
**David Silvestre Daud**

Bachelor in Science and Computer Engineering

# Live Collaboration in App Development

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

Master of Science in
**Computer Science and Engineering**

Adviser: Nuno Preguiça, Associate Professor, NOVA University of Lisbon
Co-adviser: Luiz Santos, Software Engineer, OutSystems

Examination Committee:

| | |
|---|---|
| Chair: | Teresa Isabel Lopes Romão, Associate Professor, FCT-NOVA |
| Rapporteur: | João Carlos Pascoal de Faria, Associate Professor, FEUP |
| Adviser: | Nuno Manuel Ribeiro Preguiça, Associate Professor, FCT-NOVA |


FACULDADE DE
CIÊNCIAS E TECNOLOGIA
**UNIVERSIDADE NOVA** DE LISBOA

**November, 2021**

**Live Collaboration in App Development**

# Acknowledgements

I would first like to thank FCT-UNL and OutSystems for providing my with the opportunity of developing this dissertation and, as a result, be part of the innovation that takes place in these institutions.

Secondly, I want to thank my advisors, Nuno Preguiça and Luiz Santos, for the guidance and motivation throughout this dissertation. I would also like to extend my gratitude to everyone at FCT-UNL and OutSystems for their dedicated involvement and insightful suggestions.

Last but not least, I want to express my deepest gratitude to my parents and friends for their patience, support and encouraging words throughout this dissertation. Could not have done it without you.

# Abstract

Real-time collaborative editing systems are increasing in popularity, having moved from document editing software to the world of software development. Live collaboration (or real-time collaboration) refers to a synchronous cooperation mechanism, allowing for concurrent changes to be made to the same object by multiple individuals. In recent years, many traditional software development tools have started to incorporate live collaboration. The motivation behind this fast expansion comes from a series of specific use cases propelled even more by the current COVID-19 pandemic, which forces people to stay at home and work in a remote manner. This hinders the possibilities for cooperation between team members during the development of a software project. In this work, we address this problem in the context of the OutSystems low-code platform, and we aim to determine how collaborative features, including real-time collaboration, can be implemented in OutSystems tools to enhance its collaborative experience for users developing applications. In this context, collaboration is defined as the processes and actions that take place between people during software development projects with the OutSystems platform, when trying to execute their work tasks.

To test the ideal experience for collaborative features, such as real-time collaboration, in the OutSystems ecosystem, we analyzed the current state of the art of the research done in the fields of CSCW (Computer-supported Cooperative Work) and UX (User Experience) and experimented with other industry standard software to analyze their collaborative features. Because features are made for people, we then moved to the end-users and interviewed several users of the OutSystems platform to understand their issues when cooperating with other people, and finally we generated a series of designs to try to address their issues. These designs were conceptualized and materialized into actual mockups that were part of several usability tests, done with OutSystems users, to realize their potential in enhancing the collaboration experience when using OutSystems.

**Keywords:** Real-time collaboration, live collaboration, low-code platform, computer-supported collaborative work, user experience, groupware, awareness

# Resumo

Os sistemas de edição colaborativa em tempo-real estão a ganhar popularidade, tendo transitado do mundo do *software* da edição de documentos para o mundo do desenvolvimento de software. O título do documento, "Live collaboration in app development", refere-se a mecanismos de colaboração síncrona, que permitem alterações concorrentes a um mesmo objecto por parte de vários intervenientes. Nos últimos anos, várias ferramentas tradicionais de desenvolvimento de *software* começaram a incorporar colaboração em tempo-real. Esta rápida expansão é motivada por vários casos de uso, que ganham uma maior relevância na atualidade devido à pandemia da COVID-19, em que muitas pessoas têm de trabalhar de forma remota a partir de casa. Esta situação dificulta as possibilidades de cooperação entre colegas de equipa num projeto de desenvolvimento de software. Este trabalho aborda estes problemas no contexto da plataforma de *low-code* da OutSystems e pretende-se determinar se e como certas funcionalidades colaborativas, como colaboração em tempo-real, podem ser implementadas na plataforma da OutSystems de forma a melhorar a experiência colaborativa para os seus utilizadores. Neste contexto, colaboração refere-se aos processos e ações que ocorrem entre as pessoas durante os projetos de desenvolvimento de software com a plataforma da OutSystems.

Para definir e testar a experiência ideal destas funcionalidades colaborativas, como colaboração em tempo-real, no ecossistema OutSystems, analisámos o estado da arte atual da investigação feita nas áreas de CSCW (*Computer-supported Cooperative Work*) e UX (*User Experience*) e experimentámos outros programas disponíveis no mercado para analisar as suas funcionalidades colaborativas. Como o software é feito para pessoas, entrevistámos vários utilizadores da plataforma OutSystems para compreender os seus problemas ao cooperarem com outras pessoas no contexto OutSystems e conceptualizámos várias ideias para tentar resolver esses problemas. Essas ideias foram depois materializadas em protótipos reais que fizeram parte de vários testes de usabilidade para perceber o seu potencial em melhorar a experiência de colaboração em OutSystems.

**Palavras-chave:** Colaboração em tempo-real, plataforma low-code, experiência de utilizador, software colaborativo, desenvolvimento colaborativo de software, colaboração

# Contents

# List of Figures

# List of Tables

# Glossary

| | | |
|---|---|---|
| **component** | Not a definition from the OutSystems Language and defined here to increase this document's readability, a component refers to the parts of an element (e.g., when applied to a screen, component refers to the web blocks or widgets that comprise it; applied to an action, it refers to the nodes and connections between them that together make the Action) | 8 |
| **conflicts** | In Version Control Systems, a conflict occurs when different individuals make changes to the same object (e.g., a document) and the system is unable to reconcile the changes (cannot decide which change should be kept) | 7, 10 |
| **element** | According to the OutSystems Language, an element within the OuySystems environment refers to the parts that together form the application (e.g., a Screen, Action, Aggregate, etc.) | 8 |
| **environments** | An abstraction of the stages of an application, to control what developers and end-users see and have access to. Environments can have different purposes but are generally useful for compartmentalizing different versions of an application, like a version with new features which are not yet ready for release | 6 |
| **Likert Scale** | A response scale in which responders specify their level of agreement with a statement typically in five points: (1) Strongly disagree; (2) Disagree; (3) Neither agree nor disagree; (4) Agree; (5) Strongly agree. | 30, 53 |

| | | |
|---|---|---|
| **merge** | A way of incorporating the changes simultaneously made to a file from two different sources | 2, 7, 10 |
| **mockup** | A scale or full-size model of a design or device, used for teaching, demonstration, design evaluation, promotion, and other purposes | 29, 48 |
| **publish** | In OutSystems, publishing an application transfers the changes made locally to the OutSystems Platform repository, so they are saved, compiled, and can be executed or tested by anyone with access | 7 |
| **real-time collaboration** | Technology/software that enables multiple users in different locations, to simultaneously work together on a common task | 1 |

# Acronyms

| | | |
|---|---|---|
| **CSCW** | Computer-Supported Cooperative Work | 1, 18, 25 |
| **HE** | Heuristic Evaluation | 28 |
| **IDE** | Integrated Development Environment | 2, 6 |
| **IT** | Information Technology | 1, 5, 30 |
| **LCAP** | Low-Code Application Platform | 5 |
| **MVP** | Most Valuable Professional | 37 |
| **POC** | Proof-of-Concept | 15 |
| **QA** | Quality Assurance | 6 |
| **TTV** | Time To Value | 41, 48 |
| **UI** | User Interface | 6, 8, 50, 57 |
| **UX** | User Experience | 26, 30, 42, 50, 52 |
| **VCS** | Version Control System | 22, 23, 24, 68 |

Chapter 1

# Introduction

This chapter serves the purpose of contextualizing the topic of Live Collaboration in the current domain of information technology. It also presents the objectives and contributions of this work to the state of the art as well as the chapter organization of this document.

## 1.1 Context

When working on a project involving different people who need to work as a team, collaboration between them can enhance productivity and generate better results. This has benefits in improving the time taken to reach that goal, but also on the quality of the work being done. That motivates the Information Technology (IT) industry to further develop ways of people being able to collaborate with each other, since collaborative work enables people to meet the demands of a "rapidly transforming and increasingly competitive business environment" [1]. The COVID-19 pandemic has created several challenges in the way collaboration is conducted, as many individuals now have to work remotely, having shifted from an office space to their homes [2]. Because many people now cannot work in a shared physical space, live collaboration becomes more relevant since it can help overcome the challenges of this new and isolated work environment.

Live collaboration (or real-time collaboration) happens in a shared workspace environment between concurrent users, who could be in different geographies. The main idea is to resemble the interactions people have when doing a team activity in the same physical space (e.g., collaboratively building a puzzle on a desk). This way, team members have a place where they can observe each other's movements and the changes they are doing to a common object, accessible to all the people involved. This has been a part of the research landscape of Computer-Supported Collaborative Work (CSCW) for quite some time, as

evidenced by decades of research work resulting in many studies that reveal its benefits and disadvantages [3–5]. Real-time collaboration can represent different experiences and purposes, since collaborative work among people can include many types of work, like project management, document editing, and any kind of tasks that require cooperation (e.g., building a puzzle).

In software engineering, real-time collaboration has increased in popularity in recent years, which could potentially be an alternative to more traditional ways of collaboration like version control systems (e.g, Apache Subversion, Git), where changes to a project are manually published and "pulled". These systems facilitate collaboration among developers but not without its pitfalls. To address the disadvantages of these systems and give an alternative collaborative experience, some software development tools have incorporated real-time collaboration features [6]. It provides a different or complementary way for development team members to collaborate with each other on a project.

Looking at the reality of software projects developed with OutSystems, cooperation currently happens with the aid of version control systems which rely on the merge operation (see Subsection 2.4.1). Analyzing telemetry data from the OutSystems Platform, the merge operation is the 11th most time-consuming activity in Service Studio (the OutSystems Integrated Development Environment IDE for developing applications), accounting for about 2% of total development time[1]. Although it might not seem much, this does not account for the time indirectly spent dealing with the merge operation (e.g., talking with a colleague to decide on what gets merged), and all this is time consumed with things not directly related to the project itself, which do not bring value to the company or the end users. This can also make a case for finding other ways to collaborate that do not involve the merge operation.

## 1.2 Motivation

Real-time collaboration changes the way cooperation can happen at the core of software development teams. It can potentially increase the efficiency of collaborative processes, accelerating product development and facilitating cooperation in specific use cases. Currently, the way cooperation happens in OutSystems involves the merge operation, which can sometimes be time-consuming, making it interesting to find an alternative to this way of collaborating. It is clear, then, that an analysis of the collaborative processes that currently exist in the OutSystems ecosystem can make a case for introducing real-time collaboration features in OutSystems' platform and tools. Keeping up with this industry trend could also place OutSystems at the forefront of an innovative collaboration experience, as real-time collaboration is an unexplored topic among low-code application platforms.

[1] Telemetry data collected from January 1st 2021 - August 1st 2021

Recently, the topic of collaboration has become more relevant due to the COVID-19 pandemic, with several studies analyzing the impact on communication and productivity of development teams [2, 7, 8]. Restrictions imposed by several countries have forced many people to stay at home. Some companies have then switched to a remote work model where people no longer can collaborate in the same physical space (e.g., an office) and so have to communicate virtually with their colleagues. Some companies question the quality and productivity of its employees when working from home [9]. It is then more important than ever to have powerful and useful collaborative tools able to counteract the inherent struggles of remote collaborative work, making an even stronger case for the motivation of this thesis' work.

## 1.3 Objectives

The goal of this thesis' work is to understand what and how to improve collaboration among developers using OutSystems' platform, with an emphasis on live collaboration. The focus was not on implementing collaboration features into the OutSystems ecosystem but instead to realize what kind of user experience and design can maximize the benefits of collaboration for development teams, to facilitate and accelerate their everyday work and consequently allowing them to deliver a better value to the business and its end-users.

### 1.3.1 Contributions

An initial contribution is a study that effectively describes the collaborative processes that take place in the OutSystems ecosystem between development teams and its members, as well as the difficulties they face that can harm a project's quality or time requirements.

The main contribution of this thesis is the definition of the strategies that can be applied by OutSystems in Service Studio and its platform to improve collaboration between users. This can enhance cooperation and thus increase the quality of the work produced and accelerate application development. Not only is this useful for OutSystems, but also to the scientific community to understand the impact of collaborative features in low-code application platforms.

## 1.4 Document Structure

This first chapter introduced the topic of Live Collaboration in App Development. The next chapters are organized as follows:

- Chapter 2 - "OutSystems": Introduction to the OutSystems platform and tools.
- Chapter 3 - "State of the Art": An analysis on collaboration across four different levels: how it happens in the OutSystems ecosystem; core concepts of collaborative

software, as described in the scientific literature; commercially available software examples; core concepts of user experience and usability;

- Chapter 4 - "Methodology": An explanation of the methods chosen to guide the work of this thesis as well as the exposition of some initial results that lead to the verdict of this dissertation;
- Chapter 5 - "Results": Exposition of the prototypes used during testing and the feedback received;
- Chapter 6 - "Conclusion": Discussion of the results obtained in testing, and respective conclusions, followed by the topics that can be further studied in the future.

# Chapter 2

# OutSystems

This thesis is being done in the context of software development using the OutSystems Platform. Therefore, this chapter covers the paradigm of low-code development and the main components of the OutSystems Platform.

