ISSN: 2754-6659

Journal of Artificial Intelligence & Cloud Computing

Research Article

Open Access

Comparison of Software Development Solution Implementations in Lightning Flow Builder and Apex Programming Language in Salesforce Technology

M Grabowski* and M Plechawska-Wójcik

Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland

ABSTRACT

The recent rapid development of the Salesforce platform has induced an expansion of no-code solutions to make automation available for less technical audience. This study examines similarities and differences between Apex programming and one of the no-code solutions on the Salesforce platform, Flow Builder. The research compares the implementation of the same applications using Apex language and Flow Builder in terms of operating time to fulfil requirements (first part) and execution time of both applications (second part).

The applications were classified into two categories regarding the complexity of the solution, namely simple and intermediate, so that the research could thoroughly verify the impact of the amount of code and the simplification of programming concepts Flow Builder allows for. In the first part of the research, the majority of the developers who took part in the research managed to implement the simple application much quicker using Apex than Flow Builder. The intermediate application needed much more Apex code and made use of concepts like bulkification of records, which are automated in Flow Builder, so it was much easier to develop the second application in Flow Builder than Apex. In the second part of the research, execution times of the applications using Apex and Flow Builder were compared, and the findings show the impact of Flow Builder "behind the scenes" automation, which was the root cause of its more complex solutions slowdown.

***Corresponding author**
M Grabowski, Department of Computer Science, Lublin University of Technology, Nadbystrzycka 36B, 20-618 Lublin, Poland.

Received: January 09, 2024; **Accepted:** January 11, 2024; **Published:** January 23, 2024

Introduction

By the term of Salesforce, it is usually meant the software, technology or platform designed by the company of the same name. Since it was established, Salesforce has insisted on overtaking the marketplace of the type of software named SaaS (Software as a service). SaaS means that the customer gets the desired applications which are kept on one of service provider's servers. The customer, therefore, uses the computing power and disk storage on demand which belongs to the service provider [1, 2].

For the last two dozens years, several main applications to handle customer relationships on the Salesforce platform has been launched: Sales Cloud, Service Cloud, Marketing Cloud, Experience Cloud. These applications are the part of the CRM (Client Relationship Management) built on the basis of the Salesforce cloud-based platform [3]. CRM is a strategy or a philosophy for entrepreneurs to improve sales processes, quality of service and marketing actions, and any other client-related actions in order to focus on clients' needs. The CRM systems and tools exemplified by the Salesforce platform focus on improving the way enterprises work to make customers more satisfied and improve the efficiency of business processes.

The former study compared the time the developer needs to implement solutions using Apex programming language and Flow Builder, a no-code tool. It was agreed that the whole two parts of the study should compare the implementation and execution times of the sample solution. Implementation time means the time programmers spent on a successful implementation of the solution using two tools: Apex programming language and Flow Builder. Execution time means the time the Salesforce platform needs to run the solution without an error-so it is the time between the start and the end of script execution at the back-end side. This research analyzed two types of applications to be deployed by developers: simple and more complex.

The latter study aimed at comparing the execution times of the sample applications (possibly most optimized) that were made both in Apex and Flow Builder. The comparison of the execution times was essential to establish which tool of those two is faster and more efficient in use.

Flow Builder itself is a relatively new tool. It was introduced as a part of the Spring '19 release and that update has replaced the older tool, Flow Designer. Released in 2012, Flow Designer was completely Flash dependent, so Salesforce has decided to launch a new solution. The Salesforce platform is not widely described in the scientific community and there are few researches that are concerned with this technology. This study is to share the innovatory findings on the differences between programmatic solutions (code only) and no-code solutions like Flow Builder in Salesforce.

Instances and Organizations

The Salesforce platform is divided into servers called *instances* which then are partitioned into *organizations* or *orgs*. The instances are physical divisions, and the organizations are only logically separated environments where users from the same organization, like a company or an enterprise, can log in. Therefore, tens or hundreds of the organizations may exist on a single instance and operate on multitenant resources from a single server.

Multitenancy is the main domain of Salesforce because the resources are available for a lot of customers. Users are assigned to the profiles which use adequate licenses to utilize the features of the Salesforce platform. Every internal organization built on the instance of Salesforce would use the full computational capabilities and the shared resources, but not only the part of the server that was leased. At the same time, data security is guaranteed, and every organization is separated from each other [4]. Only the metadata and data from the organization would be accessed by that organization, and there is no possibility to obtain any information about the other organizations.

Users on the Salesforce Platform

On Salesforce instances, users are created using unique usernames. The username cannot be duplicated on the whole platform because users can log into any Production instance or any sandbox instance using universal URL dedicated for those two types of instances. For the production instance, the URL is https://login.salesforce.com/, whereas for the sandbox instance, it is https://test.salesforce.com/. Using the unique username and password, the Salesforce platform automatically redirects the user to the correspondent organization he or she was registered. It means that the user should have an actual suffix at the end of the username, so that it could be clear which organization the user relates to. What is more, the username is built on a domain name convention like electronic addresses, for instance john_smith@company.com.sit. The suffix at the end means that this user comes from the SIT (System Integration Testing) organization.

Declarative Tools and Programming Languages

The Salesforce Platform, formerly known as Force.com, is the leading technology made by Salesforce, Inc. It provides a multitude of tools, applications and widgets for developers to build applications based on the Salesforce user interface [5]. The declarative tools mostly used to build automations and user interaction screens are Flow Builder, Process Builder and Workflow Rules. Custom applications are also built utilizing Lightning Components (browser side) and Apex language (server-side) [6].

Lightning Components is being divided into two frameworks: LWC (Lightning Web Components) and Lightning Aura Components. Both of those frameworks are based on the conceptions of stateful client and stateless server. It means that the state of the application is being kept on the client's (browser) side and not the server. The server is only designed to retrieve and save the data at the Salesforce's infrastructure side. Lightning Web Components and Lightning Aura Components use JavaScript language to run actions on the browser.

Literature Overview

There are a couple of studies that verify the impact of utilizing no-code tools instead of conventional programming solutions. One of the researchers, Virta, compares the relation of low-code development tools, like Flow Builder (formerly known as Cloud Flow Designer) and Process Builder, to standard software development including Apex programming language and Lightning Components framework. The goal was achieved by interviewing the employees in one of the Salesforce consulting companies in Finland. It was confirmed that low-code solutions are double-edged.

The benefits were indeed its fast development and the understanding of the processes by non-technical people like clients. On the other hand, when the system grows and processes start to be more complex, there is a risk that the whole process of development will be too expanded and hard to maintain. The interviewees also mentioned the lack of unit tests in the Flow Builder and Process Builder tools. Unit tests are required for every Apex code that is being developed to check whether the reliability of the code is achieved. The main drawback of the low-code solutions is also their poor performance in more complex systems [7].

The whole study examines only the employees' opinions expressed in the survey and does not count the indicators like number of clicks and presses on a keyboard and the time consumed to implement the applications or execute the processes. It may seem to be unsatisfactory and still more research could be recommended to achieve more calculable results.

