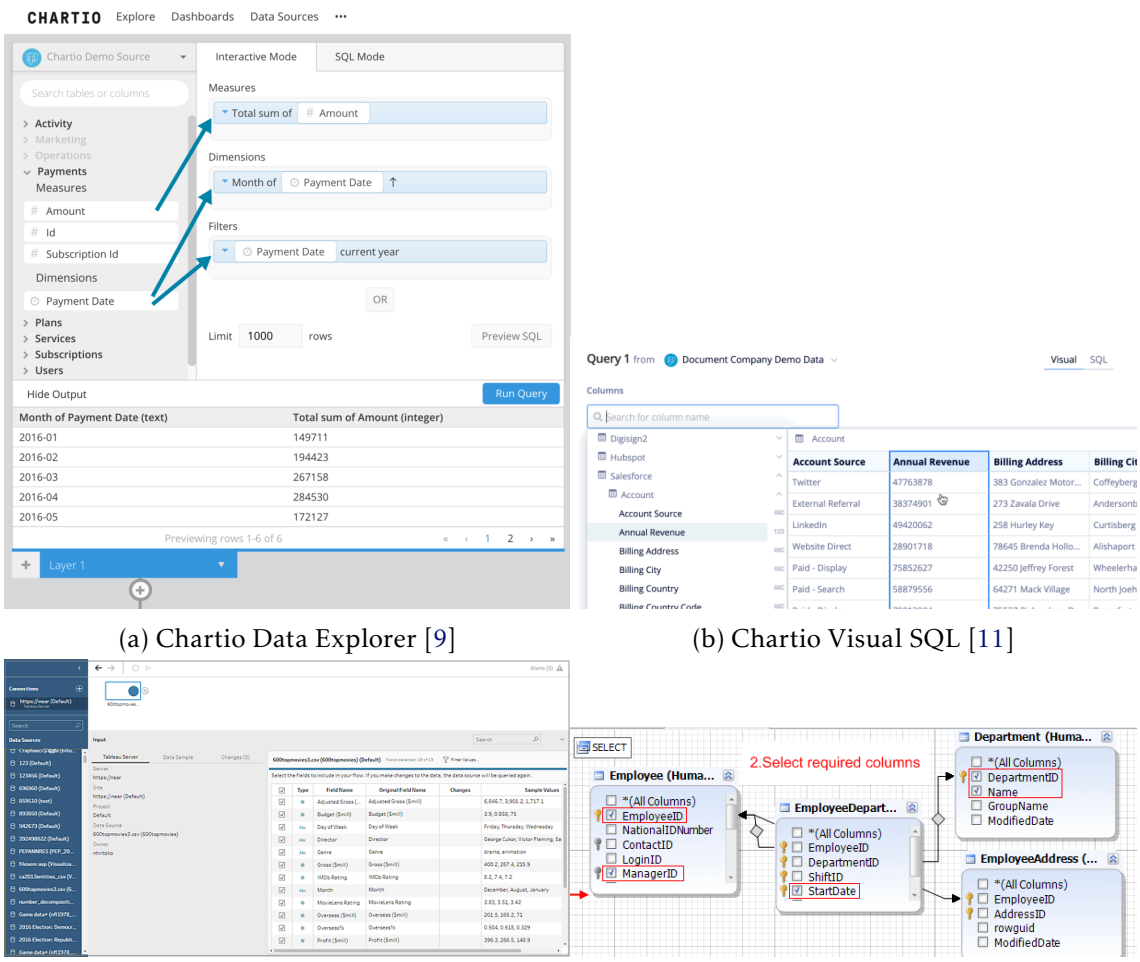
On the other hand, the new component of Chartio, the Visual SQL provides a different interface to select the data sources. Contrary to the previous approach, there is any fixed list in any part of the window to choose the attributes required to the query. In this way, there is only a search text component that is activated when the user clicks on "add column" action. After this, is presented over a new interface component that has a list similar to the referred above where is possible to view a preview of some data entries in the table (Figure 31b) [11].