## 2.1 Low-Code Development

Low-code development is a software development approach that aims to reduce the time and effort required to create new applications. Developing with minimal hand-coding makes it possible to not only dramatically accelerate application development and deployment, but it also allows people without a strong background in IT to take part in the process of developing applications [10, 11]. This development approach is enabled by Low-Code Application Platforms (LCAPs), such as OutSystems, enabling developers to develop and deliver complete applications in a visual manner. According to Gartner [12],

> "An LCAP is an application platform that supports rapid application development, deployment, execution, and management using declarative, high-level programming abstractions such as model-driven and metadata-based programming languages, and one-step deployments. LCAPs provide and support user interfaces (UIs), business processes and data services."

## 2.2 OutSystems Platform

OutSystems is a low-code application platform built to accelerate the delivery of mobile and web applications. It covers the development, quality assurance, deployment, and management stages that comprise an application's development life cycle[13].

OutSystems ecosystem of tools provides several features for interacting with the OutSystems **Platform Server** to develop and manage applications (Figure 2.1). In the development stages, OutSystems provides two tools for creating the different components of an application[14]:

- **Service Studio** - The OutSystems IDE where developers can create the different layers of an application like the User Interface UI, data model, and the business and application logic. Each layer can be designed with the drag and drop of visual elements, which can then be edited and connected between each other;
- **Integration Studio** - A development environment that interoperates with Microsoft's Visual Studio[15] where components can be hand-coded in the C# programming language[16]. These components extend the OutSystems Platform, allowing to integrate it with third-party systems and external databases. These components can become available to anyone in the OutSystems community, allowing them to be reused by any other application.

For management of the entire development, quality assurance, and deployment process, two other tools come into place: **Lifetime** and **Service Center**. These allow for management and configuration of all environments (which typically can be of three types: Development, Quality Assurance (QA) and Production), as well as the infrastructure, security and access control of the OutSystems Platform Server [17].

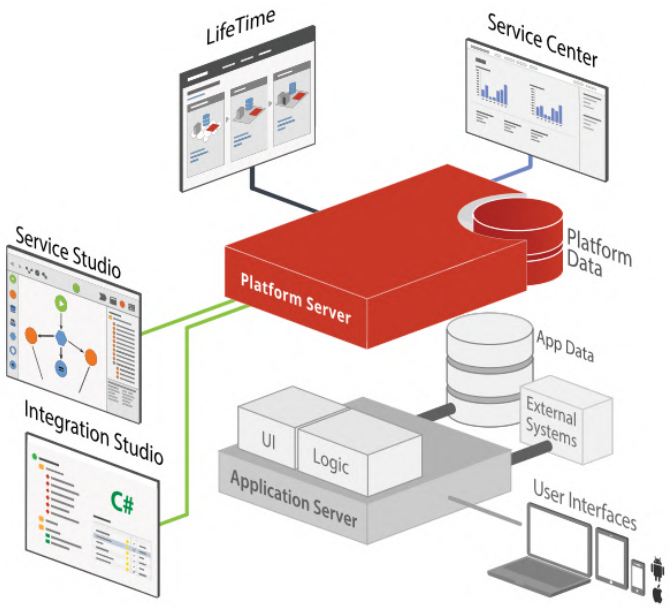



Figure 2.1: Architecture of the OutSystems Platform[18]

Once an application is published to the Platform Server through Service Studio, it will compile and generate optimized code for that application and deploy it to an **Application Server** that uses traditional external databases and systems. The Application Server

is then where end-users access the applications developed with OutSystems on their personal devices.

A crucial component to the OutSystems Platform is the **Platform Server**. It is responsible for generating, building, packaging, and deploying the applications. It is also the server to which developers commit the changes made to the application. Regarding the design and building stages of an application, the Platform Server offers two important services[19]:

- **Code Generator** - It analyzes the application's model and generates native code for all components that were developed with visual elements in Service Studio, also checking for optimizations and managing the necessary external dependencies;

- **Version Control** - When developers are editing the same application module at the same time and publish their version, the Platform Server can automatically identify which conflicts exist between both versions and informs the developer what are the differences between the two. This way, the developer has all the information to merge and publish its work. It also has access to a history of all published changes of an application, with the possibility to roll back to a previous version.

## 2.3 Service Studio

Service Studio is OutSystems' low-code visual development environment where developers can design, debug, and deploy different modules of an application[20]. Its workspace comprises different sections as illustrated by Figure 2.2.

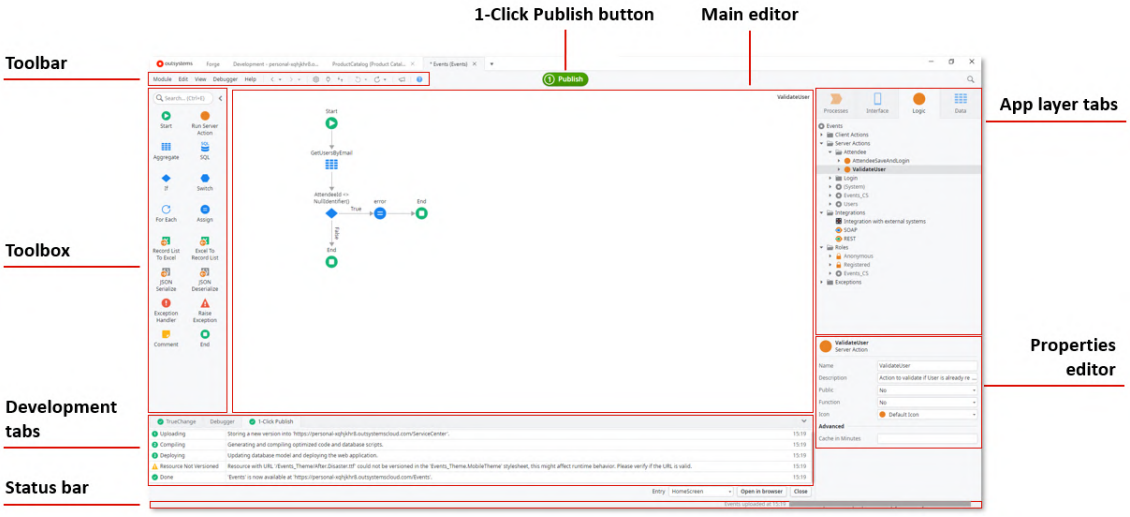



Figure 2.2: Different sections of the Service Studio workspace

### 2.3.1 Workspace

The Main Editor is where the UI, Logic and Data layers of the application are created and edited, where users can drop, for example, components and widgets from the Toolbox area or entities from the data tab in the App Layer Tabs. The properties of the visual elements in the Main Editor can be edited in the Properties Editor. The 1-Click Publish Button is responsible for uploading the application model to the Platform Server, which in turn will compile and deploy it.

The different layers of the application are distributed among tabs in the App Layer Tabs section:

- **Processes** - where business processes can be created;
- **Interface** - definition of the application's UI through a collection of **Screens**. Each Screen has its visual representation (see Figure 2.3) which is represented logically by a hierarchical tree composed of web blocks and widgets;
- **Logic** - definition of the application's logic through a model called **Action**. Actions are flow diagrams that define a sequence of steps (represented as nodes) to represent the execution of a certain process of the application, as illustrated by Figure 2.4;
- **Data** - Here is where the database model for the application is declared with the composition of different **Entities**. Entities correspond to database tables or views and queries can also be made to the database using **Aggregates**.

To make it easier to read and understand, we have defined two conventions for the language of this document:

- The term element is part of the OutSystems Language, and it applies to the parts of the application such as Screens, Actions and Aggregates; along this document, it is also sometimes referred to as "module element";
- The term component has been defined for this document to define the parts of an OutSystems element (defined above). For example, a component, when applied to a screen, refers to the web blocks or widgets that comprise it; applied to an action, it refers to the nodes and connections between them that together make the Action.

### 2.3.2 Modules

In OutSystems, the design of an application's structure typically follows a modular architecture [21]. By having the functionality of the application separated into different independent modules, multiple developers and teams can work in different areas of the application without conflicting with each other's work.

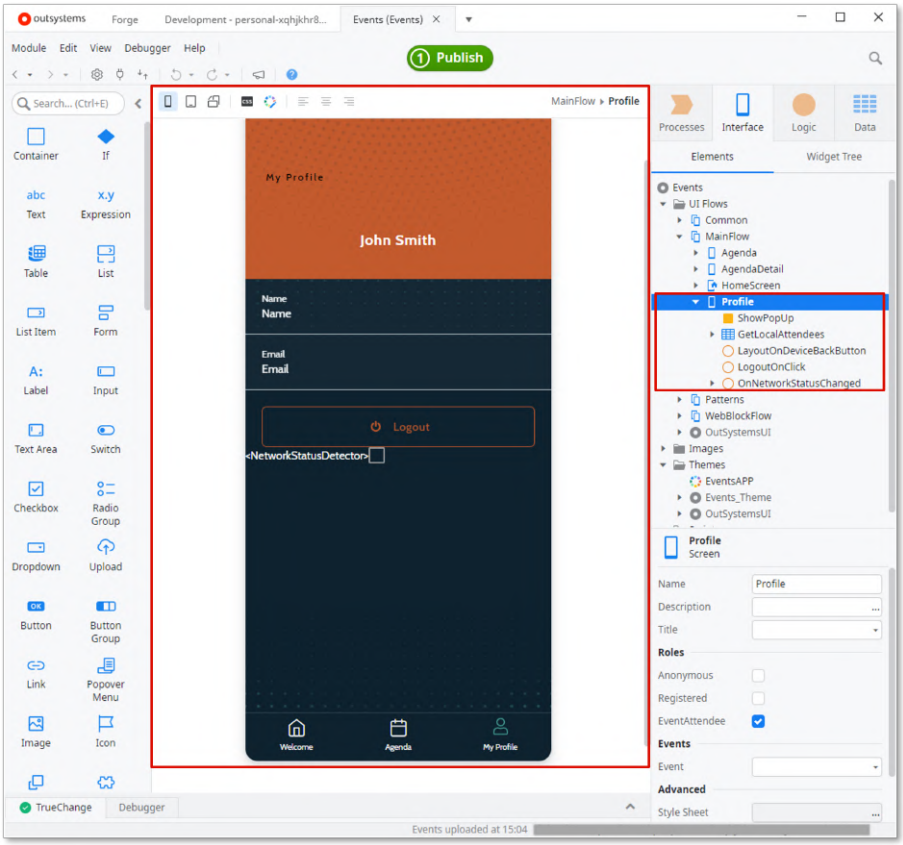



Figure 2.3: Example of an Interface screen

In Figure 2.2, Service Studio's workspace represents a specific module of an application. The App Layer Tabs organize the different layers of that module (described in Subsection 2.3.1) and each layer comprises different components. These components are hierarchically laid out to give the developer a global view of each layer of the module in a tree structure.

### 2.3.3 TrueChange

TrueChange is an important service of Service Studio, as it automates the work of the developers in several ways. It performs bug checking and impact analysis in Service Studio, helping in achieving no errors or inconsistencies in the application's different components. It can also aid a developer's work by recommending, through an AI-fueled analysis, solutions for correcting some of those errors.

## 2.4 Publishing Changes

Developers during the development process of an application have to eventually publish their work. When they first open Service Studio, it fetches the latest published version of the application so they can start their work. However, changes made to different

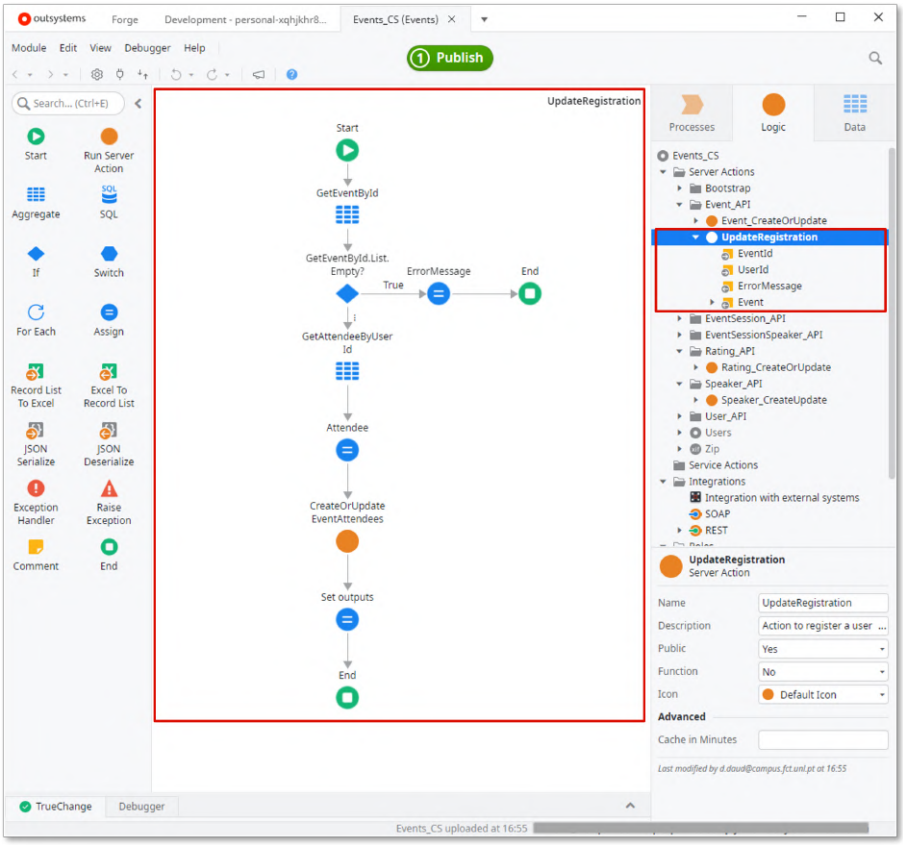



Figure 2.4: Example of a Logic server action

components only happen locally, so these changes need to be regularly published to the OutSystems Platform Server, which acts as a central repository for the application [21], with features such as version control and code integration (merge). After the changes have been sent to the server, it will generate native and optimized application code and then deploy it to the Application Server, becoming immediately available for users or testing. An important feature of modules is that each one has a single development server which acts as a single execution and testing environment, shared between all developers working on that module.