A practical approach is also taken by Miącz in his research into the differences among the parameters of using point and click solutions and the Apex code. It was compared how time consumption on the Salesforce platform depends on the quality of bandwidth and the type of the tool used to develop the solution. It was found that the point and click solutions are less sized in megabytes and execute less SOQL queries probably because of collecting all the required conditions and running them at once [8]. Nevertheless, that research does not show the essential differences in the time needed to develop both conventional programming and the no-code solutions.

In Salesforce.com

The Development Dilemma, the authors describe mainly the history and achievements of the first years of the Force.com technology. That study examines the methodology of project management in Salesforce.com's development process and how the technology has risen to life [5].

The usability of the Salesforce platform and the capabilities of tools have been frequently researched. The authors from the Computer Science and Engineering Galgotias University discuss the benefits of multitenancy in the CRM applications [9]. The result of this multinational collaboration is the paper 'A real-time service system in the cloud' that describes the details of cloud computing architectures by means of Salesforce as an example. The data model and all technological details are usefully referred to [10].

The whole guides treating about the nature of software development in the Salesforce platform are an excellent source of knowledge [3,11,12]. As for the declarative tools like Flow Builder, Process Builder and Workflows, there are also the sources that could help in understanding that matter [6].

An interesting reference in the literature might be Kermanchi's thesis which compares the experiences of 18 developers who used both Apex programming and low-code solutions like Flow Builder and Process Builder [13]. The experiment by Kermanchi was verifying the participants' thoughts and feelings about using the tools mentioned above.

The findings have shown that the developers turned out to be skeptical about the low-code development tools because they had had the background and knowledge about a low scalability of these solutions and reviewed programmatic tools as more suitable for complex implementations. In conclusion, in programmers' thoughts Apex is more efficient and useful for the processes that are to be extended in the future.

The use of Apex language and the cases of how the Apex classes are used to build real-time service systems in Salesforce are described in one of conference papers [14]. The authors examine the ways Apex language is being used, and what this language enables the programmer to do. Apex is also compared to the other programming languages and the authors let the reader know the advantages and uniqueness of Apex in comparison to the other languages. It is important to mention that the limitations of the Salesforce platform and its back-end language are also presented in this paper to indicate that the potential programmer works with the cloud.

Basic Concepts of Work with Apex Programming Language
Apex is often being compared to Java as its main principles are based on an object-oriented programming paradigm and its model of class construction is nearly identical to the one known in Java. The language is server-side and strongly typed which means that the type of every variable needs to be predefined and stay the same type in that context [4].

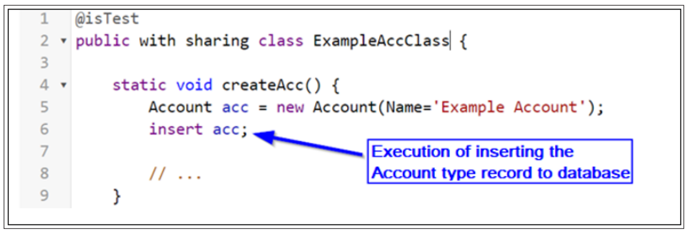
Moreover, Apex lets developers directly operate on database with DML (Data Manipulation Language) operations and SOQL (Salesforce Object Query Language) without a need to treat these actions as an asynchronous operation which is vastly convenient and allows the engineer of the system not to use callbacks or promises (Figure 1). These operations work like an await expression in JavaScript without having to use that expression. The actions invoked with Apex have direct access to data sources in the organization that the user is logged into. All results are filtered with accesses and permissions that the user has – the Salesforce platform is a supervisor on this account.

Salesforce has established the Execution Governors and Limits document for the use of shared resources so the access to use the multitenant platform is not monopolized. The Apex runtime obliges system engineers to comply with those limits. If they are not complied, the runtime throws an exception that cannot be omitted with try-catch blocks. For instance, exceeding the limit of the number of DML operations, the SOQL queries or execution time in a transaction (10 seconds for synchronous and 120 seconds for asynchronous with callouts to external systems) will throw a System.LimitException exception.

While working with Apex language, one has to be conscious of Salesforce API (Application Programming Interface) versions the platform uses for the identification of the execution context of the processes in the system (Figure 2). The API means the rules and the principles on how the applications or programs communicate with one another. The Salesforce API is a specification on how external applications should communicate with the Salesforce platform and send messages to it to request services or share data.

Within a single Salesforce organization, classes and triggers with different versions can coexist. The version may be changed in any moment. Classes with the same name though must not be saved upon a few different versions.

It is worth saying a word about the use of DML operations outside any loops. Developers ought not to do it at any circumstances because it is a way to the quick reaching to execution limits and throwing an exception. Rather than including a DML operation in the loop, the developer should add all records to the list in the loop, and the DML statement ought to be performed after the loop block.

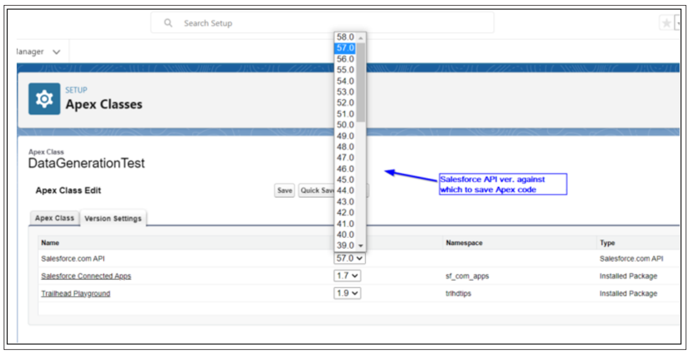


```
@isTest
public with sharing class ExampleAccClass {

    static void createAcc() {
        Account acc = new Account(Name='Example Account');
        insert acc;

        // ...
    }
```

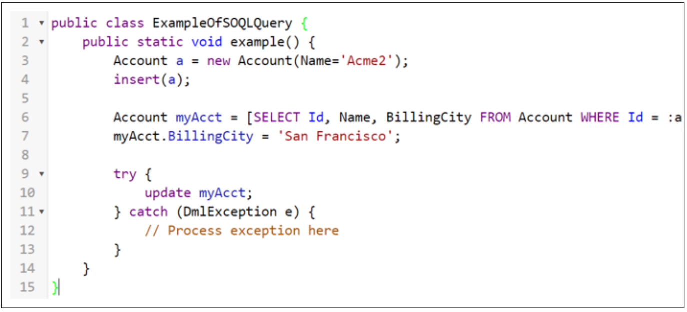
Figure 1: Sample of a DML Operation in Apex Language -The Unit Test Method of Inserting Account Records



Name		Namespace	Type
Salesforce.com API	57.0		Salesforce.com API
Salesforce Connected Apps	1.7	sf_com_apps	Installed Package
Trailhead Playground	1.9	trlhdtips	Installed Package

Figure 2: Choice of Salesforce API Versions for the Apex Class

SOQL as an Auxiliary Language for Apex SELECT Operations with Database
One could possibly notice that the DML operations which could be insert, update, upsert, delete, merge and undelete do not contain the 'select' queries to the database, because those queries are performed with SOQL. Every SOQL statement is included in Apex using square brackets and the returned value of that statement would be either a list of records or a single record (Figure 3). A system engineer has to be careful though, because if the returned value is a list of more than one record, the assignment of this list into the variable of type record and not a list, would result in an exception. A SOQL query in its form would be considered as a dialect of SQL (Structured Query Language).



```
1  public class ExampleOfSOQLQuery {
2      public static void example() {
3          Account a = new Account(Name='Acme2');
4          insert(a);
5  
6          Account myAcct = [SELECT Id, Name, BillingCity FROM Account WHERE Id = :a
7          myAcct.BillingCity = 'San Francisco';
8  
9          try {
10             update myAcct;
11         } catch (DmlException e) {
12             // Process exception here
13         }
14     }
15 }
```

Figure 3: Line 6 of the above code contains an example of a SOQL query. At the end of the query, there is a binding to Id of "a" variable inserted in the 3rd line.