In the systems referred above the columns are added one by one sequentially, but other systems have different methods to select the table's columns. For example, in Tableau Prep [47] and Microsoft Power BI [26] the table is chosen using a list, and all its attributes are added automatically (Figure 31c). In these approaches, the users can remove, if they want, the columns not desired after. [48] [27]

Other systems, as Devart dbForge Query Builder [16] uses a diagram-based interface to select the entities and attributes of the query. In this system, the user can drag and drop the desired tables to the diagram area, and select through checkboxes the attributes needed, that are presented in the database schema diagram (Figure 31d). Figure 32 presents some examples of the techniques referred above to select the entities and attributes of the query.

Merge Type:

When data of multiple tables are required, it is necessary to establish how to merge them. Then, the visual language needs to adopt an interaction and representation technique to specify it. When data of multiple tables are required, it is necessary to establish



(a) Chartio Data Explorer [9]

(b) Chartio Visual SQL [11]

(c) Tableau Prep [48]

(d) Devart dbForge Query Builder [16]

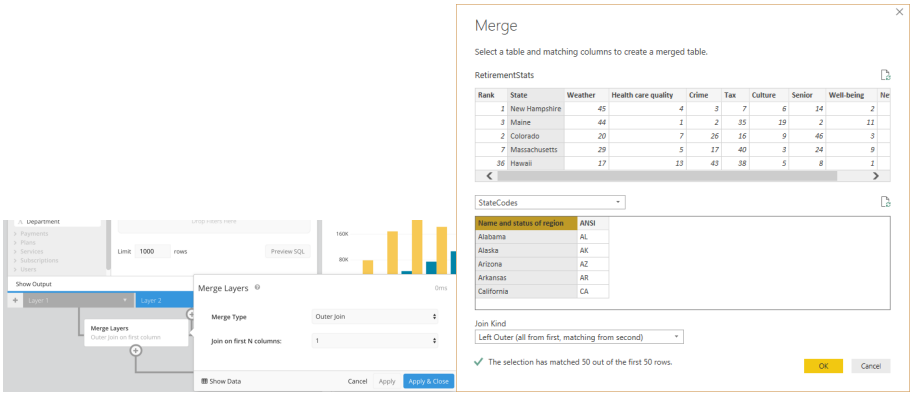
Figure 31: Different approaches to select the entities of the query.

how to merge them. Then, the visual language needs to adopt an interaction and representation technique to specify it. To define a join in Devart dbForge Query Builder [16], the user can only select the attributes' checkboxes of the different tables and the system generates an inner join automatically. To specify the join kind in this system there is a button on the toolbar to select all rows of one table, another, or both. This option can be used to perform left, right and outer joins [15].

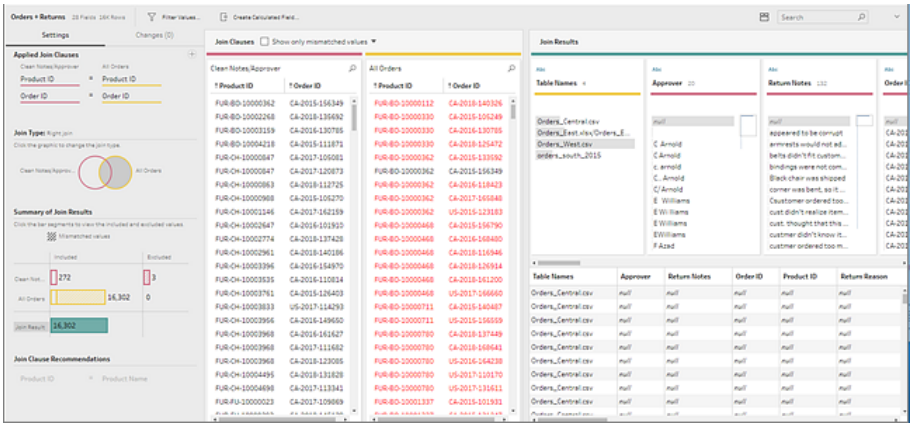
Another approach used by some systems, such as Chartio Data Explorer [9] and Microsoft Power BI [26], is a form-based interface to insert a join. In the former, two queries can be merged clicking on a button to popup a form which can be used to select the merge type and the first columns that will be merged using dropdowns [9] (Figure 32a). Also, if there are null values on the merge related columns, there is an option to include or not the null values match rows [10]. Similarly, the latter provides a button to merge queries that opens a modal where can be chosen the attributes that will be used on the merge (viewing also a table preview) and the join kind, using a dropdown [27] (Figure 32b).