OutSystems Platform Server acts as a central development server, implementing a version control system, where all applications and modules are automatically stored and labeled after each successful publication. This feature enables access to an application's history, including the date and authorship of all changes, and the possibility to roll back the application to an earlier version.

### 2.4.1 Merge Operation

As previously mentioned, applications in OutSystems can have several independent modules. An application is separated into reusable logical parts, facilitating maintenance and avoiding conflicts between developers by having them work in distinct modules in

parallel. As a result, it is possible to have multiple developers working on an application simultaneously, but each in different modules of the application, to avoid conflicting changes to the same module [22].

Even with an application split in independent modules, sometimes it is inevitable to have several developers working in the same module. In these situations, if several developers make changes to the same module at the same time, their separate versions have to be merged. The merge operation can reconcile the multiple changes made to the module by different developers to a single published version. It can be done automatically in certain situations, requiring no intervention from the developers. However, sometimes the platform cannot automatically merge the changes. This would mean that there are conflicting changes to the module being published which require the merge operation to be done manually by the developer publishing the more recent work.

As described in the OutSystems documentation [22], Figure 2.5 represents an example of a conflicting merge situation:

1. Developer A opens version 4 (the most recent version) of the application's module in their local instance of Service Studio and starts working;
2. Developer B opens the same version 4 of the same application's module in their local instance of Service Studio and starts working;
3. Eventually, developer A publishes the changes they made to the module creating a version 5 in the Platform Server;
4. Developer B tries to publish their changes and a conflict is detected.

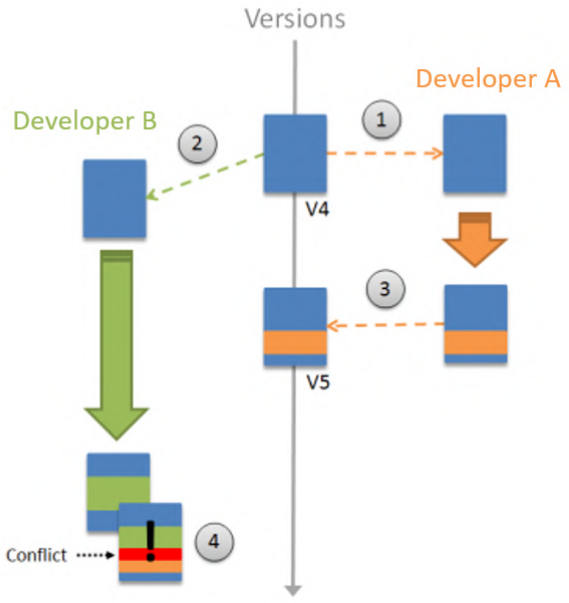



Figure 2.5: Timeline representation of a merge conflict (adapted from [22])

A conflict arises because the OutSystems Platform Server and Service Studio detect that both developers started their work from the same version of the module (Version 4) and at least one of the components that Developer B has altered was also changed and published by Developer A who created Version 5. Thus, the platform cannot automatically decide which changes should be kept, and a window is displayed to Developer B to decide what gets merged and published in the final version.

Essentially, when a user clicks the 1-Click Publish Button to publish their changes and the Platform Server detects that multiple users are publishing different versions which have diverged from a common base, it takes a series of reconciliation steps. It starts by identifying differences between versions to detect possible conflicts; if no conflicts were detected, it merges automatically; otherwise, the user is prompted to manually resolve the conflict, deciding which changes should be kept. Finally, a new merged version of the module is created in the Platform Server.

Figure 2.6 shows the Modified Version Detected window, which appears when a developer is trying to publish a module and there are conflicts with the new version published by another developer.

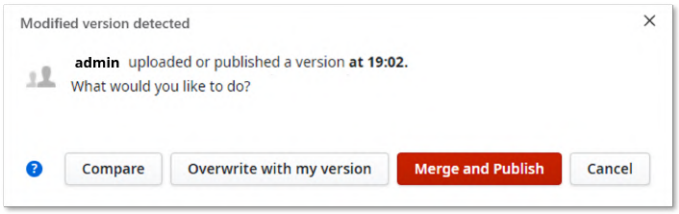
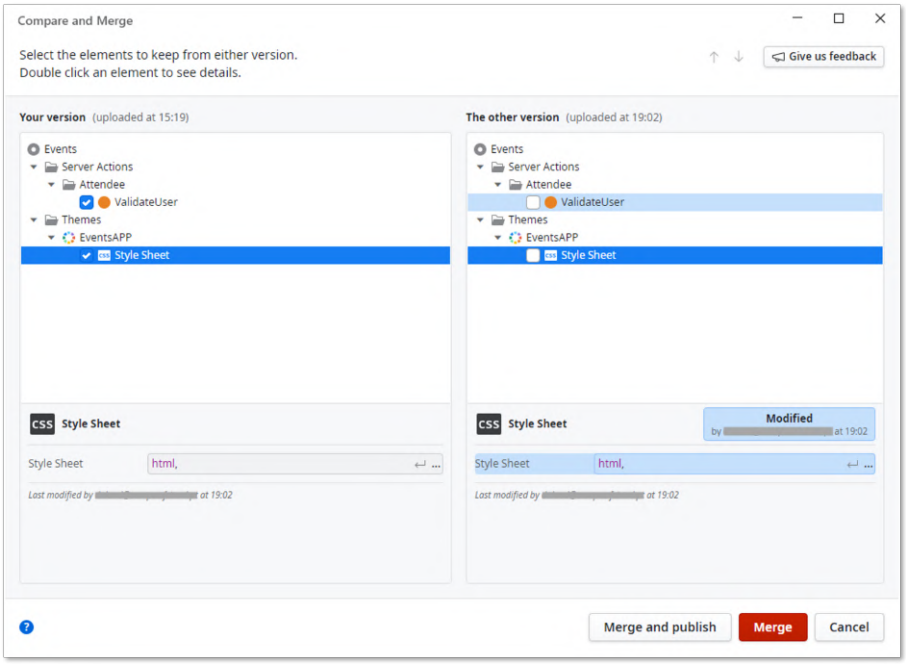


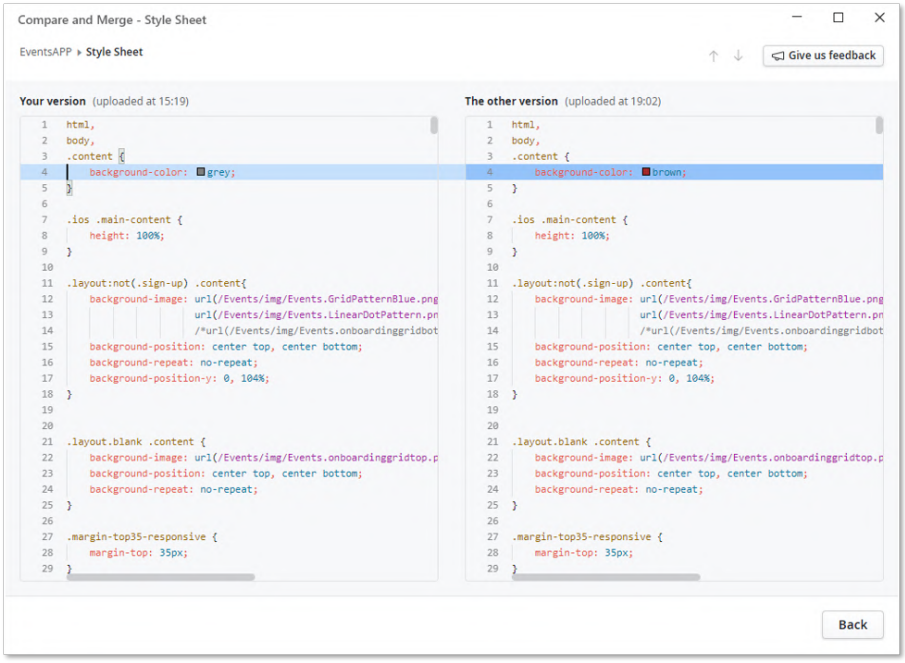

Figure 2.6: Modified Version Detected window

Following the previous example of Developer A and Developer B, the Modified Version Detected window would appear to Developer B at the end of step 4. This window gives three different options to the developer[23]:

- **Publish my version** - The server version is overwritten by the local version that the developer is trying to publish. This is a destructive option since all changes from Developer A will be lost;
- **Merge and publish** - The server attempts to automatically merge the server version with the new local version of the module. If that is not possible, the developer has to manually resolve existing conflicts in the Compare and Merge window (see Figure 2.7);
- **Compare** - The Compare and Merge window opens and the developer can compare, side-by-side, both versions and decide on how to resolve any existing conflicts. For instance, in Figure 2.7 there is a conflict because both versions conflict on the same color attribute for a CSS selector.




(a) Main window




(b) Conflict resolution in a style sheet

Figure 2.7: Compare and Merge window examples

# Chapter 3

# State of the Art

Live Collaboration is a broad informal topic, as it could refer to different solutions designed to improve different aspects of collaboration. In this thesis, we are interested in solving collaboration issues that can happen during application development in OutSystems' low-code platform. Therefore, this chapter first depicts how collaboration between developers takes place in the OutSystems ecosystem. The following sections present an analysis of the industry's work on tools commercially available to users that possess a functionality in the solution space of live collaboration. Finally, some concepts described in the scientific literature for collaborative software, user experience and usability, relevant for our research, are explained.

## 3.1 Collaboration in OutSystems

We have conducted several interviews with development teams to understand their internal structure when developing projects in OutSystems and how they collaborate with each other and with other development teams. This section will cover some of those findings, complemented with information from the OutSystems documentation and a Master's thesis conducted at OutSystems in 2013, that researched about the same topic [24].

### 3.1.1 Previous Work

A thesis from 2013 entitled "Real-Time Collaborative Editing of OutSystems DSL Models" [24] explored the possibility of a real-time collaboration feature in OutSystems' Service Studio software and implemented a prototype, serving as a Proof-of-Concept (POC). This previous work has served as an assist for the research work conducted throughout this thesis. However, the OutSystems platform, tools and work methods have evolved and

changed, making it significantly different, at times, from what they were in 2013. Furthermore, implementing real-time collaboration in OutSystems falls outside the scope of this thesis, which has sought instead to conceptualize its design and user experience.

Although some insights of the 2013 thesis are now inaccurate, some of them can and were still used to help understand the collaborative ecosystem of OutSystems. Moreover, it is necessary to point out that this document represents a whole new work on its own for researching the topic of real-time collaboration in OutSystems and so it is not a follow-up to that previous thesis.

### 3.1.2 Agile Methodology

Agile methodology is the basis for delivering projects with OutSystems. It has several benefits when compared with older techniques, such as waterfall methods, and "it relies on breaking work into short durations, close team collaboration, continuous learning, and frequent feedback on completed work" [25]. Applying the Agile methodology, there is an established foundation common to all development teams which comprises six areas: business, demand and governance, development, architecture, user experience, and operations (Figure 3.1). Teams usually have at least one person representing the Business area (Business Sponsor), one person for Demand and Governance (Product Owner / Engagement Manager) and three people for the Development (two Developers and one Tech Lead). Bigger projects could have more members in each of these areas or in other areas (e.g., a security expert could be required for developing a banking application). For our research, we are only focused on further exploring collaboration within the Development area in a team's ecosystem.


Figure 3.1: Basic foundation team delivering projects with OutSystems [26]

### 3.1.3 Tech Lead

The Tech Lead is a role inside a team's Development area. It comes with responsibilities that differentiate it from other roles related to development. Before the development team implements a solution, the Tech Lead is responsible for designing a scalable and high-performance architecture of the product and to drill down the project's requirements to specific well-defined tasks, which will be assigned to developers (these tasks in OutSystems are often referred to as **User Stories**). During this process, a Tech Lead is focused on the business goals and end-user needs, so the final solution can bring value to both [27].

Once development begins, the Tech Lead has more responsibilities. They lead the development team through the project, keeping them focused on following deadlines and best practices while ensuring that high-quality solutions, compliant with the planned architecture, are being delivered to production.

### 3.1.4 Developer

The Developer role is also part of a team's Development area. As previously mentioned, two Developers and one Tech Lead is the foundation for the members of a team's Development. The Developer implements the technical solution and comprehensively tests it, keeping it consistent with the architecture defined by the Tech Lead. They can also assist the Tech Lead at the start of the project in drilling-down requirements and design the architecture of the solution. However, their main work comes once the technical implementation of the solution begins. Developers should complete the tasks assigned to them, making sure they comply with the established architecture and follow best practices to create a scalable, high-performance deliverable [28].