Developer Tools Designed for Apex

With the Salesforce platform, one of the tools delivered was Developer Console considered an editor for Apex classes, Aura components, debugging of code, executing tests, examining the unit test code coverage and viewing the logs on several different levels of complexity. Developer Console has even more possibilities. It allows us to execute direct SOQL queries. With the use of Developer Console, one might execute an anonymous Apex code which is essential to verify the validity of a newly written code. The user interface is quite simple and convenient. There are some disadvantages though. The layout and background design cannot be changed, so a programmer is enforced to work with a light-oriented theme of the environment in the user interface, not having the possibility to choose a dark theme which makes possible to work at night without eye strain. The Lightning Web Components framework components cannot be developed using Developer Console. Developers are encouraged to use Visual Studio Code instead.

Yet another developer tool is Visual Studio Code used to manage the metadata of the Salesforce platform and develop solutions. This tool has a variety of extensions dedicated, especially for use of Salesforce developers. With these extensions, it enables to do exactly the same operations which are enabled using Developer Console and even more. Also, the possibilities of changing the design of this tool allows a developer to feel confident about the conditions of developing solutions. With the Salesforce Extension Pack, Visual Studio Code is able to create components for the LWC framework and deploy them to the platform. The extensions allow us to highlight the syntax, integrate with the remote Salesforce platform, run asynchronous tests on the platform without the use of command line, execute SOQL queries and more.

No-Code Solutions to Automate the Processes on the Salesforce Platform

Several no-code tools have been introduced by Salesforce over the past twenty years. At the very beginning of Force.com, there were Workflows, and their basic application was to provide a self-explanatory, easily maintainable instrument for business logic automations. The Workflows are effortless to utilize, because they suffer from absence of many features found in the newer tools like Process Builder and Flow Builder. For instance, the Workflow does only evaluate a single output at a time with only a single set of entry criteria, cannot perform complex actions, is not able to delete any records and may update only related records' fields. Flow Builder itself does enable all of these actions.

Process Builder is one more no-code solution tool which may be applied to automate more complex business processes than the Workflow. This tool is capable of evaluating one condition after another and after fulfilling the criteria, the operations defined for that criteria are executed. Processes built with Process Builder might start with:

- The values on specified fields change (record gets updated or inserted),
- An outbound message is received from the channel of platform events,
- The process is launched by another process.

Operations that can be launched with Process Builder are, for example: create a record, update a record, execute a Quick Action on the Salesforce platform, launch a Flow, insert a post in Chatter, invoke an Approval, execute Apex code. Process Builder is not limited to executing immediate actions. The other type of operations is scheduled actions which enable time shifting of action executions.

Both Workflows and Process Builder are getting deprecated. Creation of new Workflows was turned off with the Winter '23 release of the Salesforce platform, whereas the ability of creating new processes utilizing Process Builder is planned to be disabled in the Summer '23 release.

Flow Builder as the Most Convenient Tool for No-Code Processes Automation

Lightning Flow is a type of process built using a declarative tool called Flow Builder. This no-code solution does not need any Apex code, however adding a code is possible by adding an Apex code block in the flow schema to send the data from a flow to one of the Apex methods in the Apex class – then the execution of the process continues in the Apex method. The way of building processes using Flow Builder is fairly simple. There are two modes setting the mechanism of building the flow. If the mode is set to Freeform, one should add the blocks to a flow by the drag'n'drop feature from the block list on the left in the Toolbox. If the mode is set to Auto-Layout, adding blocks is possible using the "+" icon on the appropriate line between the blocks and then choosing the type of block.

Lightning Flow resembles flowcharts diagrams used to describe the algorithm's step-by-step approach. The Start and End blocks are marked with a circle at the top and bottom of the flow. The conditional operations (Decision blocks) are diamonds, and the other operations like assignment of the variable, a loop or CRUD (Create Read Update Delete) operations are marked with a square (Figure 4). CRUD operations are the four basic executable operations in the database using the statements: *insert* for creation of records, *select* for reading the records, *update* for modifying the records or *delete* for removing the records.