Tableau Prep [47] provides two options to start a join between two tables: clicking on

a "add join"hover button above the table visual representation with suggestion of related tables, or merge the visual representation of two tables using a drag and drop action. After this selection, the inner join type is selected automatically by the system according to the tables relationship. [45] [44] However, the user can configure the join in a dedicated section (Figure 32c) where it is possible to define the join type using a Venn Diagram, to manage the join clauses using dropdown lists to select the fields, including also some join clause recommendations based on the database schema. Moreover, a summary of the join result that contains counters with the values included and excluded by each table, in a visual way using diagrams is provided. Finally, there is presented a list of the values included and excluded, where the red values represent the values excluded, as well as a preview of the join result [45].



(a) Chartio Data Explorer [9] (b) Microsoft Power BI [27]



(c) Tableau Prep [45]

Figure 32: Data merging approaches.

Filtering Criteria:

To represent and manage the filtering criteria of the queries, usually is used text to indicate the logical conditions. However, some systems have been tried to optimize the usability of this process, helping the user in the specification process through some autocompletes to diminish the necessity to remember all the syntax and the name of the

functions. Besides, some systems organize the conditions in order to be more understandable the overall conditions of the query. One example of this, is the strategy adopted by Devart dbForge Query Builder [16] that represents the where and having clauses in a tree where the user can organize the conditions into the groups they want [14].

Furthermore, some query systems that also allows the user to view the query result, provide a shortcut in the column name to insert filters in, to change the query while are viewing the results. In this way, the user can apply a filter and view its effect at the following moment. Power BI Query Editor [26] is an example of a system which uses this approach.

Different interfaces can be used to select filters regarding the concerned data type, using the user interface to provide better support in the definition of the filtering criteria. For example, if a filter will be applied to constraint a date, a date input box with a visual calendar can be useful to simplify the date typing. In this way, some softwares, such as Chartio Visual SQL [11] and Chartio Data Explorer [9], not only helps in the data typing, but also provide dropdown lists that contain operators that can be applied to the referred data type (e.g. less than, contains, like, etc) to helps the user in the boolean operator specification [9] [12].

Tableau Prep [47] follows this in a more strict way, distinguishing between data types when filtering criteria are indicated. It provides different forms depending on the data type. The main difference of this system is that not only provides a more diversity of controls, as integrating range selectors, radio button, and the option to include or exclude fields through an action accessible near its value, but also gives to the user to option to use a calculation form where the interaction style is more textually and more extensible [46].

Sorting Criteria:

3.2 Data Visualization

3.3 Data User Exprerience and Expressiveness

3.4 Discussion

Chapter 4

Proposed Solution

After the problem contextualization, and the description of the related concepts, techniques and studies, this chapter presents the work done so far. Starting with a requirements analysis which includes user interviews and extraction of metrics to conclude in a quantitative way what are the cases where the SQL is used. Continuing with the priority assigned for each problem based on the requirement analysis. A current progress state of the solution developed will be presented, describing the actual development state and a foreseeing schedule of the work plan for the remaining time until the final of the project.

4.1 Requirements Analysis

The project has started with an exploration of the OutSystems Platform data querying tool, the Aggregates, to understand its functionalities and visual approaches used to perform queries visually, as has been described in section 2.2.2. Meanwhile, not only were identified the expressiveness problems of the visual language, previously mentioned in section 1.3, but also usability problems could be perceived in the performance of available actions.

After that, user interviews have been prepared to explore better the barriers of the tool and understand the impact of the problems in user actions. So, in the interviews, after perceiving the background of the participant, more directed questions were asked to pick up the first responses of the users about the advantages and disadvantages of the visual query tool. This approach promotes that users talk about the more impactful problems for them and the main situations where this tool is useful. Next, it was asked what are the situations when they use SQL to obtain insights about the reason to not use the visual tool to perform these queries. Also, were required that users present examples to explain the problems they have, whenever possible, because this strategy can be useful to understand

the impact level of the problem and sometimes show other problems not referred. In addition, when the users did not mention anything about expressiveness problems, direct questions about if they never needed to use these operations in Aggregates were colocated to cover the possibility of forgetting to mention that aspect.