According to a previous Master's thesis done about collaboration in OutSystems [24] and the insights received during the interviews with development teams, these are the steps taken by developers when completing a task:

- **Contextualization** - A developer receives a task and a deadline to complete it. First, it needs to contextualize with the task by opening Service Studio, analyzing any actions/entities/interfaces or any other components of the module they will be working with. Sometimes, it is necessary to run and/or debug the latest version of the application. During this phase, developers collaborate with each other by discussing ideas and giving assistance to clarify some aspects of the project;
- **Development** - Being aware of the current state of the application and the task they must complete, the developer can now begin to implement changes to the required module. These changes only affect the local copy of the module, meaning that the latest published version in the development server is unaffected. Once the developer feels confident about his implementation and/or wants to proceed with testing it, they can go to the next step, as long as Service Studio is not displaying any error

messages (warnings should be minimized but are not an impediment). During development, there is cooperation between developers to assist in any implementation details;

- **Testing** - To test the implementation, the developer needs to publish his changes to the central development server (the OutSystems Platform Server) which will automatically upload, compile and deploy the new version of the application. To do so, the developers only need to press the 1-Click Publish button (see Figure 2.2) and the OutSystems Platform automates the process. Once it is done, the deployed application can be executed in a browser window (or mobile phone, if it is a mobile application), where the developer can test the functionality of the changes they incorporated to the application. Since publishing to the development server can cause conflicts (see Section 2.4), developers reported that in those situations, they usually communicate with their team members before and during publishing to understand what should be merged and avoid losing important changes by accident;

- **Closure** - After testing, the developer might need to go back a step and continuously loop between the development and testing steps, until they feel confident that their solution is of high-quality and meets the task's requirements. Once that happens, no further action is required since all changes are published to the development server when testing the application. A Tech Lead will eventually review the developer's solution and assess if it meets the expected specifications.

While completing a task, changes made to the module are published to the OutSystems Platform Server, which acts as a central repository server, keeping a version history of all applications and modules published. This process is described in Chapter 2. This approach, alongside the fact that solutions developed with OutSystems need to have a highly modular architecture (so developers can work separately in their respective module and avoid conflicts), makes collaboration very important during development.

In Section 4.2, the reality of collaboration in the OutSystems ecosystem is further described and analyzed, based on the results of the conducted interviews and other metrics.

## 3.2 Collaboration in Scientific Literature

Many studies have been done in the last few decades to address the properties and difficulties of collaborative group work [29]. The term **CSCW** (Computer-Supported Cooperative Work) has been used in the scientific literature since it was first coined by Irene Greif and Paul Cashman in 1984 [30]. It is often used along with the term **groupware**. It is not part of this thesis' work to decide on what should be the appropriate definition for each expression (as opinions may vary), so we will use these words according to what is more commonly accepted in the scientific community. We assume CSCW refers to the research on the nature of human cooperative work, workplaces and organizations, and

that groupware refers to the actual technology and applications that implement concepts from the field of CSCW.

### 3.2.1 Computer-supported Cooperative Work

According to Schmidt and Bannon [31], "CSCW should be conceived as an endeavor to understand the nature and characteristics of cooperative work with the objective of designing adequate computer-based technologies". This definition illustrates that CSCW's scope goes beyond the field of technology. It is a necessary step, prior to the design and implementation of technical features, where human behavior is examined in different dimensions when performing collaborative work. This allows for human-centered software, focused on improving cooperative work through the analysis of typical human behavior rather than developing practical features alone. As stated by Gross et al.[5],

> "Focusing on the development of technical features ignores the potential influence of distributed information on users (beings part of the socio-technical system) and their behavior. Developers might not receive relevant feedback for their work, spending effort on unreflected improvements".

### 3.2.2 Awareness

In the field of CSCW, some studies identify information sharing, knowledge of the group and each individual's activities, and also coordination as key aspects for a successful collaboration experience [4, 32, 33]. Information about these aspects is typically referred to as awareness. In a popular definition [4], "awareness is an understanding of the activities of others, which provides a context for your own activity". A definition more focused on the case of team development [32] states that "awareness is seen as a means by which team members can become aware of the work of others that is interdependent with their current tasks, therefore enabling better coordination of teams".

As an example, a group tries to complete a puzzle where all individuals are in the same room and each one has been assigned an area of the puzzle to complete. The final goal is to complete the areas that together will make up the puzzle. This activity requires that group members can see which parts of the puzzle have already been pieced together, who handles which area of the puzzle, if someone is having more difficulty in doing their part and need help, and so forth. This sense of awareness is inherent to a physical group activity but not to a distributed group setting, where people in different places are working together through the use of a computer [34]. According to Gross et al.[5], who analyzed different classification systems in several papers, awareness can be classified as:

- **Informal awareness** - A general sense of who is around and what their current and future actions are;

- **Social awareness** - It represents intuitive information about the other group members, like a colleague's emotional state. Typically, it is perceived through nonverbal communication like facial expressions or body language;
- **Group-structural awareness** - Defines the information regarding the roles and responsibilities of the different group elements;
- **Workspace awareness** - It refers to information possible to observe in the workspace, such as the collaborator's interactions with it and the objects it contains.

Workspace awareness comprises several elements. Gutwin and Greenberg[35], describe a framework for creating collaborative software with respect to all existing workspace awareness concerns, as shown in Figure 3.2.

| Category | Element | Specific questions |
|---|---|---|
| Who | Presence | Is anyone in the workspace? |
| | Identity | Who is participating? Who is that? |
| | Authorship | Who is doing what? |
| What | Action | What are they doing? |
| | Intention | What goal is that action part of? |
| | Artifact | What objects are they working on? |
| Where | Location | Where are they working? |
| | Gaze | Where are they looking? |
| | View | Where can they see? |
| | Reach | Where can they reach? |
| How | Action history | How did that operation happen? |
| | Artifact history | How did this artifact come to be in this state? |
| When | Event history | When did that event happen? |
| Who (past) | Presence history | Who was here, and when? |
| Where (past) | Location history | Where has a person been? |
| What (past) | Action history | What has a person been doing? |

Figure 3.2: Elements of workspace awareness [5]

Another system has been proposed for classifying awareness, which should be mentioned since it reflects the notion that collaborative software can be classified according to how people's work is coordinated in time. This differentiation becomes relevant when comparing it with the way groupware is classified, also according to the synchronization of people's actions. In this alternative classification for awareness [5], there are two groups:

- **Synchronous awareness** - "users should be able to obtain some idea of what co-workers are doing, ascertain a co-worker's availability for contact, control their own level of availability, control the information about themselves which is broadcast to others, know when shared documents are in use by others, know exactly what others are doing during a shared editing session" [5];
- **Asynchronous awareness** - "users should be able to determine when shared artifacts have been changed by others, determine how those artifacts have changed, and determine when and where others have left messages for them" [5].

### 3.2.3 Groupware

As previously mentioned, the term groupware is used to refer to collaborative software, which implements the concepts and ideas that originate from the field of CSCW. This means that groupware incorporates a very diverse set of software that was built to support any type of group activity, like messaging, collaborative writing, project management and many other kinds of applications.

When classifying groupware, many systems have been proposed. Most of them derive from the time/space matrix proposed in 1988 by Johansen [36] (see Figure 3.3) which takes two main factors into account: **time** and **space**. Other systems have these factors as a common ground but then propose other aspects to consider, like the number of individuals collaborating [37].

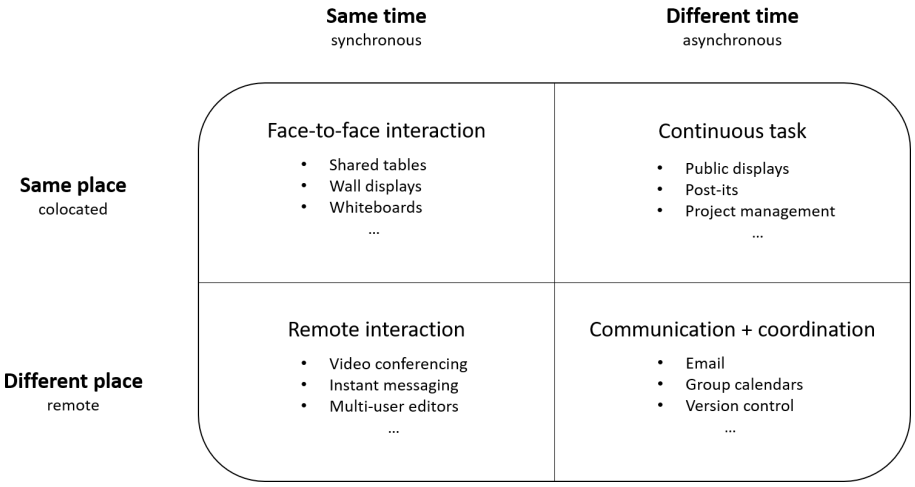



Figure 3.3: Time/space groupware matrix, according to Johansen [36]

In groupware, space refers to the physical location of the group members. They could either be working remotely or all in the same place. Time reflects on whether individuals are working simultaneously or in different periods of time. An example of asynchronous collaborative work is the current collaboration model in OutSystems, where changes done to an application by a developer are submitted to a shared repository where other developers can check on those changes (as described in Section 2.4). As for synchronous collaboration, a clear example is Google Docs, where multiple users can edit a document in real-time (more on this in Section 3.3.2.1). Collaborative software can also be referred to as **semi-synchronous**, when they reflect both synchronous and asynchronous collaborative features [4].

In this thesis, the focus will be on groupware that enables multiple users to collaborate from different places (remote) and at the same time (synchronous) through the use of a computer (or semi-synchronous, as some asynchronous features could prove useful and interesting to explore as a complement to synchronous features). A commonly used expression to refer to groupware designed for people collaborating in a different place

and at the same time is **real-time collaboration** [38].

## 3.3 Collaborative Software

By definition, collaborative software (or groupware) can include many applications that serve distinct purposes. Therefore, it is important to point out that because this thesis covers the topic of Live Collaboration in App Development, the focus of the related work will be groupware designed for real-time collaboration.

In order to get a more practical view of what is the current general development in collaborative software, we have analyzed groupware systems with both synchronous and asynchronous features currently available in the market to any user. Because this thesis will focus on the experience of a real-time collaboration solution for OutSystems, our analysis will emphasize on the user experience and not on implementation details.

### 3.3.1 Asynchronous Collaboration

Version Control Systems (VCS) are commonly used for asynchronous collaboration in editing and sharing of data [39]. Two alternative approaches exist when implementing these systems: lock-modify-unlock and copy-modify-merge. The first model only allows a single person to edit a file at a time, preventing conflicting changes made when more people are editing the same file. The copy-modify-merge approach is a less restrictive alternative where every time someone wants to edit a file, it gets copied to a private local directory; users can then work on their local private copy of the file simultaneously and independently and, eventually, a new final version is created by merging all the local private copies to a single file.

The advantage of VCS is the ability to facilitate collaboration between users as they can contribute to the same project in parallel and easily understand what changes have been made by other people to the project. A user can develop its work independently and, when ready, can submit the changes to a central repository where, if necessary, get merged with other alterations done to the same files by other users. This is considered asynchronous because these changes are done independently of other users and are not immediately reflected to everyone, since they can only be seen when published. The merge operation can also be considered a disadvantage because it can sometimes be troublesome to manually merge all conflicting changes.

A good example is the VCS of the OutSystems Platform (described in Chapter 2), implemented with a copy-modify-merge solution. There are many other different open-source (e.g., Apache Subversion [40], Git[41]) and proprietary (e.g., Azure DevOps Server[42]) VCS' available to users and companies providing different features, technologies, and purposes since VCS' are not exclusive to software development.

### 3.3.2 Synchronous Collaboration

Synchronous Collaboration is more closely related to real-time collaboration as it offers a live experience to users, unlike asynchronous collaboration features offered by VCS. In a synchronized environment, several users can simultaneously collaborate on the same workspace instead of working in isolation on their private local copies. We analyzed several tools available in the market with these features and for different purposes. The collaborative experience of Google Docs is described in this section (see Section 3.3.2.1) because, to our understanding, it has a complete and flawless collaboration experience when compared to other document editing groupware. However, software development is the goal of the OutSystems platform, so it is important to also get a sense of the real-time collaboration features offered by tools in this field. To do so, we analyzed both low-code and traditional software development tools.

The low-code tools which were part of our research were Mendix [43], Salesforce [44], Kony [45], FileMaker [46], Appian [47] and Pega [48]. To analyze how these companies incorporate real-time collaboration in their tools, we have searched in their respective official documentation and installed their applications. Although some of the mentioned platforms offer a very strong collaborative experience, none of them have synchronous collaborative features.

The more traditional development tools that we scrutinized are Microsoft Visual Studio [15], Eclipse IDE [49], AWS Cloud9 IDE [50], Intellij IDEA [51], Atom [52] and Codepen [53]. The research methods were the same as the ones for the low-code platforms, looking for synchronous collaborative features in the documentation and through hands-on testing. The results were different this time, since all mentioned programs support real-time collaboration, either officially or through a third-party plugin. Therefore, not only were we able to conclude that there is a trend in supporting synchronous collaborative features in more traditional development tools, but we also noticed that the user experience is very similar across these tools. In order to get a better sense of that experience, Microsoft's Visual Studio Live Share is described in this section because it includes all features observed in the tools of our analysis.

#### 3.3.2.1 Google Docs Editors

Google Docs Editors is a Software as a Service developed by Google in 2006 [54] which gives multiple users the possibility to concurrently edit files like documents, presentations, and spreadsheets in a distributed real-time collaborative setting. At a given time, multiple users can be editing the same document while being able to see real-time changes taking place as other collaborators edit the document.

After a hands-on analysis of the word processor of Google Docs running in a web browser, it was possible to understand what strategies were implemented to enhance the collaboration process, as it offers several synchronous and asynchronous features to

enable a fluid collaboration experience for users. The main synchronous collaboration features observed were:

- **Awareness information** - When doing real-time collaborative editing of a document, there are two ways that users are aware of the presence of other users. First, there is an indication on the corner of the screen with an icon and name representing each individual, and second, the cursor position of other collaborators is displayed on the respective area of the document, with a different color and name for each user;
- **Automatic saving** - Every small change made to the document is automatically saved to Google's servers [55]; this differs from the reality of VCS, where changes are only reflected on the project when a user publishes its work. This allows for a continuous/synchronized update to the document, which will immediately be reflected to all other collaborators, regardless of whether they are also working on it.