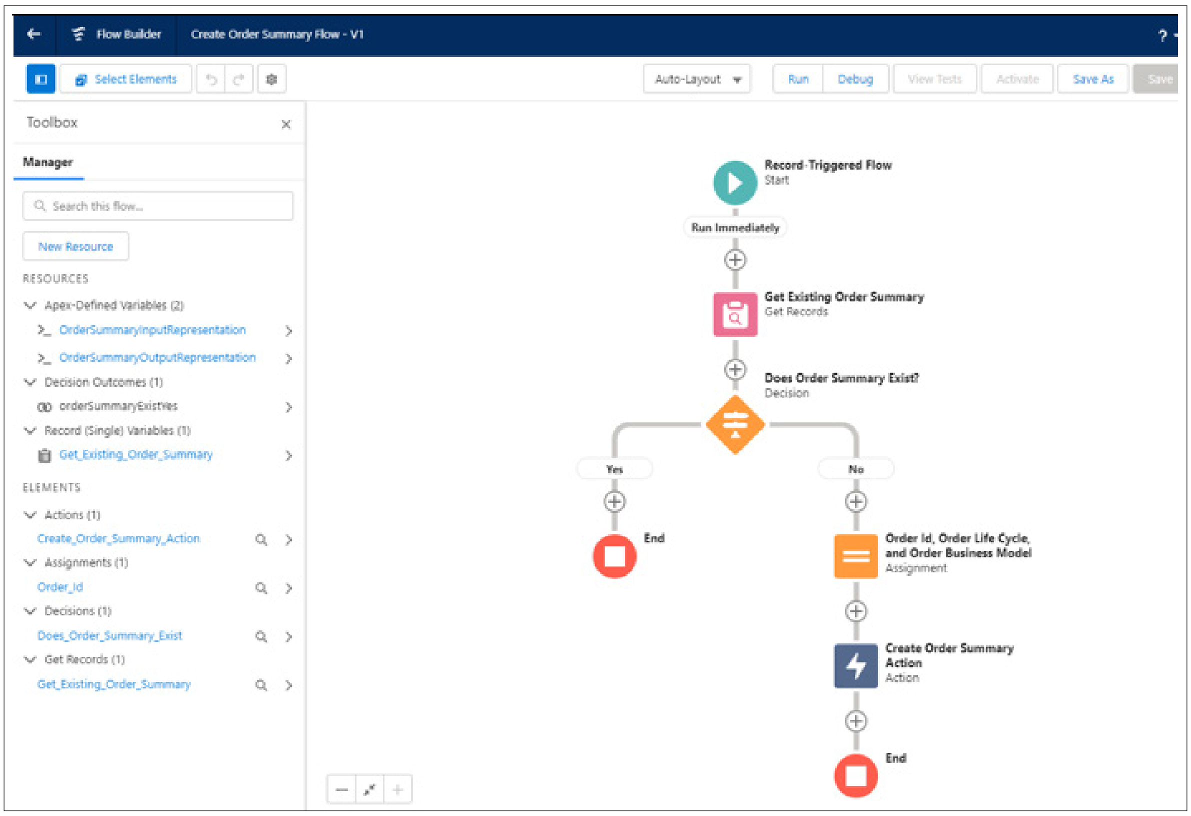
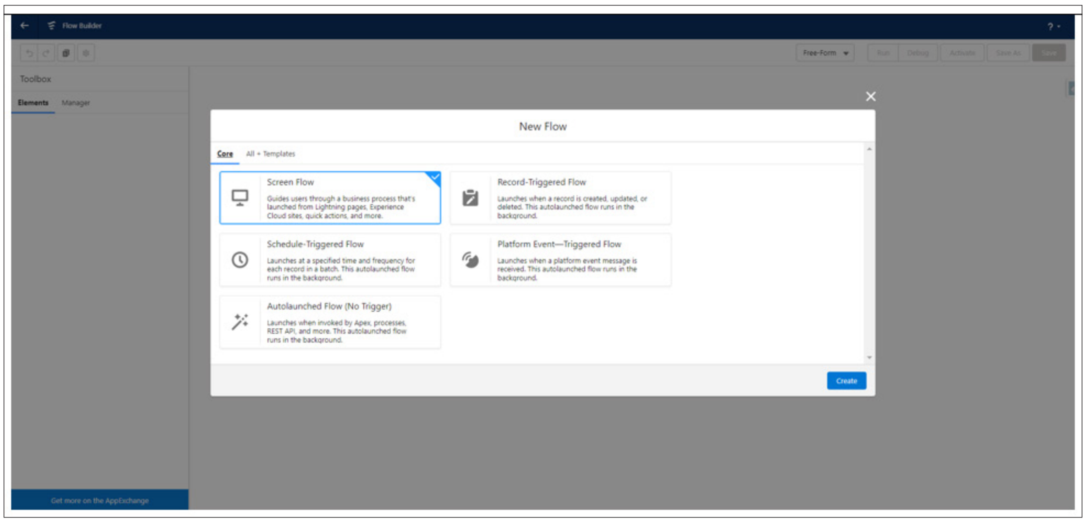

Flow Builder
Create Order Summary Flow - V1
Select Elements
Auto-Layout
Run
Debug
View Tests
Activate
Save As
Save
Toolbox
Manager
Search this flow...
New Resource
RESOURCES
Apex-Defined Variables (2)
OrderSummaryInputRepresentation
OrderSummaryOutputRepresentation
Decision Outcomes (1)
orderSummaryExistYes
Record (Single) Variables (1)
Get_Existing_Order_Summary
ELEMENTS
Actions (1)
Create_Order_Summary_Action
Assignments (1)
Order_Id
Decisions (1)
Does_Order_Summary_Exist
Get Records (1)
Get_Existing_Order_Summary
Record-Triggered Flow
Start
Run Immediately
Get Existing Order Summary
Get Records
Does Order Summary Exist?
Decision
Yes
No
End
Order Id, Order Life Cycle, and Order Business Model
Assignment
Create Order Summary Action
Action
End

Figure 4: User interface of Flow Builder with Auto-Layout mode. The Picture Shows One of the Standard Flows Added with the Basic Features of the Salesforce Platform.

Flow Builder allows us to build a few basic types of Flow processes that depend on requirements (Figure 5). These can be the Screen Flows which are the interactive processes launched in the window that brings the user interface to communicate. This type of Flow has an additional type of the block, comparing to the other Flows. It is the Screen block which has all the possible kinds of form inputs and outputs for displaying values. Building the block of Screen is about dragging and dropping components and configuring them using inputs for metadata like the label of that input field, the API name, "required" modifier, a default value etc. It is possible to add custom inputs that were programmed using Aura and Lightning Web Components frameworks.

Another type of the Flow process is a Record-Triggered Flow. It is an automatically triggered process that activates on the insertion, update or deletion of the record of the selected object. This specific Flow may be compared to Apex Trigger. Apart from the type of CRUD operation and the name of object, it is needed to select the optimization type of the Flow of two: Fast Field Updates and Actions and Related Records. The Fast Field Updates optimization type is executing the process before the specified CRUD operation, and Actions and Related Records executes the process after the CRUD operation on the database. Both of those are used depending on if the record should already exist and have the identifier or the system should update the data before inserting the record to the database.



Screen Flow
Guides users through a business process that's launched from Lightning pages, Experience Cloud sites, quick actions, and more.

Record-Triggered Flow
Launches when a record is created, updated, or deleted. This autolaunched flow runs in the background.

Schedule-Triggered Flow
Launches at a specified time and frequency for each record in a batch. This autolaunched flow runs in the background.

Platform Event—Triggered Flow
Launches when a platform event message is received. This autolaunched flow runs in the background.

Autolaunched Flow (No Trigger)
Launches when invoked by Apex, processes, REST API, and more. This autolaunched flow runs in the background.

Figure 5: The Choice of Type of the Process for Initialization of the Flow

Other types of Flow processes are the Schedule-Triggered Flow, the Autolaunched Flow (No Trigger) and the Platform Event-Triggered Flow. The Schedule-Triggered Flow is executed in the certain type in the future based on other actions (e.g. a day after inserting the record) with a selected frequency. The Event-Triggered Flow would be a process that is launched when an event occurs on the Salesforce platform and the conditions are fulfilled. The events may be triggered by the Apex code, the other Flows or by external systems (using REST or SOAP queries). REST (Representational State Transfer) and SOAP (Simple Object Access Protocol) are protocols which enable the systems to be integrated with one another by exchanging the data between them. The last one type of the Flow, the Autolaunched Flow (No Trigger), is a flow that does not have the specification of the object and the CRUD operation required to launch the flow. This process can be launched by the Apex code, other processes or API call.

Possible Limitations of the Use of the Flow Builder

Using Flow Builder to implement solutions has its disadvantages which are the limits that do not exist when the developer chooses to implement the solution in Apex language. Below, there are several of the issues that a developer may encounter using Flows.