The results reveal that the most novice users, with less than six months of experience, feels that Aggregates are easy to use and covers his necessity, referring also that is easier to learn than SQL. The most experienced users, who use the OutSystems platform to develop applications every day for professional purpose and have a technological background, reported that in the visual tool they cannot have a good understanding of the global view of the query, mainly if many tables, columns and business rules are involved. Switch between tabs in the interface to view the data sources, and the filtering and sorting criteria were other issue presented, as well as the lack of control on the query output [1]. Asked about the aggregation functions, which were implemented when Simple Queries have been replaced by Aggregates, they do not refer any problem with the approach interaction strategy adopted, not considering these new features as a problem that blocks the use of the tool. Furthermore, other usability problems that decrease the users' satisfaction have been pointed out, like the difficulty to search for a column in the query result, when there are many columns.

The problem analysis was not limited to users interviews, having been complemented with an analysis of the queries executed on the cloud due to perceiving the quantitative representation of the problems detected in the current progress analysis and inspecting the results of the interviews. The queries analysed, which have been extracted in July 2019 from customers' projects, were built using Advanced Queries [2].

Firstly, the data set is composed of 214.400 statements, but only 60.8% were used in this study since only the queries are important for this context and not other instructions like inserts, updates, deletes and transactions. As the last set also has duplicated queries, these were removed and the final queries set used to extract information comprehends 67.828 queries. The operators and clauses were figured out using a SQL Parser developed in JavaScript [3].

After the abstract syntax trees of the queries were obtained in a JSON file, these were analysed in a program to count the operators and the clauses that are present in the queries. A first analysis can measure the operators that are not supported by the visual tool, since the textual language are the only way to perform these operations. Figure 41 summarizes the number of the queries that contain these operations, where the intersection is not null, so there are queries that have two or more of the indicated operations. Thus, each percentage represents the subset of queries that have the operation inside, no matter if they have other operations or not.

Furthermore, it was measured how many queries are performed using the textual

[1] As referred on section 2.2.2.2, when a user adds an entity to an Aggregate, all its attributes are added automatically and if the user hides them the output of the query does not change.

[2] The option of the OutSystems Platform, invoked on section 2.2.2.1, that allows the query design in a

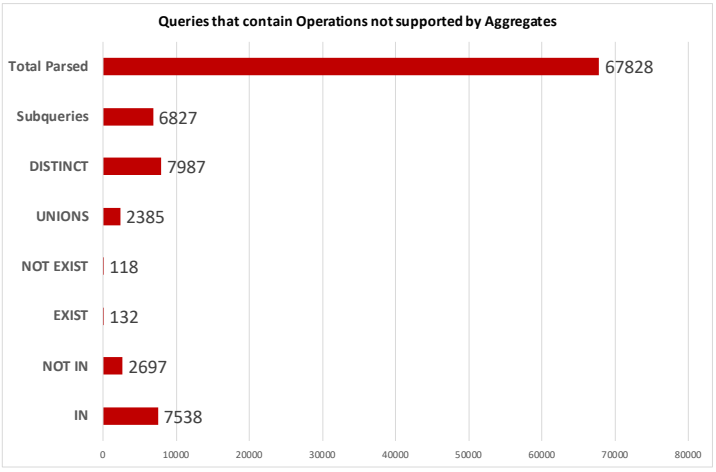


Figure 41: Number of the queries that contain operations not supported by Aggregates

language but could be designed using Aggregates. Figure 42 represents a chart of the results obtained that divide the queries performed in three categories:

- Not Supported: Queries which include operations not supported by Aggregates, such as IN, NOT IN, EXIST, NOT EXIST, Unions, Distincts and Subqueries;
- Supported by Aggregates: Queries which could be designed totally using Aggregates. These are divided into two subcategories:
  - Simpler: Queries which include only operations supported by aggregates excluding the indication of sorting criteria and the use of aggregation functions (e.g. GROUP BY or SUM, AVG, MIN, MAX, COUNT);
  - More Complex: Queries that are supported by Aggregates excluding the above (simplers).