As for asynchronous collaboration features:

- **Comments** - Collaborators can create small boxes of text in specific areas of the document, to share their insights about the document without actually changing it;
- **Role assignment** - In order to control what different users may do, there are three different roles available: editor, commenter, and viewer. Editors, as the name implies, have no limitations and can freely edit the document; commenters can only write comments and view the document; viewers can only view the document and cannot edit or write any comments;
- **Version control** - Similarly to a VCS, Google Docs keeps track of all changes done to the document, enabling users to easily roll back to an earlier version of the document [56];
- **Chat** - When there are several users simultaneously editing the document, they can send written messages to each other inside the Google Docs platform [57].

#### 3.3.2.2 Microsoft Visual Studio Live Share

Microsoft Visual Studio offers the possibility of having a real-time collaboration experience with multiple users through a feature called Live Share [58]. To get started, a user creates a session where they can share their project to anyone with an invitation link. These guests can then join in using Visual Studio or a web browser.

Members of a session can view all the files of the project the host shared with them and can also edit them if the host has given them the required permissions (role assignment feature). Just like Google Docs, users have awareness information, making it possible to

track exactly where each guest's cursor is, each identified with a different color and name. Simultaneous changes to the document are reflected to everyone, and those changes are automatically saved in the host's computer. For direct communication, the platform incorporates audio calls between all participants.

Regarding software development features, users can run and debug their applications in a collaborative manner. Because the source code of the project is stored in the host's computer, the project can only run in that computer. However, any output gets streamed to all other participants so they can see what is happening. During a debug session, the same applies: everything happens on the host computer, but events are viewed by all users in the session.

An important topic of research for this analysis was to understand why software development tools, like Microsoft Visual Studio, have incorporated these real-time collaboration features. In the official documentation [6], there is a highlight of the most common use cases for using real-time collaboration with Visual Studio:

- Quick Assistance
- Pair Programming
- Interactive Education
- Code Reviews
- Technical Interviews
- Working Remotely

### 3.3.3 Summary

When designing a groupware system, there are many aspects to consider when trying to deliver the best possible collaborative experience to users. Concepts of CSCW like awareness have been studied and tested for many years with connection to the field of social sciences to comprehend human behavior when collaborating with others. The challenge is to bring these ideas to the world of information technology, to serve as a basis for building powerful and useful collaborative processes and features.

Synchronous collaborative features exist for several purposes, like document editing and software development. Google Docs has been around since 2006 [54], bringing real-time collaboration to millions of users around the world who use its capabilities to collaborate with other people to create documents, presentations, and spreadsheets.

In traditional software development, many companies are following Google Docs' footsteps and incorporating a real-time collaboration experience into their development tools. This creates an opportunity for OutSystems to follow that trend and design a real-time experience in Service Studio, which would be unique among low-code development

platforms. Not only could this put OutSystems on the edge of current real-time collaboration technology, but also be a relevant feature for common use cases among developers and software development projects. To understand the use cases that apply to the OutSystems ecosystem, several interviews were conducted with development teams (see Chapter 4).

## 3.4 User Experience and Usability

To design the best experience for a live collaboration feature in OutSystems, user experience and usability are an important part of the process. Real-time groupware needs to be tested for usability to attest for user-friendliness, or it faces the risk of not being adopted by the target users [59].

It is important to first distinguish between **user experience** (UX) and **usability**. User Experience (UX) is a wider concept that "encompasses all aspects of the end-user's interaction with the company, its services, and its products" [60]. Usability is the "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" [61]. This shows an important difference, as usability specifically covers the interaction of a user with a system, product or service, making it a more specific field of UX. Another definition of usability describes it as a composition of five dimensions called "5Es" [62]:

- **Effective** - How completely and accurately the work or experience is completed or goals reached;
- **Efficient** - How quickly this work can be completed;
- **Engaging** - How well the interface draws the user into the interaction and how pleasant and satisfying it is to use;
- **Error tolerant** - How well the product prevents errors and can help the user recover from mistakes that do occur;
- **Easy to learn** - How well the product supports both the initial orientation and continued learning throughout the complete lifetime of use.

Two important concepts for categorizing usability are **"little u"** and **"Big U"**. As explained by Barnum [63], "little u" usability represents **usability testing**, which focuses on observing and learning from users while they interact with the product, simulating a real situation where they perform a series of tasks meaningful to them. This way, researchers can realize if their product could be improved to better fit the way users use it. "Big U" usability is a broader concept that encompasses "little u" usability testing and everything else that goes into creating a product made for people, it "encompasses the entire process and includes all the techniques in the usability specialist's toolkit" [63].

### 3.4.1 Little u

As mentioned, "little u" or usability testing means the activity of observing users working with a product, performing real tasks. This testing can be subdivided into two types, depending on when it happens during product development [64]:

- **Formative testing** - done while the product is being developed, with the goal of finding and fixing usability issues;
- **Summative testing** - done after the product is finished, to validate that the final product meets the established requirements.

Several tools and techniques exist for conducting formative testing, but a general baseline is described as follows [63]:

1. **User profile** - defining one or more subgroups of the user population as a basis for finding participants for the test;
2. **Task-based scenarios** - creating scenarios with tasks designed to reach a specific goal is important to give all participants the same context, resulting in unbiased evaluation results;
3. **Think-aloud process** - using a think-aloud process has participants sharing its thoughts out loud while iterating with the product and completing the tasks;
4. **Correct and test again** - after analyzing the results from the usability test, make the necessary changes to the product to fix any discovered problems and test again.

Usability tests can be done in the field (e.g., at the users' workplace), in a dedicated lab with specialized tools and personnel or remotely. Due to the current COVID-19 pandemic context, there are limitations to how these tests are performed, making remote usability testing the only option for most cases.

Remote usability testing can happen in two ways: synchronously or asynchronously. In a synchronous (or moderated) remote test, there is the presence of a moderator, observers and the participants, much like a test conducted in the same physical space. In an asynchronous (or unmoderated) remote test, it's possible to reap the benefits of working in a remote setting since participants can use an application that "captures the screen, keystrokes, mouse clicks, navigation path, drop-off rates, and so forth and collates these data into a report" [63], without the need of having one or more individuals accompanying the user through the test.

### 3.4.2 Big U

"Big U" encompasses a bigger scope than "little u", involving many tools and techniques, going beyond just usability testing, to ensure that a product complies with all dimensions

of usability. Barnum enumerates several of these tools and techniques which can be used before, during, and after a product's development [65]. However, some methods yield better results when applied to the context of a software product [66]. These methods are **Heuristic Evaluation**, **Cognitive Walkthrough** and **Action Analysis**.

Heuristic Evaluation (HE) is a widespread method for discovering potential usability issues in user interfaces [66, 67]. It involves a group of usability experts visually inspecting the various elements of the product and comparing it to a list of usability guidelines. Each expert starts by doing its own individual evaluation, followed by a meeting between all experts where they can then share their insights and aggregate their discoveries. This makes sure each evaluation is not susceptible to some form of group pressure or bias and maximizes the number of discovered issues.

In Cognitive Walkthrough, a team member takes the role of a user and walks through the product, step-by-step, for a given task. The goal is for designers to understand the user's perspective, by trying to go through the different elements of the product and evaluating the ease of learning and related concerns [66]. It offers a way of realizing the user's concerns and assumptions without requiring to iterate with them, although that can also make the evaluation susceptible to an inherent bias from the designer.

Action Analysis (or Keystroke-level Analysis) is performed by a team member and focuses on inspecting the necessary individual actions performed to complete a task [66]. It breaks down the process of task completion to a sequence of single actions performed by the user, as well as registering the time taken for each step (e.g., moving a computer mouse to a part of the screen).

### 3.4.3 Concept Testing

Another useful technique, used before a product's actual development, is Concept Testing [68]. While usability testing involves observing users while they interact with the product to look for any issues, after it has been developed, concept testing happens before a product's development life cycle. Its goal is to validate with the product's target audience which ideas are worth pursuing and which ones are not.

For example, looking at this thesis' main objective (see Section 1.3), the goal is to create a real-time collaboration experience able to bring value to users of the OutSystems Platform. Because there are different features and ideas related to real-time collaboration that could be implemented, and the company must spend time and resources to develop them, it is not viable to pursue all of them so that, upon release, users dislike or cannot take any value from them. Concept testing can help in this situation by validating potential ideas with the target users beforehand to get a better sense of which ones are worth pursuing and which ones are not.

According to a guide created by Maze [69], a company that creates products and helpful guides related to remote testing, there are four different methods for concept testing [70]:

- **Monadic Testing** - "It involves dividing test participants into groups and showing each group one concept to answer in an in-depth survey on the concept. So, for instance, if you're testing various feature designs, give each group only one design variant to review and share insights on." [70];
- **Sequential Monadic Testing** - "As with monadic testing, sequential monadic testing requires you to divide test participants into groups. The chief difference, however, is that all groups are shown all the concepts at once." [70];
- **Comparative Testing** - "This one's the simplest of all the concept test methods since you share two or more concepts with survey respondents and ask them which they like the best." [70];
- **Protomonadic Testing** - "This method involves asking respondents to evaluate different concepts, followed by doing a comparative test to summarize which variant they prefer. Put simply, a protomonadic test is a combination of sequential monadic and comparative tests." [70].

Choosing the right concept testing method is important as it depends on what insights from users the research requires and on the resources available to do (e.g., some tests may require a very big amount of participants). Having chosen the appropriate method, the same guide created by Maze enumerates the steps required for a successful run of a concept test [71]:

1. **Set a goal for your test** - Exactly stating what we are trying to achieve with the concept test will help focus on the main objective of the test throughout the process;
2. **Craft your script and questions** - Creating a group of questions to ask all participants reduces potential biases in the test results as all participants are asked the same questions. These questions should also be carefully crafted as to not bias the test participants;
3. **Recruit the right participants** - Test participants should be part of the target users of the product concept being tested. It is also relevant to include a diverse group of the target users, varying in gender, age, and other factors to realize which features of the concept appeal to specific groups within the target audience;
4. **Determine the flow of the concept test** - A typical concept test starts with a brief introduction to the concept which may or may not include a prototype or mockup of it. Then, some screening questions to understand the participant's background, experience with similar products, or any other metric that could be relevant for later analysis. Finally, there should be questions about the overall concept while diving deeper into questions about more specific characteristics of the concept and possibly ask for a comparison (e.g., can you order the concepts presented to you according to your personal preference?);

5. **Integrate quantitative measurements** - To make sense of all the data from the tests, having some quantitative measurements can facilitate the analysis, by assigning points to questions or having participants fill in a written survey at the end with quick questions using, for example, a Likert Scale.

In the previous enumeration, there were some relevant topics that are worth exploring a little more in the context of this thesis. During a test, biases are very important as they can influence the outcome of a research work, as they could lead to wrong results and conclusions (this is further explored in Section 3.4.3.1).

#### 3.4.3.1 Biases

In the scientific literature, many psychological studies exist about the biases that influence the human mind. Because part of our research involves making design choices for a real-time collaboration product, it becomes important to dig deeper into the strategies to reduce any biases that can influence the outcome of this work. Among the many types of biases that exist, **cognitive biases** are our main interest as they represent "cases in which human cognition reliably produces representations that are systematically distorted compared to some aspect of objective reality"[72]. Cognitive biases happen when people have their own subjective interpretation of reality, which they acquire from their perception of the world. A common example which strongly relates to our research work is the **confirmation bias**, which is a tendency of people to more easily accept and select information that confirms their prior beliefs and to ignore opposing information [73]. Looking more deeply into the field of design and IT, Liedtka (2015) concludes that a **Design Thinking methodology** could reduce cognitive biases when designing a product, since it mitigates a well-known set of cognitive weaknesses. As she states [73],

> "Humans often project their own worldview onto others, limit the options considered, and ignore disconfirming data. They tend toward overconfidence in their predictions, regularly terminate the search process prematurely, and become overinvested in their early solutions—all of which impair the quality of hypothesis generation and testing. A design-thinking methodology, I have suggested, may help decision-makers address many of these inadequacies".

Biases affect both researchers and test participants alike. In this effort for reducing biases, research should be well planned as to reduce any biases from the researcher while working and prevent falling into any pitfalls. However, testing should also be well-thought-out to avoid skewing the users' feedback. Since concept testing is a process which is part of this research, a specific type of cognitive bias common in these tests is the **Framing Bias**. Framing is an occurrence when the exact same information can lead to different opinions and conclusions, depending on how it is presented. Therefore, when testing with users, it is important to follow some strategies, as a UX researcher conducting a user test, to reduce the impact of the framing bias [74]:

- Do not rush the planning of the tests. Carefully considering all alternatives for framing for the information to be presented to users is important as to make sure that the information is not being presented in such a way that biased the thoughts and opinions of the test participants;
- Gather enough context to present the information without being biased towards a certain path;
- Try different frames, asking the test questions in different ways and with distinct points of view, and choose the one with the potential to influence test participants the least.

# Chapter 4

# Methodology

This chapter presents the techniques and procedures used to research on the topic of Live Collaboration in App Development. It also justifies the choices made for the steps to guide this thesis' work.

## 4.1 Design Thinking

Because this dissertation's topic is broad, it is necessary to understand the issues and inefficiencies that currently happen when collaborating with OutSystems, to which Live Collaboration might be a solution (or not, since we should conduct our research while being agnostic about the solutions). We tried to find a methodology able to research about these problems but also to design a solution for them. To achieve this, we followed the ideas coming from **Design Thinking**.