- Having implemented more than one process on the same object and of the same CRUD operation does not guarantee that the order of executing the processes is kept the same.
- Using of the Subflows and Apex actions does not produce the detailed information on the Execution Track Log.
- In the case of the automation of the Record-Triggered Flows, it is often noted that there is a poor performance of nested formula calculations. That issue becomes major when there are a lot of records being evaluated by the trigger (and there can be up to 200 records per batch) because formulas are being compiled and calculated serially, while the process is in operation [15].
- It is not possible to use the 'LIMIT' operator in SOQL queries without using the Apex code. The only build-in feature for Lightning Builder is to reduce the returned records to 1.
- It is not possible to use the 'IN' operator without using the Apex code.
- The RTF (Rich Text Field) inputs in HTML formatted texts are not supported well. The HTML tags <a>, <b>, <br>, <font>, <i>, <li>, <p>, <span>, <u>, <div> are converted into a text.
- There is no type of associative array or map in the Flow processes. In the case of the requirement to save key-value variables in a map, one should use an additional loop. It may be time-consuming and not so efficient.
- Flow Builder does not support UTF-8 encoding in text inputs.

Methodology of Research

The Comparison of Implementation Speed in the Apex Language and Flow Builder Applications

In the research 10 developers were asked to implement two applications: simple and compound. They got an exact instruction on what should be implemented at a suitable level of detail. The questions the developer was asked were answered before and during the time of an attempt of implementation. The developers could go back to the instructions at any moment of the experiment. They were familiar with the Salesforce platform and the environment as it was their self-prepared place of work. Their experience differed due to the years of working with the Salesforce platform: Junior (less than 2 years of experience), Regular (2-5 years of experience) and Senior (over 5 years of experience). The knowledge of the developers may vary but their overall experience of how to handle the issues with the technology in theory should be higher than the Regular and Senior skill levels.

Once the attempt began, the programmers had Visual Studio Code environment with the Salesforce Extension Pack installed and configured. The instance of the Salesforce platform used to perform the research was EU42. It was established that Apex unit tests are not necessary. As the unit tests are other programs to be developed independently of the rest of the application, it could take relatively much time. Once the developer was ready, the stopwatch was turned on. The stopwatch was a device with 10ms precision.

It was necessary to implement both applications, namely Apex language and Flow Builder in two environments, so, overall, a single developer created four applications in four different timespans. One of the developers did not finish the compound application for Apex and Flow Builder, and another one did not finish the Flow Builder compound application only. The reason why the developer did not manage to accomplish the implementation is that it was abandoned due to time out for completing it and that the developer did not wish to continue the experiment.

The implemented simple application was an Apex trigger that would invoke after the moment an Account record was inserted to the database. The trigger then should create ten records of Contact object related to the created Account. For every new inserted Contact record, the fields should be copied from an Account record. Those fields were: Address, Phone, Fax, AccountId, LastName, Description. This process required a simple loop that should create 10 Contact records. For Flow Builder solutions, it would cause confusion because the standard 'Loop' block could not be used. That is why a programmer had to build it similarly to the sample process exemplified in Figure 6.

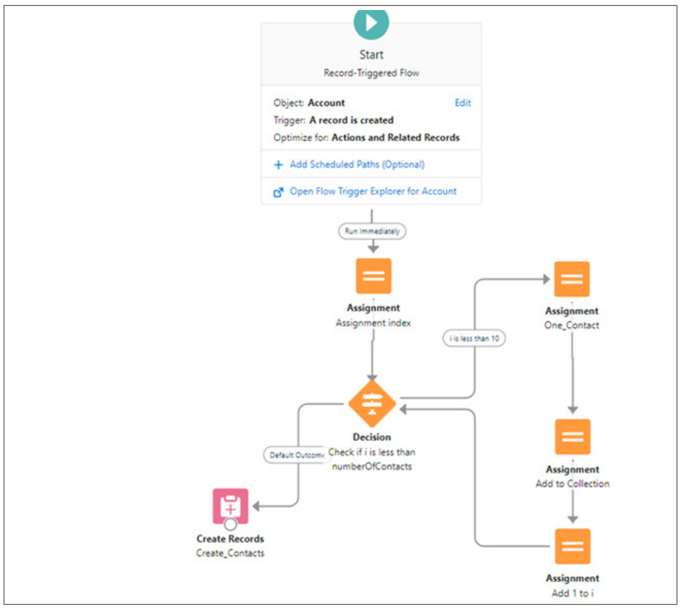

Start
Record-Triggered Flow
Object: **Account**
Trigger: **A record is created**
Optimize for: **Actions and Related Records**

Figure 6: The Sample Process Built With Flow Builder for the Simple Application

The latter application was a compound one regarding its implementation, but not necessarily time-consuming while executing. That application consisted of a few triggers. The call of the first trigger should occur after an Account record was inserted or updated. The trigger should select all the Opportunity records related to the updated Account record. If there is no Opportunity for an Account, the application should create one per Account. The developer should bulkify records in a list so that the application does not call DML operations of SOQL queries more than it is allowed according to Governor Limits.

After creating the first trigger, the developer was to create the second one for an Opportunity record (before inserting the trigger) (Figure 7). This trigger should add a new Pricebook2 object record and then relate Opportunity with that Pricebook. For the Pricebook2 object, there should be a Price Book Entry created and per every Price Book Entry record another Product2 object record as well. For each of these records, the developer was to input default values and the relationships between the records.

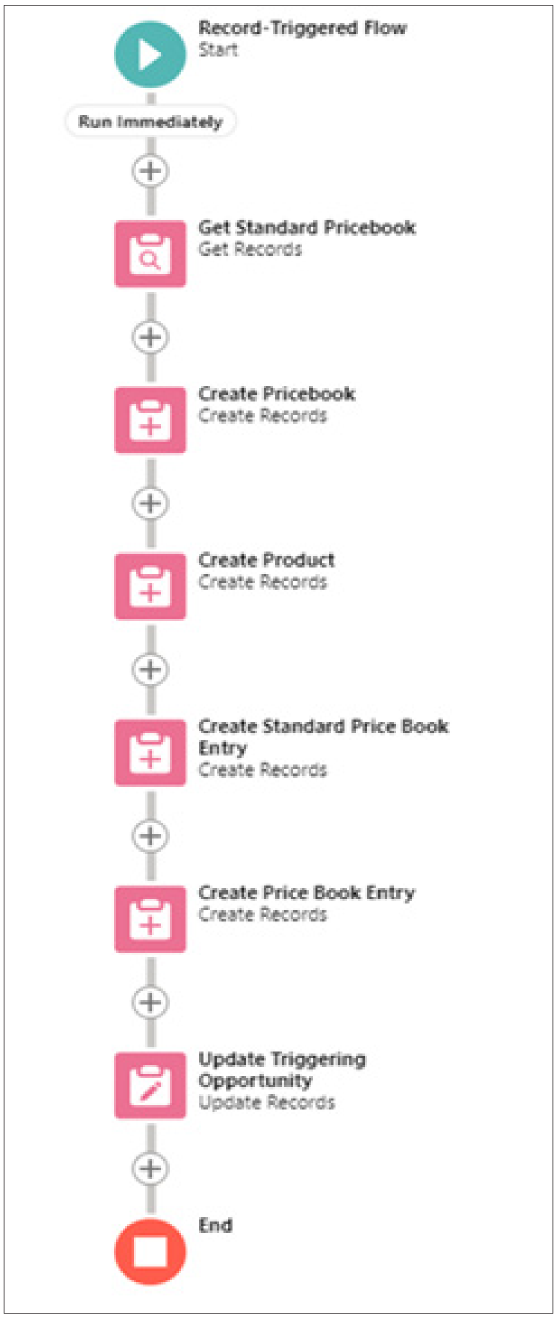


Figure 7: The sample process built with flow builder for the second trigger of the compound application. It is clear that there is no loop and the whole process is not complicated