Since user interviews suggested that aggregation functions and sorting criteria were not the main problems. The queries supported by Aggregates were divided to compare the quantitative analysis with the qualitative analysis extracted in interviews. This is important because these operations can be indicated using a different interaction technique where the user changes the query when he is interacting with the query result, as mentioned in section 2.2.2.2. Thus, this division can be useful to validate the approach used to indicate these aspects of the query.

textual way using a language based on SQL.

[3]js-sql-parser repository page: https://github.com/JavaScriptor/js-sql-parser

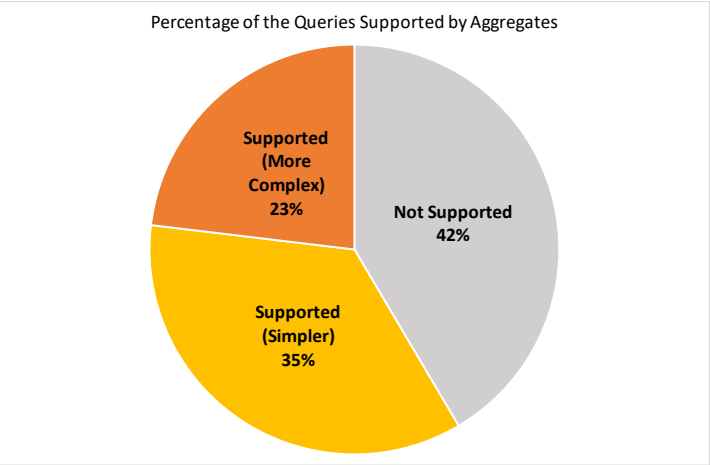


Figure 42: Relationship between the queries that could be designed using Aggregates and the queries which this tool does not support.

4.2 Scope Definition

The results of the quantitative analysis show that there are in conformity with the qualitative analysis results (user interviews). Both of the analysis, leads that the main priority of the project should be the usability improvement of the visual querying tool interface, since not only is the main concerned pointed out by the users, but also has metrics that sustain that hypothesis. In the set of the queries analysed, around 58.5% could be totally designed using Aggregates, could be an indication that the lack of SQL expression is not the major problem.

Although the results show a small set of queries that are not supported, these expressiveness problems of the visual language can influential the users to change to the textual language, because many users do not like to use different systems to perform identic tasks. So, as the visual language does not support some features users may want to change to the textual language where can perform all the queries.

Under the circumstances, the results were discussed together with the stakeholders to order by priority the problems to be tackled. So, it was decided that the main problem is the usability of the system and regarding the improvement of the visual language expressiveness it was defined that the main priorities are the IN / NOT IN and the DISTINCT operations. Thus, along the project, the main focus will be the improvement of the usability and if is possible to conciliate with the main goal, these two new features of language expressiveness will be studied.

4.3 Proposed Implementation

Following, will be detailed the intended project, indicating all the problems identified and all the approached that will be adopted.

4.4 Work Plan

Bibliography

[1] G. D. Abowd and R. Beale. "Users, systems and interfaces: a unifying framework for interaction." In: *HCI'91: People and Computers* VI (1991), pp. 73–87.

[2] A. Alshamrani and A. Bahattab. "A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model." In: *International Journal of Computer Science Issues (IJCSI)* 12.1 (2015), p. 106.

[3] Balsamiq. *Balsamiq*. URL: https://balsamiq.com/ (visited on 01/29/2020).

[4] K. L. Berg, T. Seymour, and R. Goel. "History of databases." In: *International Journal of Management & Information Systems (IJMIS)* 17.1 (2013), pp. 29–36.

[5] T. Catarci and G. Santucci. "Diagrammatic vs textual query languages: a comparative experiment." In: *Working Conference on Visual Database Systems*. Springer. 1995, pp. 69–83.

[6] T. Catarci, M. F. Costabile, S. Levialdi, and C. Batini. "Visual query systems for databases: A survey." In: *Journal of Visual Languages & Computing* 8.2 (1997), pp. 215–260.