Design thinking is a methodology which comprises a set of processes and principles focused on building effective and meaningful solutions centered on users. This allows for human-centric design applied to end products, so there is a care for design concerns right from the start of the product's development as to give as much value as possible to its end-users. As Brown (2008) puts it [75], design thinking "is a discipline that uses the designer's sensibility and methods to match people's needs with what is technologically feasible and what a viable business strategy can convert into customer value and market opportunity". The definition reflects the importance of design for customers but also for the business, as it needs to be aware of opportunities available in the market to better plan its strategy. This reflects one motivation for this dissertation, that there are no real-time collaboration features available in other low-code platforms similar to OutSystems (see Section 3.3). It could be advantageous for OutSystems to explore this field, and design thinking processes take this into account.

As mentioned in Section 3.4.3.1, it has been studied that a design thinking methodology applied to product development can reduce cognitive biases which are inherent to any human thought-out process [73]. Minimizing these biases is essential, as they can lead to incorrect results and conclusions. To further make a case for design thinking, the Nielsen Norman Group, a reputable UX research and consulting firm, summarizes the main advantages of applying this ideology to product development [76]:

- "It is a user-centered process that starts with user data, creates design artifacts that address real and not imaginary user needs, and then tests those artifacts with real users";
- "It leverages collective expertise and establishes a shared language and buy-in among your team";
- "It encourages innovation by exploring multiple avenues for the same problem".

Hence, design thinking is a great option for guiding the work of this dissertation. It is a flexible methodology, it is not a fixed set of steps, like a recipe for success. Instead, it is more of a framework with enough flexibility to be tweaked when necessary. Overall, it has three main stages: 1) **understand**, 2) **explore**, and 3) **materialize**. These stages comprise six more specific phases: **empathize**, **define**, **ideate**, **prototype**, **test**, and **implement**. This composition is represented in Figure 4.1. Each phase has its own purpose [76]:

- **Empathize** - Involves intensive research to gain knowledge about what users do, say, think, and feel;
- **Define** - All the knowledge gained is combined to identify the problems affecting the users. Tackling those problems can then present opportunities for innovation;
- **Ideate** - Involves the generation of lots of different ideas, no matter how crazy they might seem, to solve the problems defined in the previous step. In this phase, quantity is actually more important than quality, so multiple pathways can be explored;
- **Prototype** - Creation of real representations of the ideas generated before. The goal is to understand their feasibility and to get a more visual depiction of the ideas;
- **Test** - Going back to users, to get their feedback about the proposed solutions. A successful solution is able to improve how users feel, think or do their tasks;
- **Implement** - If the ideas have a green light from users in testing, it is time to put the vision into effect and the proposed solutions can now be materialized.

As mentioned, design thinking has flexibility and should not be followed linearly. What this means is that each phase is part of an iterative and cyclical process, where it is possible to go back and forth in the different phases as needed, as shown in Figure 4.2.

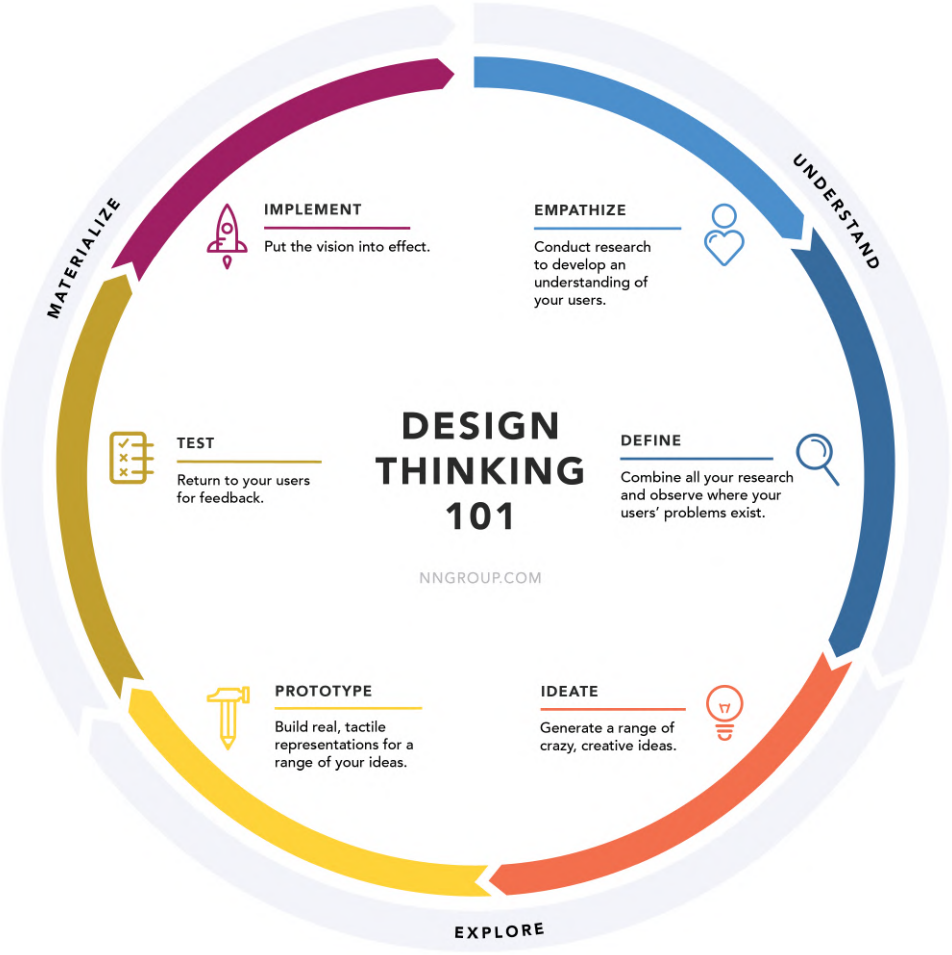



Figure 4.1: Diagram representing the Design Thinking ideology [77]

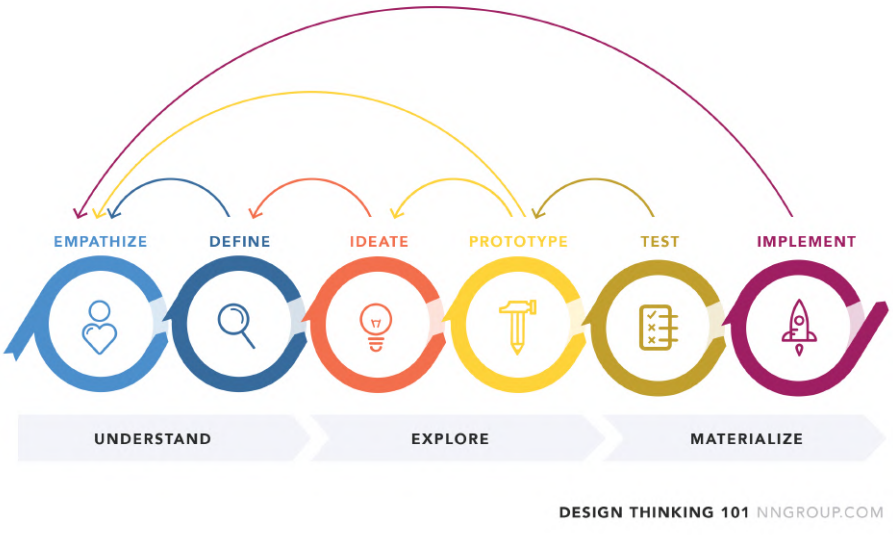



Figure 4.2: The iterative and cyclical process of Design Thinking [77]

The following sections are organized according to the steps of the design thinking methodology to explain what has been done in each phase. The **implement** step is not part of this thesis' work, since the aim is to idealize the experience of a collaborative feature in OutSystems' Service Studio and leave its implementation details for a future work. Also, the focus of this collaborative feature is on Service Studio's Actions and Screens, which represent an application's logic and user interface (see Chapter 2), as these are by far the elements where developers spend more time (based on the platform telemetry data).

## 4.2 Empathize

When empathizing with users of the OutSystems platform, it is important to realize what users **do, say, think,** and **feel**. To achieve this, we have followed four strategies:

- Interviews
- Literature Analysis
- Competitive Analysis
- OutSystems Platform Telemetry Data

A direct one-on-one **interview** with users of the OutSystems platform was the main method to comprehend the issues they might face with collaboration. Insights from these interviews allowed an in-depth understanding of the friction points when collaborating in software development projects with OutSystems. How these interviews were structured, and their results, are explored in Section 4.2.1.

**Literature analysis** involved looking at what the scientific literature has studied regarding collaboration and collaborative software and its intricacies. This allows us to get a better sense of how collaboration between people working with each other from different physical locations can be improved by collaborative software. Section 3.2 details this analysis. We also looked at a 2013 Master's thesis conducted at OutSystems [24] about live collaboration, and the OutSystems official documentation, to create a detailed study about the state of collaboration in the OutSystems ecosystem. Insights from the interviews also helped in achieving this goal. Our findings can be found in Section 3.1.

Examining the tools available on the market with real-time collaboration features is relevant to realize what features are currently available to the public. Therefore, we did a **competitive analysis** to identify trends and opportunities in collaborative features and work on them for our product, always with the end-users in mind. This is elaborated in Section 3.3.

### 4.2.1 Interviews

To better understand how collaboration happens in the OutSystems ecosystem and potentially find issues and inefficiencies, we conducted remote interviews using video conferencing software, with development teams and managers between December 2020 and January 2021. Our goal was to understand the reality of collaboration in development teams, especially when working in a distributed setting. For that purpose, we elaborated a script with five questions for the interviews to achieve consistency and reduce any biases:

1. How does collaboration take place in the OutSystems ecosystem?
2. What difficulties/inefficiencies are experienced during collaboration?
3. How often do said difficulties/inefficiencies occur?
4. What is the impact of these difficulties/inefficiencies?
5. How do people work around them?

The script assumes that people can identify some difficulties when collaborating, which might not be true. Therefore, before asking the second question, we would first ask a simple yes or no question to understand if the interviewee had experienced any issues during collaboration. However, all participants had issues to report regarding collaboration in the OutSystems ecosystem.

Asking about collaboration also created some doubts among participants, since it is an extensive topic and could refer to many things. We also omitted real-time collaboration throughout the interviews. This was intentional, to allow the interviewees to elaborate openly on their experiences with no bias in their thinking.

To get an accurate depiction of the reality of collaboration, interviewees were chosen as to represent both internal and external OutSystems professionals. This distinction was important because internal teams could have working processes different from other businesses. It was also relevant to interview individuals with different roles. To determine potential candidates, we used OutSystems' internal communication channels to find OutSystems employees and the OutSystems Community[1] to get in touch with clients and business partners, Most Valuable Professionals (MVPs), of OutSystems.

In total, 14 people were interviewed. Ten were working at OutSystems and four were clients or MVPs of the company. From the people working at OutSystems, based on their roles, we interviewed six Developers, two Tech Leads, one Engagement Manager and one Solution Architect. As for the clients and MVPs, we interviewed: a Solution Architect, a Tech Lead, a Founder & CEO, and a Functional Specialist.

The results of the interviews explained why OutSystems developers need to collaborate and the difficulties that turn up during the process. According to our research,

[1] https://www.outsystems.com/community/

collaboration happens on three main levels: between **team members**, between **different teams** and **with clients or users external to the organization**.

Based on our observations, collaboration is a crucial process across these levels to ensure: **acceleration** of product development; **alignment** with the current status of the application; **traceability** of the changes being made to the application by all people involved in a project; **engagement** of team members in some part of the development process.

During the interviews, every new mention of a problem was added to a list. In the end, similar problems were grouped into the same category. Appendix A summarizes all issues reported during the interviews and their classification (explained in Section 4.3).

Analyzing the reported issues, it is possible to understand that the most common use cases for Live Share in Visual Studio (see Section 3.3.2.2) also apply to the OutSystems ecosystem. Quick assistance, pair programming, interactive education, code reviewing, technical interviews, and remote work were all mentioned during the interviews, making real-time collaboration a possibility to replace or complement the current processes in place for dealing with these situations.

An issue not covered by Visual Studio's use cases is **"Product architecture's focus should be on bringing value to the customers and the business. However, the architecture of a product also has to be planned in order to avoid conflicts between developers during development, decreasing the value it can deliver in the process"**. It shows how an organization's work and communication methods can highly influence the final product they are developing. This situation is not new to the world of software development, being commonly referred to as **Conway's Law** [78], which states that "Any organization that designs a system (defined broadly) will inevitably produce a design whose structure is a copy of the organization's communication structure".

Throughout all interviews, everyone reported some negative experience regarding the merge operation. The reports focused on: crashes, slow merging operation, and the long time taken to deal with the manual merge operations. It is a time-consuming activity because developers need to be careful when merging changes in order to publish their new changes without breaking the ones from other developers. Part of this process involves communicating with their peers to understand what has been done and how their changes should be merged to get the desired outcome.

To get a more accurate depiction of these problems, we accessed the platform telemetry data which has many statistics about the usage of the OutSystems platform by developers. These are the most relevant metrics for our research (regarding the period of January 2021 - June 2021):

- 1.6% of total development time is spent on the merge operation, making it the 11th most significant area of impact on productivity;
- Approximately 15.6% of all merges are manual (forcing developers to fix the conflicts manually) and they take, on average, 130 seconds, which is 14x longer than

automatic merge operations (9 seconds);

- About 7.9% of manual merge operations are not completed, as they are canceled by developers (they give up on going through with the manual merge);
- 0.05% of all merge operations ended up with Service Studio crashing.

Looking at these numbers, some conclusions can be drawn. 1.6% of total development time spent on the merge operation might seem little, however, this is time spent dealing with work idiosyncrasies and not on creating direct value to the end-product, and as such it should be minimized as much as possible. Moreover, this percentage does not reflect the time developers spend communicating to prevent or deal with conflicts of the merge operation (a metric very hard to measure), as it was commonly reported throughout the interviews as a very time-spending activity.