The complexity of the apps was determined by the number of operations the process in the application should execute in traditional code-based programming (not the no-code solutions) and the number of objects used for the process. The simple application was to use only the Account and Contact objects (2 in number), and the compound application was defined with Account, Opportunity, Pricebook2, Price Book Entry and the Product2 objects (5 in number). For the simple application, the developer should create 10 Contact records per Account created with a few fields filled with values, and for the complex application, the developer should create different object records and collect the records in lists to use them in references in other records.

The Comparison of Code Execution Time in the Apex Language and Flow Builder Application

In order to verify the performance of the simple application, the Apex Anonymous Block was executed with 20 Account records for one series of executions and 200 Account records for the second series of executions. Those two types of transactions were executed 50 times. After each execution all the data was deleted, so the database would be empty. Because of the created trigger, with every of the 20 Account records there were also created 10 more Contact records, which caused overall 220 records created in one transaction. For 200 Account records, it was 2200 overall records created per one transaction.

Similar to the compound application, the identical process was performed. For every 20 Account records, there were 140 overall records created, and for 200 Account records, it was 1400 records. The research was conducted on EU42 Salesforce instance on 20 January 2023 afternoon.

The tool used for time execution logging was a Developer Console's Debug section. The EXECUTION_STARTED event indicated the start of execution time and the EXECUTION_FINISHED event indicated the end of execution time. The Debug Log shows the time of these events with 1ms precision.

Results of Research

The results of the research are presented as box-plots to show all the indicators of the collected data.

For the first part of the research, the charts were divided into the several box-plots depending on the seniority of the developers. The first two box-plots (Figure 8) present the results from the simple application implementation times of the developers performing the experiment in Apex and Flows, respectively. The second image (Figure 9) shows the compound application implementation times.

The results of the second part of the research present the application execution times, and the sample applications from Apex language and Flow Builder tool are compared. The simple application results (Figure 10) have two boxes for the 20 Account creation experiment and 200 Account creation experiment. Similarly, the compound application results (Figure 11) have the same concept as the simple application results.

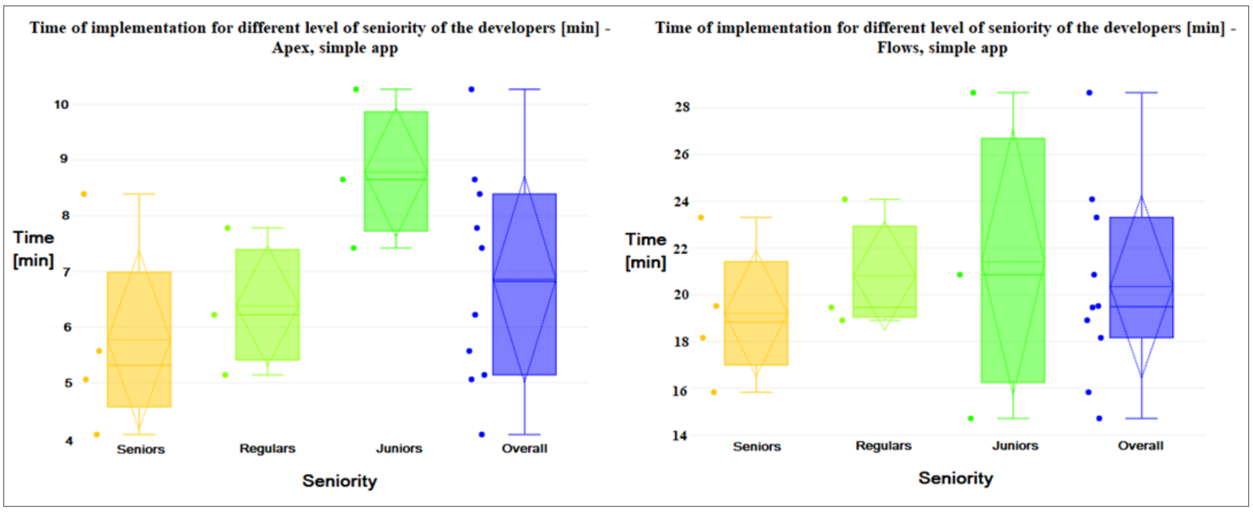


Figure 8: Box-plots for the Simple Application. Left-Apex Application Implementation. Right-Flow Builder Application Implementation.

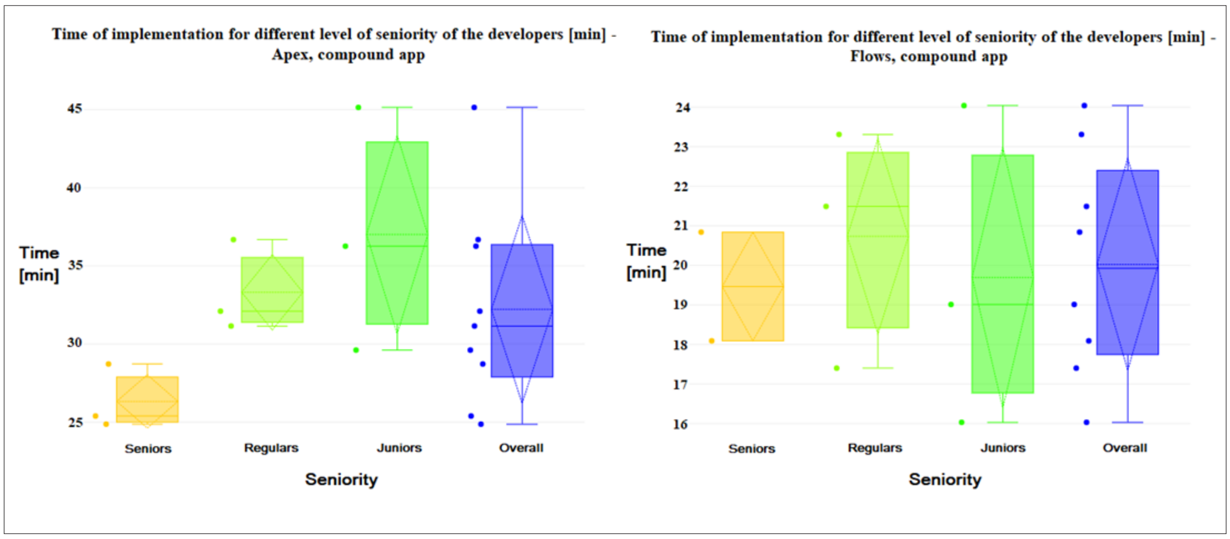


Figure 9: Box-Plots for the Compound Application. Left-Apex Application Implementation. Right -Flow Builder Application Implementation.