[7] D. D. Chamberlin and R. F. Boyce. "SEQUEL: A Structured English Query Language." In: *Proceedings of the 1974 ACM SIGFIDET (Now SIGMOD) Workshop on Data Description, Access and Control*. SIGFIDET '74. Ann Arbor, Michigan: Association for Computing Machinery, 1974, 249–264. ISBN: 9781450374156. DOI: 10.1145/800296.811515. URL: https://doi.org/10.1145/800296.811515.

[8] Chartio. *Chartio*. URL: https://chartio.com/ (visited on 02/07/2020).

[9] Chartio. *Chartio Data Explorer | Documentation*. URL: https://chartio.com/docs/data-explorer/ (visited on 02/07/2020).

[10] Chartio. *Chartio FAQs: Joining Data Across Databases*. URL: https://chartio.com/docs/visual-sql/actions/ (visited on 02/10/2020).

[11] Chartio. *Chartio Visual SQL (beta) | Documentation*. URL: https://chartio.com/docs/visual-sql/ (visited on 02/07/2020).

[12] Chartio. *Visual SQL Actions | Chartio Documentation*. URL: https://chartio.com/docs/visual-sql/actions/ (visited on 02/10/2020).

[13] H. Desurvire, J. Kondziela, and M. E. Atwood. "What is Gained and Lost When Using Methods Other than Empirical Testing." In: *Posters and Short Talks of the 1992 SIGCHI Conference on Human Factors in Computing Systems*. CHI '92. Monterey, California: Association for Computing Machinery, 1992, 125–126. ISBN: 9781450378048. DOI: 10.1145/1125021.1125115. URL: https://doi.org/10.1145/1125021.1125115.

[14] Devart. *Building WHERE or HAVING Clause*. URL: https://docs.devart.com/querybuilder-for-sql-server/building-queries-with-query-builder/building-where-and-having-clause.html (visited on 02/10/2020).

[15] Devart. *Making Joins Between Tables*. URL: https://docs.devart.com/querybuilder-for-sql-server/building-queries-with-query-builder/making-joins-between-tables.html (visited on 02/10/2020).

[16] Devart. *Query Builder Tool in dbForge Studio for SQL Server*. URL: https://www.devart.com/dbforge/sql/studio/query-builder.html (visited on 02/07/2020).

[17] A. Dillon and C. Watson. *User analysis in HCI: the historical lesson from individual differences research*. 1996. URL: http://hdl.handle.net/10150/105824.

[18] A. Dix, A. J. Dix, J. Finlay, G. D. Abowd, and R. Beale. *Human-computer interaction*. Pearson Education, 2003.

[19] J. Gehrke and R. Ramakrishnan. *Database management systems*. McGraw-Hill, 2003.

[20] Google. *Google Sheets*. URL: https://www.google.com/sheets/about/ (visited on 02/05/2020).

[21] H. Henriques, H. Lourenço, V. Amaral, and M. Goulão. "Improving the Developer Experience with a Low-Code Process Modelling Language." In: *Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*. MODELS '18. Copenhagen, Denmark: Association for Computing Machinery, 2018, 200–210. ISBN: 9781450349499. DOI: 10.1145/3239372.3239387. URL: https://doi.org/10.1145/3239372.3239387.

[22] ISO. *ISO 9241-11:2018(en) Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. 2018. URL: https://www.iso.org/obp/ui/#iso:std:iso:9241:-11:ed-2:v1:en (visited on 01/27/2020).

[23] P. Jennifer, R. Yvonne, and S. Helen. "Interaction design: beyond human-computer interaction." In: *NY: Wiley* (2002).

[24] J. V. M.A. "I. On the diagrammatic and mechanical representation of propositions and reasonings." In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 10.59 (1880), pp. 1–18. DOI: 10.1080/14786448008626877.

[25] Microsoft. *Microsoft Excel*. URL: https://products.office.com/en/excel (visited on 02/05/2020).

[26] Microsoft. *Microsoft Power BI*. URL: https://powerbi.microsoft.com/en-us/ (visited on 02/07/2020).

[27] Microsoft. *Tutorial: Shape and combine data in Power BI Desktop*. URL: https://docs.microsoft.com/en-us/power-bi/desktop-shape-and-combine-data (visited on 02/07/2020).