A manual merge requires user intervention, happening about 15.6% of the time, and it is slower than an automatic merge. Therefore, maximizing automatic merges should be a priority to reduce the number of times that developers need to intervene and save time. Manual merges only happen when the platform cannot incorporate all changes automatically. The main reason for this is the granularity of the merge operation, as it was reported that the platform considers a conflict any changes made within the same element (like a Screen or Action), even if they are independent of each other.

During our inquiries, interviewees, overall, shared their thoughts regarding the merging process as its functionality could have some improvements. Although there is room for improvement, the merge operation is seen as part of the development process and required for collaboration. Therefore, having 7.9% of merge operations cancelled could mean that developers prefer talking with other developers or redo their changes, rather than immediately merging their changes with the work of other developers.

The OutSystems platform telemetry also reports that about 0.05% of the merge operations crashed, which seems to be happening with bigger, more complex applications. Despite the low crash ratio, these crashes can have a catastrophic effect, often leading to lost work and/or rework and support intervention.

The analyzed data and reports from the interviews exposes some downsides of the merge operation. It can lead to some friction points during the development process, causing some project delivery delays and concerns from developers. In the next section, we analyze all this data collectively and define the most relevant issue of collaboration in OutSystems, considering also other aspects beyond the merge operation.

## 4.3 Define

After gathering all feedback from the interviews, it was vital to make sense of the data collected. All issues were grouped so a single category would represent similar problems. Then, they were classified and analyzed to understand which problem was more relevant.

This is the "define" phase, where we try to understand where our user's problems exist and then try to prioritize them to focus our attention in the most relevant one. **RICE** (Reach, Impact, Confidence, Effort) [79] is a prioritization framework that evaluates each idea with a score, based on four attributes whose definition was adapted for this study. It is appropriate for the work of this phase since it can compare the problems we are analyzing against each other, based on a common formula with the following properties:

- **Reach** - How frequently these problems happen during development with OutSystems;
- **Impact** - The repercussions of a certain problem. For example, a problem with a high impact could make a business lose money, and a low-impact problem could cause a meaningless delay in a project's delivery time;
- **Confidence** - The amount of certainty we have in the values for Reach, Impact and Effort (e.g., if the OutSystems platform telemetry data has data backing those values, for example, our confidence should be close to 100%; but if our classification is based on a "gut feeling", the confidence level is much lower);
- **Effort** - The commitment/energy for completing the three stages of solving the problem: finding the best solution for that problem, developing it and "selling" it to the individuals that will use it.

Then, each problem gets a final score based on the following formula, where a higher score means it is a bigger priority:

$$Score = \frac{Reach * Impact * Confidence}{Effort}$$

Appendix A lists the problems users reported in the interviews, which were classified according to the RICE scoring model.

The three highest scored problems are further analyzed and discussed, with the highest scored problem being the focus of the next steps of this thesis' work. The issue with the third highest score is **"Difficult to identify changes when trying to resolve merge conflicts in more complex screens of the Interface layer"** (e.g., the Web/Reactive Screen) and it relates to the friction points of the merging process. Developers complained that during the merge operation it is hard to recognize the differences between their version and the server's version and choose the right one (see Figure 2.7 for an example). In particular, interviewees reported that they have difficulties recognizing the changes when merging a screen of an application's interface layer, especially when it is large and complex, with many components. This creates in developers a feeling of anxiety and uncertainty because they could be overwriting someone else's work.

This issue gets a high impact score because of the time lost trying to solve it. A medium score for the reach is appropriate, since not all OutSystems software projects require the development of a user interface or have a very complex one.

**"Conflicts happening with small changes due to the granularity of the merge operation"** is the issue with the second highest score. Several developers talked about publishing their work and getting a conflict detection window in Service Studio when multiple developers are working on different sections of the same object (see Figure 2.6).

One developer (referred from now on as Developer A) gave a real example where he was altering the header of a Web Screen at the same time as another developer (referred from now on as Developer B) was editing the footer. Both developers started their work from the same common version. Developer B published his work first and when Developer A published his changes after, Service Studio detected a conflict with both versions, requiring a merge operation. The platform cannot detect that, although the same screen was changed, the changes were done in different independent sections. This is where the term "granularity" applies, as it describes the size of the scope of changes that the merge operation can compare. As a result, this forced Developer A to stop his work and communicate with Developer B to understand if there really was a conflict. Developer A states that sometimes it is easy to understand exactly what happened, but sometimes it is necessary to discuss the changes with Developer B to understand if merging will not produce unwanted results. Even with this communication, Developer A after choosing between their version and the version of Developer B, will need to replay all the changes that were in the version which was not chosen.

This issue is scored with a high impact, as it involves an inefficient use of the time allocated to a project, resulting in longer project deliveries. It has about the same reach and confidence score as the previous problem because we believe it is a slightly more common situation.

Finally, the highest score is assigned to **"Delays and inefficiencies with excessive communication between developers (from different teams or the same one) to make sure no conflicts happen"** (e.g., calls, extra daily meetings). This problem originates because conflicts are constantly avoided, forcing developers to have many iterations with each other. They communicate to make sure everyone is working on distinct modules of the application or on different independent objects of the same module. This problem was reported by almost every interviewee, giving it a high reach, and it has a relevant impact since it highly influences the time for completing a project, a measurement often referred to as Time To Value (TTV).

All problems were scored in a group session, where the persons involved with this thesis research work expressed their opinion regarding the values for the attributes of the RICE framework for each problem. Based on this, we have thus decided that the main problem to tackle is the one which achieved the highest score. Following the "define" phase, the focus of our work shifted from the "understand" stage of design thinking, where we explored our users and the problem space of this dissertation's topic, to the "explore" stage, where we traverse the solution space.

## 4.4 Ideate

The "ideate" phase represents the first steps in thinking of different ideas to solve the problem defined earlier: **"Delays and inefficiencies with excessive communication between developers (from different teams or the same one) to make sure no conflicts happen"** (e.g., calls, extra daily meetings). The main goal is to come up with as many ideas as possible, no matter how crazy they might seem, to make sure all possibilities are explored for solving the problem at hand. To start a discussion on potential solutions, we organized an **ideation session**. It is a creative work session with designers, developers and stakeholders to generate solutions in a collaborative and efficient way. It was proposed and explained by the UX department of OutSystems, which recurrently uses this process internally, taking its principles from the book "Sprint: How to solve big problems and test new ideas in just five days" [80].

The ideation session is structured to last between 1h30m and 2h00m, starting with a presentation of the problem and research results to all participants, to give them some context. Following that, participants will have to sketch different solutions for the presented problem and, finally, each participant explains to everyone their ideas and designs. To structure the session and make a more efficient use of everyone's time, we followed the agenda represented in Table 4.1, with the session taking place remotely through the use of video conferencing software on March 29, 2021.

Table 4.1: Agenda for the ideation session

| Time | Duration | Activity | |
|---|---|---|---|
| 0:00 | 10 min | Present problem and research results | |
| 0:10 | 20 min | Sketch | Gather key info |
| 0:20 | 20 min | | Doodle rough solutions |
| 0:40 | 10 min | | Crazy 8s |
| 0:50 | 30 min | | Figure out the details |
| 1:20 | 3 min * participants | Each participant explains their design | |
| 1:xx | 5 min | Wrap up and close | |

In the session, a moderator was responsible for making sure the agenda was followed, focusing the discussion on more obvious points to avoid breaks and to ensure that everybody takes part. Six participants were actively discussing ideas and designs, which were chosen from different departments within the OutSystems company and from other companies, as to have a diverse group of people regarding their roles and backgrounds. All participants had a deep knowledge and understanding of the OutSystems platform and Service Studio. Their role and experience are described as follows:

- **Participant A**: A Computer Science Master's student knowledgeable in OutSystems

but with but no experience in software development projects;

- **Participant B**: A software engineer working at the engineering department of OutSystems with no experience in software development projects with OutSystems;
- **Participant C**: A tech lead (see Section 3.1.3 to understand the responsibilities of the role) in OutSystems with years of experience in software development with OutSystems;
- **Participant D**: A product designer working at OutSystems with experience in software development projects with OutSystems;
- **Participant E**: A product manager working at OutSystems with no experience in software development projects with OutSystems;
- **Participant F**: A software development manager with a large experience in projects done with OutSystems but working at another company.

Some participants have experience in projects developed with the OutSystems platform and others do not, because they work on the development of the platform and not with the platform itself.

The sketch activity happens after giving context of the problem being addressed to all participants. It is the most important part of the session since our goal is to get new ideas from the analysis of the final sketches from each participant. As shown in Figure 4.3, it is split into four steps:

1. **Gather key info** - To start, participants should gather their thoughts, revisit the initial presentation, do some research and identify major points to address. Everything they write or draw will not be shared with the rest of the participants;
2. **Doodle rough solutions** - Participants now draw rough ideas, draw doodles, sample headlines, diagrams, stick figures in a sheet of paper - anything that gives form to their thoughts. As in the previous part, none of this will be shared with the rest of the participants;
3. **Crazy 8s** - Each person takes his or her strongest ideas and rapidly sketches eight variations in eight minutes. This exercise forces people to push past their first reasonable solutions and make them better, or at least consider alternatives. The "crazy" in this exercise refers to the pace of the sketching and not to the nature of the ideas, as in this phase the focus should now be on the good ideas;
4. **Figure out the details** - Finally, each person will sketch their best idea as a three-panel storyboard and with as much detail as possible, so it can easily be understood by all other participants.

Appendix B shows the final drawings sketched by every one of the six participants.

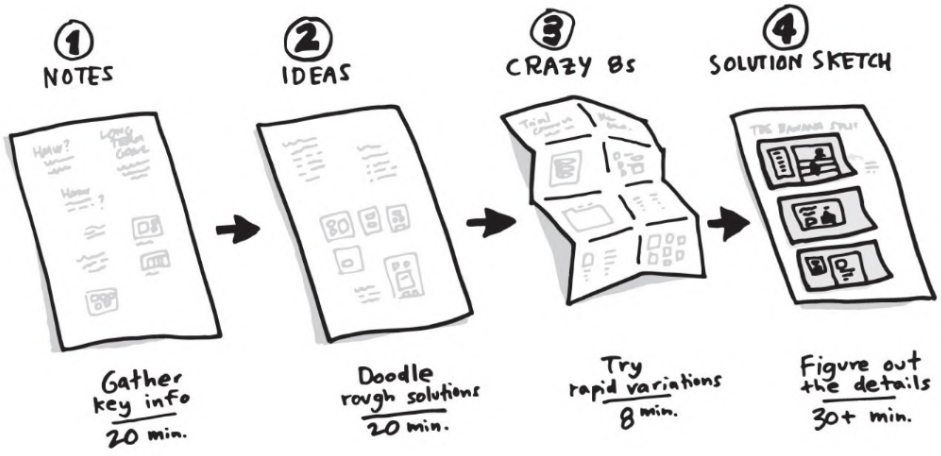



Figure 4.3: Structure of the four-step sketch

### 4.4.1 Proposed Solutions

Because there were many ideas and opinions coming out of the ideation session, we will categorize them into general solutions and then elaborate on how different participants mentioned them and why we have decided to pursue them or drop them. The categories are:

- Awareness of other people working in the workspace
- Awareness of changes being made by other developers
- Automatic updates to the local module
- Improvements on the manual merge operation
- Improvements on the granularity of the merge operation
- Issue tracking

The first category, **"awareness of other people working in the workspace"**, was referred by most participants in the ideation session. Right now, in Service Studio, there is no awareness of who else is "online" in the module the developer is working on. There is no information on who has permission to edit the module and Service Studio does not support any kind of direct communication (chat, calls, etc.). Since preventing conflicts is one of collaboration's goals, there could be more features in Service Studio to promote it. Participant F (see Figure 4.4) suggested having a top bar which shows which team members are online, which is to say that they are logged into their accounts and with Service Studio open in the same module. Such a feature would also allow for group calls to be made to create a session to exchange ideas or solve any problems more collaborative. Then, there would be a sidebar which displays who from the team has actively

been publishing changes to the module. This gives the developer the ability to know the developers who have been more actively working on that module.

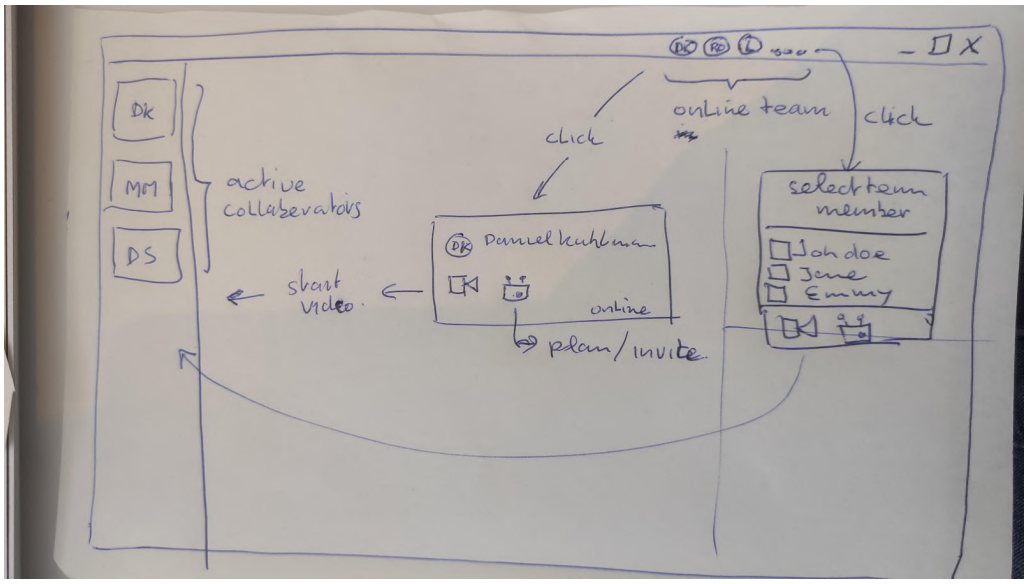



Figure 4.4: Sketch from Participant F

Participant A also had a similar design, with a top bar indicating who is online, but whose icon would go more transparent if the user had not edited the module in a certain amount of time. It also shows the possibility of having awareness in Service Studio's right-side tree of which actions/screens other developers are working on (see Figure 4.5).