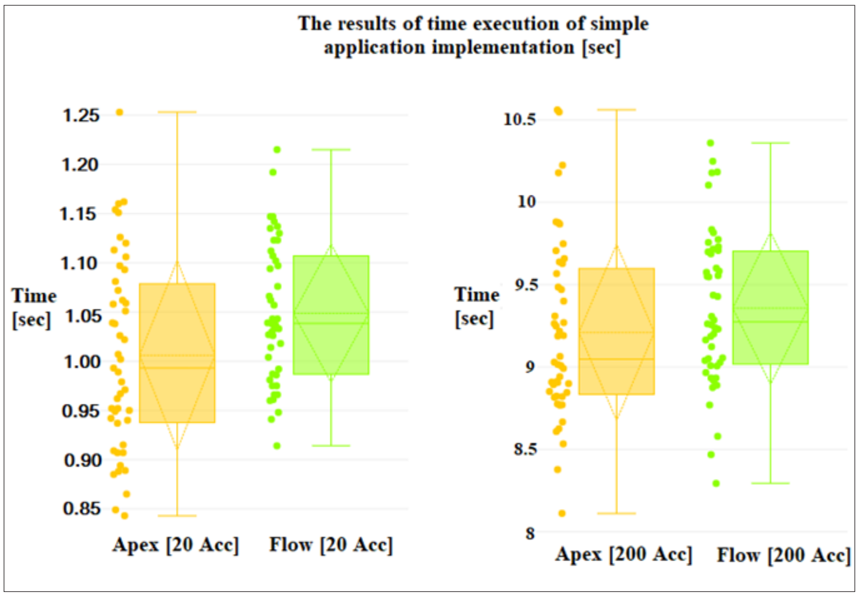


Figure 10: Box-plots for the Simple Application Execution Times Per Research. There were four series: Apex with 20 Accounts, Flow Builder with 20 Accounts, Apex with 200 Accounts, and Flow Builder with 200 Accounts.

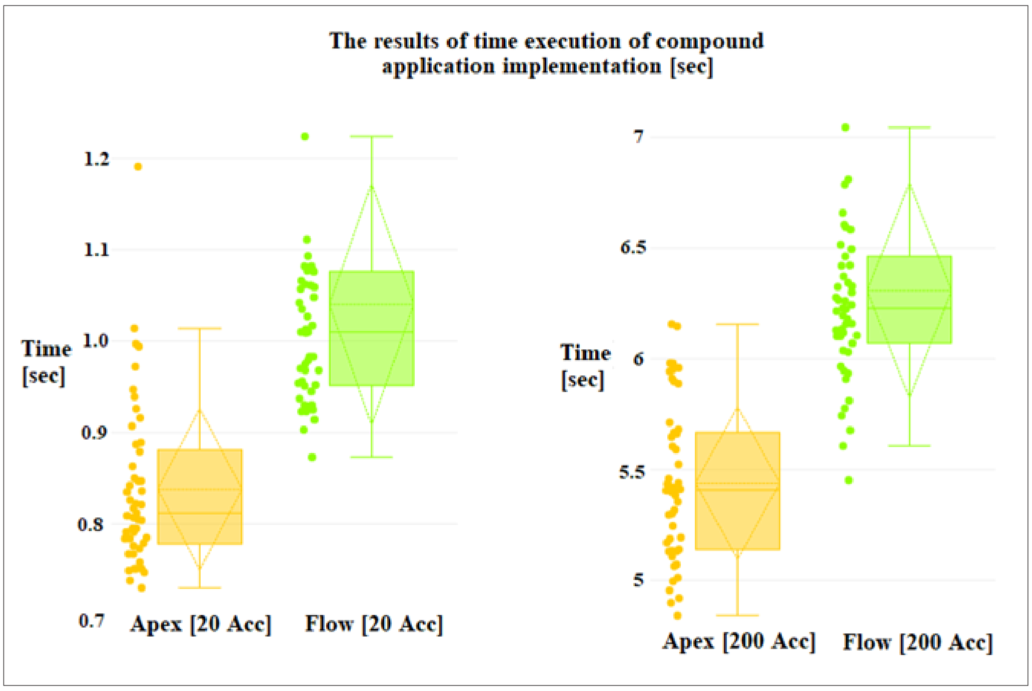


Figure 11: Box-plots for the Compound Application Execution Times Per Research. There were four series: Apex with 20 Accounts, Flow Builder with 20 Accounts, Apex with 200 Accounts, and Flow Builder with 200 Accounts

Discussion and Conclusions

The Comparison of Code Execution Time in the Apex Language and Flow Builder Application

The programming in Apex lasted longer for the least experienced programmers. For the simple application, the junior developers were implementing the app 2.4 min. on average slower than the regular developers and 2.7 min. slower than the senior ones. For the compound application, the junior developers were implementing the app 3.8 min. on average slower than the regular developers and 10.7 min. slower than the senior ones. The performance by the regular and senior developers was comparable, including the simple application, and it was much better for the senior developers, including the compound application (6.9 min. faster for the senior developers). The Flow solution implementing process was quite more diverse for all programmers, and no single group can be impartially indicated a winner, though averagely the senior group remains the quickest in both variants of the application, but for the compound application, the seniors are only slightly quicker than the other groups.

What might be surprising is that one of the junior developers was quicker than any other developer in implementing both variants of applications – this person reached times at most 16 minutes for the compound application and about 14.8 minutes for the simple application. It is, however, clear from Figures 8 and 9 that there was also one junior developer who was the slowest of all the developers and reached the time of over 28 minutes to implement the compound application and over 24 minutes to implement the simple application. That is why the average time of the junior developers on Flow Builder application is still higher than that of the senior ones.

Because of the low diversity of the simple application implementation, the time of the implementation in Apex language was way shorter than the implementation process in Flow Builder. On average, the time of the simple app implementation in Apex was 13.5 minutes shorter than in Flow Builder. The quickest attempt in Apex was 4.07 minutes and in Flow Builder it was 14.71 minutes.

The findings on the second compound application are different from the first one. It turns out that the mean time of the implementation in Apex has exceeded the time of implementation in Flow Builder by 12.18 minutes.

It may be a case that the implementation of the simple app in Flow Builder might have been more difficult because of a custom 'while' loop problem. In Apex, it is fairly easy to write a 'for' or 'while' loop. In Flow Builder, there is only a build-in 'for each' loop, so a developer needs to initialize variables in Toolbox's Manager, and this needs a lot more clicks than a usual code. This solution is also not very quick to be invented by a programmer. The quicker time of debugging for Apex might have impacted on the times of both implementations because the debugging tools for Flow Builder have not been developed much yet comparing to the Apex tools which have been developed since 2006, i.e. when the language was released, whereas Flow Builder was released in its current version in 2019. Still, graphical debugging and the process of opening operation blocks in the window in Flow Builder take longer than code investigation and placing system log lines to verify the values of the variables.