[28] Mockingbird. *Mockingbird*. URL: https://gomockingbird.com/home (visited on 01/29/2020).

[29] J. Nielsen. "Iterative user-interface design." In: *Computer* 26.11 (1993), pp. 32–41. ISSN: 1558-0814. DOI: 10.1109/2.241424.

[30] J. Nielsen. *Usability Engineering*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1994. ISBN: 9780080520292.

[31] J. Nielsen and R. Molich. "Heuristic Evaluation of User Interfaces." In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. CHI '90. Seattle, Washington, USA: Association for Computing Machinery, 1990, 249–256. ISBN: 0201509326. DOI: 10.1145/97243.97281. URL: https://doi.org/10.1145/97243.97281.

[32] OutSystems. *Evaluation Guide (Developing with OutSystems*. URL: https://www.outsystems.com/evaluation-guide/developing-with-outsystems/ (visited on 02/01/2020).

[33] OutSystems. *Low-Code Development Platform for Enterprise Applications*. URL: https://www.outsystems.com/platform/ (visited on 01/31/2020).

[34] OutSystems. *Service Studio Overview - OutSystems 11 Documentation*. URL: https://success.outsystems.com/Documentation/11/Getting_Started/Service_Studio_Overview (visited on 02/02/2020).

[35] OutSystems. *What's new in OutSystems Hub Edition 2.0*. 2003. URL: https://drive.google.com/file/d/1wkKESumKhJbwK6aoeW2DGU4-TMq-snOp/view (visited on 02/03/2020).

[36] OutSystems. *What's new in OutSystems Hub Edition 2.2*. 2004. URL: https://drive.google.com/file/d/0B7C37RyL27_oNHF4UmFRX3pFMlpMTVlCWnV5dzJCcmhRS2tj/view (visited on 02/03/2020).

[37] OutSystems. *Agile Platform What's New in Version 5.0*. 2009. URL: https://www.outsystems.com/home/document-download/542/31/0/0 (visited on 02/03/2020).

[38] OutSystems. *OutByNumbers - Benchmark Overview Repor*. Tech. rep. 2013.

[39] P. G. Polson, C. Lewis, J. Rieman, and C. Wharton. "Cognitive walkthroughs: a method for theory-based evaluation of user interfaces." In: *International Journal of Man-Machine Studies* 36.5 (1992), pp. 741 –773. ISSN: 0020-7373.

[40] M. Revell. *What Is Low-Code?* 2020. URL: https://www.outsystems.com/blog/what-is-low-code.html (visited on 01/24/2020).

[41] T. Simões. *What's (Not) New in OutSystems: A Product Timeline*. 2018. URL: https://www.outsystems.com/blog/posts/not-new-product-timeline/ (visited on 02/03/2020).

[42] C. Souther. *Low-Code vs. No-Code: What's the Real Difference*. 2019. URL: https://www.outsystems.com/blog/posts/low-code-vs-no-code/ (visited on 01/25/2020).

[43] C. Stephanidis. "User interfaces for all: New perspectives into human-computer interaction." In: *User Interfaces for All-Concepts, Methods, and Tools* 1 (2001), pp. 3–17.

[44] Tableau. *Add More Data in the Input Step - Tableau*. URL: https://help.tableau.com/current/prep/en-us/prep_add_input_data.htm (visited on 02/10/2020).

[45] Tableau. *Aggregate, Join, or Union Data - Tableau*. URL: https://help.tableau.com/current/prep/en-us/prep_combine.htm (visited on 02/10/2020).

[46] Tableau. *Filter Your Data - Tableau*. URL: https://help.tableau.com/current/prep/en-us/prep_filter.htm (visited on 02/10/2020).

[47] Tableau. *Tableau Prep*. URL: https://www.tableau.com/products/prep (visited on 02/07/2020).

[48] Tableau. *What's New in Tableau Prep Builder*. URL: https://help.tableau.com/current/prep/en-us/prep_whatsnew.htm (visited on 02/07/2020).

[49] M. Unsöld. "Measuring Learnability in Human-Computer Interaction." Master's thesis. 2018.