Bulkification of records was evident in the compound and not the simple application.. Regarding the specifics of the Salesforce platform, the DML operations should not be executed in a loop, so a developer has to collect records in a list and then execute a DML operation, which is time-consuming to implement if records are related and there are more complex requirements. If there is a need to work with many records which are correlated with other records, a developer needs to include all of other records in a list or a map to process many records in a single transaction.

This has to be done if a developer needs to use the data from these related records in the code. This process should be executed to avoid unnecessary queries to database to collect data for every single record that has a relationship to other record. That technique involves not exceeding the limits of the cloud in transaction. If there are any loops in the code containing a query to database, the technique of bulkification applies. It is used to cover all queries in a single query before the loop and save results in the heap memory.

This problem does not occur in Flow Builder because it is automated and bulkification occurs behind-the-scenes so the implementation of the compound solution in Flow Builder was much simpler than in Apex. What is more, there was no need to implement a facile 'while' loop in the compound application which turned out to be more difficult to implement in Flow Builder. From the perspective of the Salesforce developer, the first application was quicker to implement in Apex, but slower in Flow Builder, whereas the compound one was quicker to make in Flow Builder, but more complex in Apex.

The Comparison of Code Execution Time in the Apex Language and Flow Builder Application

To unambiguously show which methods and tools are able to provide better optimized applications, there was one more study needed. Both applications (simple and compound) had to be tested on the Salesforce platform environment to check debug logs for the execution times of these applications for both tools, namely Apex and Flow Builder.

The Apex-programmed simple application managed to create 20 Account records in about 1s on average. The analogic task was completed by Flow Builder a little bit later, i.e. in about 1.05s. The Apex-programmed simple application created 200 Account records in 9.2 s on average, whereas the Flow Builder solution needed almost 9.35s on average to complete this task.

The results for compound application were completely nowhere near the simple application outcome. The differences between the Apex-programmed solution and Flow Builder process were relevant to what had been expected. To create 20 Account records, the Apex solution needed an average time near to 0.85s, whereas the Flow Builder application was on average faster than 1s. Similar differences were recorded for the 200 Account records experiment. The Apex language completed the task in 5.4s on average, whereas the Flow Builder in about 6.3s.

While this difference is not so considerable for the simple application, there is a significant deviation in the compound application executions for the Apex and Flow Builder built solutions. It seems probable that it is automatic bulkification that occurs behind-the-scenes for Flow Builder due to the significant difference in time execution. Due to fact that in Apex the programmer must optimize the code and in no-code solutions it is a matter of imperfect process, the differences in execution time cannot be ignored. If the application consisted of hundreds of processes and automations, it would be difficult not to exceed the limit of time execution and the no-code solution would worsen that issue.

Our research shows that the compound application is much slower in execution when it is implemented in Flow Builder than the Apex coding although it takes longer in the development process to implement the solution coded in Apex than Flow Builder. It should also be remembered that the commercial systems are usually much more complex and the processes in them need to be executed as soon as possible to avoid delays. Having that in mind, it is recommended to use Apex programming for compound applications and processes rather than the no-code solution. On the other hand, there are simple applications whose execution time is comparable to the Apex programming method.

The implementation time of the simple application takes longer in Flow Builder than Apex, but the flows are much more readable for a non-technical person than the code. The scalability of such solutions is not very well if the developer had a task to expand the solution. For small processes, it is, however, more convenient to use Flows than traditional programming because of their clear user interface and the fact that a non-technical person could modify them. In simple applications, both solutions have their pros and cons so it is a developer or a product owner who can decide which method to use and why [16-18].

References

1. Marańda W, Poniszewska-Marańda A, Szymczyńska M (2022) Data Processing in Cloud Computing Model on the Example of Salesforce Cloud. Information 13.
2. Miłosz M, Michalczyk M, Wojdyga A (2014) Cloud Computing on the example of Salesforce, Problems of Contemporary Engineering. Programming technologies.
3. Kao L, Paz J (2016) Salesforce for Dummies. Hoboken, New Jersey, USA: John Wiley & Sons, Inc https://www.wiley.com/en-be/esforce+For+Dummies%2C+7th+Edition-p-9781119576327.
4. Developer Docs (2022) Salesforce https://developer.salesforce.com/docs#browse.
5. Levitt ER, Fry C, Greene S, Kaftan C (2011) Salesforce.com: The Development Dilemma. Collaboratory for Research on Global Projects http://stanford.edu/~robertk3/APM/HO%2000244%20Salesforce.pdf.
6. Weinmeister P (2015) Practical Salesforce.com Development Without Code: Customizing Salesforce on the Force.com Platform. Standard Libraries https://searchworks.stanford.edu/view/10805362.
7. Virta T (2018) Relation of low-code development to standard software development. Lappeenranta University of Technology https://lutpub.lut.fi/bitstream/handle/10024/158441/masters_thesis_virta_tatu.pdf?isAllowed=y&sequence=1.
8. Miącz D (2019) Performance analysis of methods for building applications on the Salesforce platform. Journal of Computer Science Institute 10: 24-27.
9. Patel S, Sharma S, Prasad R (2020) Multitenant Effective CRM Application Using Salesforce. Reserachgate https://www.researchgate.net/publication/338701344_MULTITENANT_EFFECTIVE_CRM_APPLICATION_USING_SALESFORCE?channel=doi&linkId=5e25ede792851c89c9b5990c&showFulltext=true.
10. Poniszewska-Marańda A, Matusiak R, Kryvinska N, Yasar A (2020) A real-time service system in the cloud. Journal of Ambient Intelligence and Humanized Computing 11.
11. Battisson P (2020) Learning Salesforce Development with Apex. BPB Publications https://bpbonline.com/products/learning-salesforce-development-with-apex.
12. Davis A (2019) Mastering Salesforce DevOps: A Practical Guide to Building Trust While Delivering Innovation. https://link.springer.com/book/10.1007/978-1-4842-5473-8.
13. Kermanchi A (2022) Developer Experience in Low-Code Versus Traditional Development Platforms-A Comparative Experiment. Aalto University https://aaltodoc.aalto.fi/server/api/core/bitstreams/18f9453c-5930-4a94-a545-4f9dc51be7aa/content.

14. Poniszewska-Marańda A, Matusiak R, Kryvinska N (2017) Use of Salesforce Platform for Building Real-Time Service Systems in Cloud. IEEE 14th International Conference on Services Computing https://ieeexplore.ieee.org/document/8035025.
15. White A (2021) 7 Things Architects Should Know About Flow. Medium https://medium.com/salesforce-architects/7-things-architects-should-know-about-flow-8173ddeeae28.
16. Help (2022) Salesforce https://help.salesforce.com/s/.
17. Salesforce Trailhead (2022) Salesforce https://trailhead.salesforce.com/.
18. Tamoń P (2021) Analysis of the Lightning Experience environment capabilities on the base of the CRM system implementation using the Salesforce cloud platform.