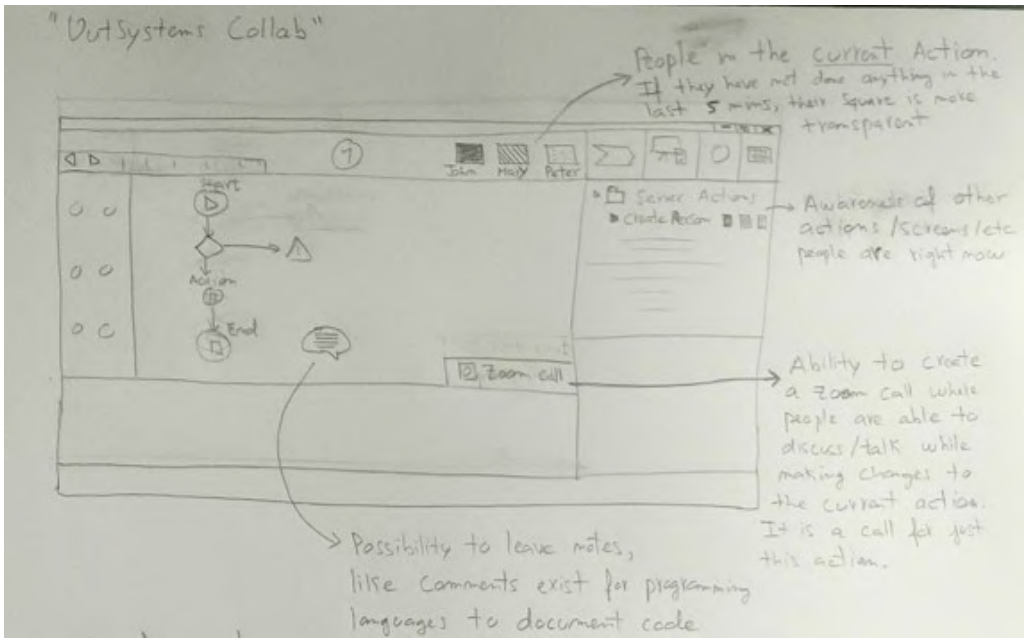



Figure 4.5: Sketch from Participant A

Overall, this type of solution seems beneficial to prevent conflicts, as knowing where

people are working inside the module is information that can help avoid clashes where several people are editing the same elements. Having the possibility for direct communication in Service Studio can be a way of facilitating and accelerating communication between developers to check on each other's work faster.

The second category, **"awareness of changes being made by other developers"**, is a step further from the first category. Besides being able to see who is online and where they are, this solution represents the ability to see what and where are the changes they are making to the module. Participant C showed a way of knowing about changes happening in the module through notifications displayed next to the 1-Click Publish Button and in the App Layer Tabs, as it can be seen in Figure 4.6. These notifications distinguish between breaking and non-breaking changes, according to how the module is affected by them. Another type of highlight was more specific and shown right on top of an action, for example, stating who removed, added or changed specific nodes of that action. Another example of these highlights was exemplified by participant E (see Figure 4.7).

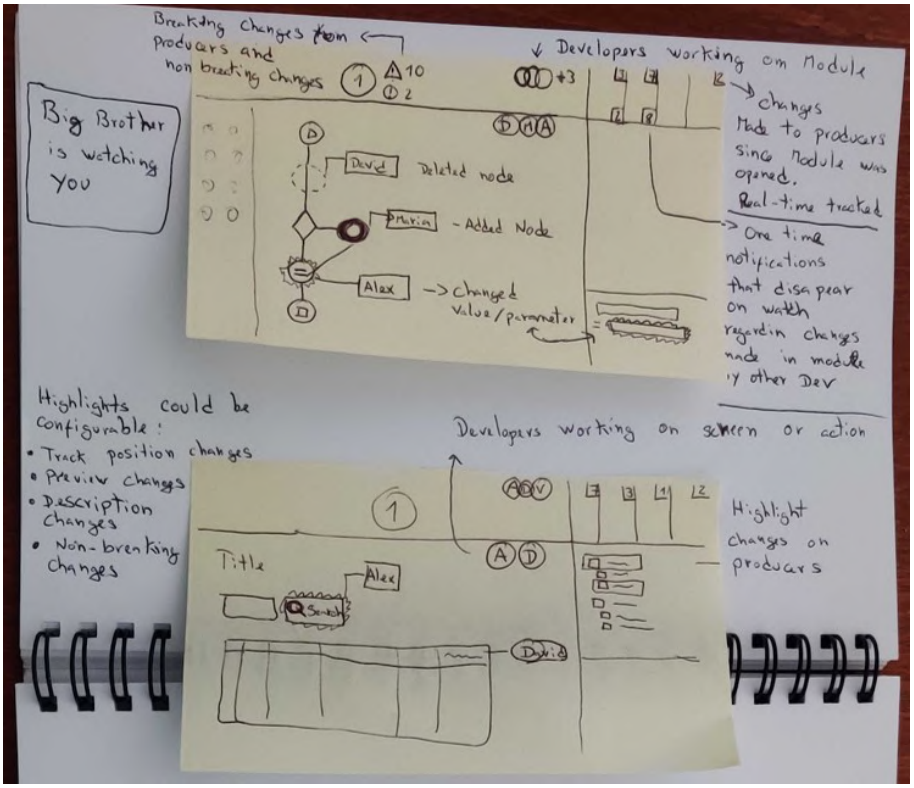



Figure 4.6: Sketch from Participant C about awareness on changes

This category of solution also presents a way of giving more information to the developer of what is happening to the elements of the module he is working on. As such, it can prevent conflicts, and it should be prototyped and tested to see if it really can bring any benefits to collaboration.

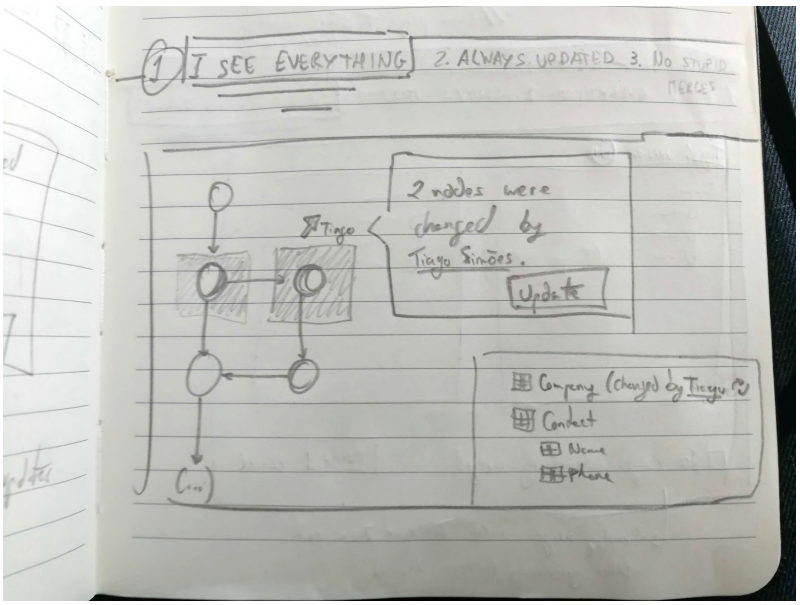



Figure 4.7: Sketch from Participant E about awareness on changes

**"Automatic updates to the local module"** was suggested by participant E and the idea consisted of continuously and automatically updating the local module with changes being published by other developers (see Figure 4.8). This way, the developer can always be working on the latest version of the module and thus, it would dramatically reduce the need for the merge operation. However, some participants raised questions about how this can potentially be too disruptive and bring more disadvantages than advantages. As such, it becomes even more essential to test this idea to understand its potential for improving collaboration.

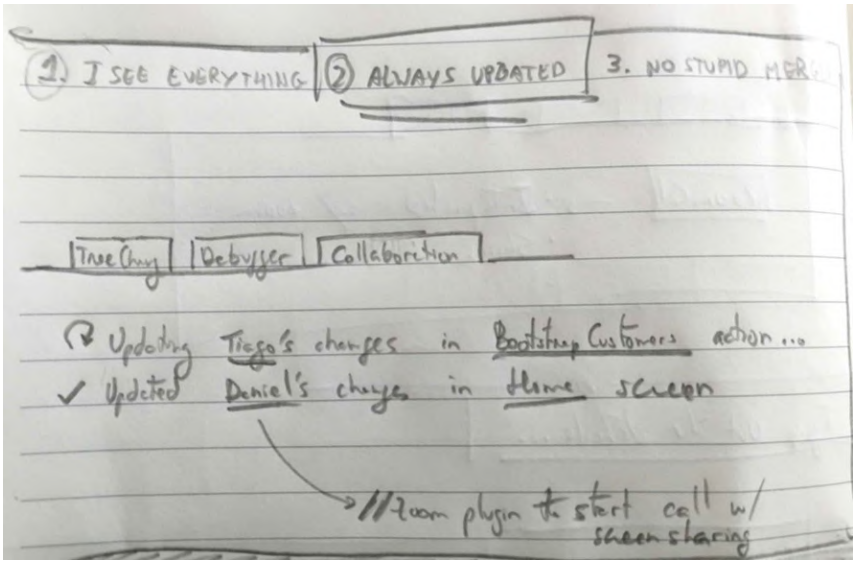



Figure 4.8: Sketch from Participant E regarding automatic updates

**"Improvements on the manual merge operation"** was a hot topic during the ideation session. The participants overall felt that the manual merge operation could be more clear to the developer in how the final merged version will be affected. Participant F and Participant D suggested removing the merge button all together from the Compare window and have a visual highlight on the differences to manually edit the module and merge the changes. (see Figure 4.9).

Although these are some interesting ideas, improving the manual merge operation is helpful for dealing better with conflicts, but it does not prevent them, which is our main goal. For that reason, ideas regarding improvements to the merge operation were not developed any further.

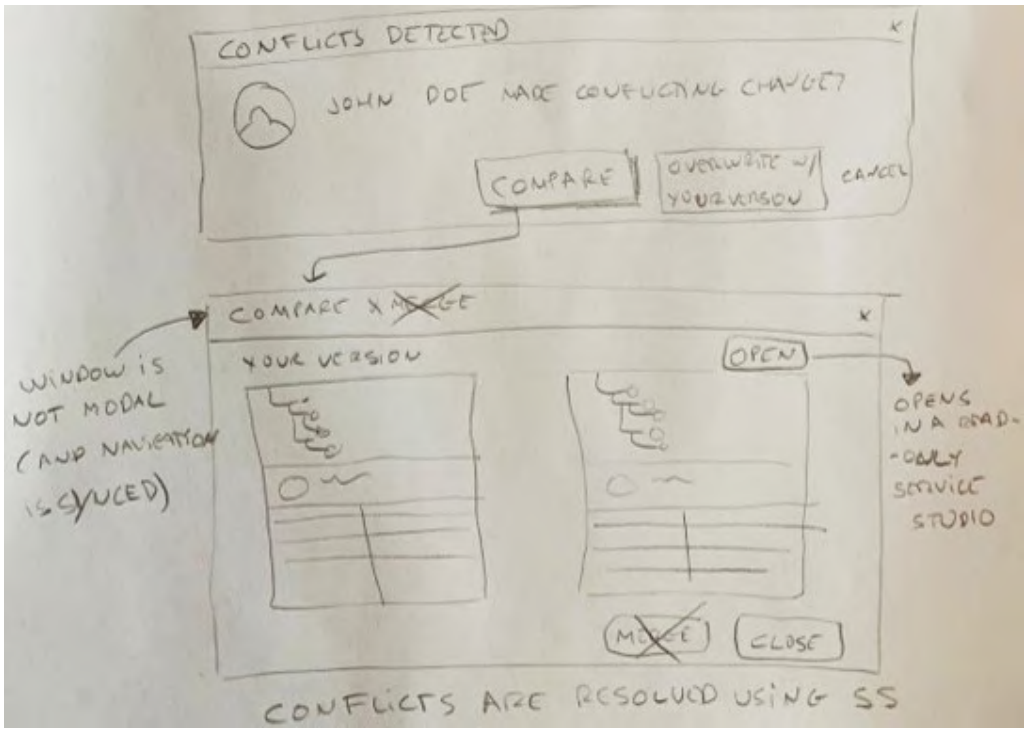
**"Improvements on the granularity of the merge operation"** was expected as a suggestion for solving collaboration issues, as it was reported during the interviews in the "emphasize" phase that many conflicts are detected by the platform because the merge operation is not granular enough to solve certain conflicts (see Section 4.3). This, however, is a feature already being developed and studied internally and that once again does not prevent conflicts, it only deals better with them. For that reason, it was also not taken further into testing.

**"Issue tracking"** is the last category of solutions presented in the ideation session, and it was brought up by Participant F. It is helpful for very specific use cases like code review, when a manager wants to track how an issue is being treated by the developers responsible for dealing with it. While considered a collaborative feature, it is related to management and it does not give a solution reducing a project's TTV or improving collaboration between developers. Therefore, it was decided that it is not a feature worth exploring.

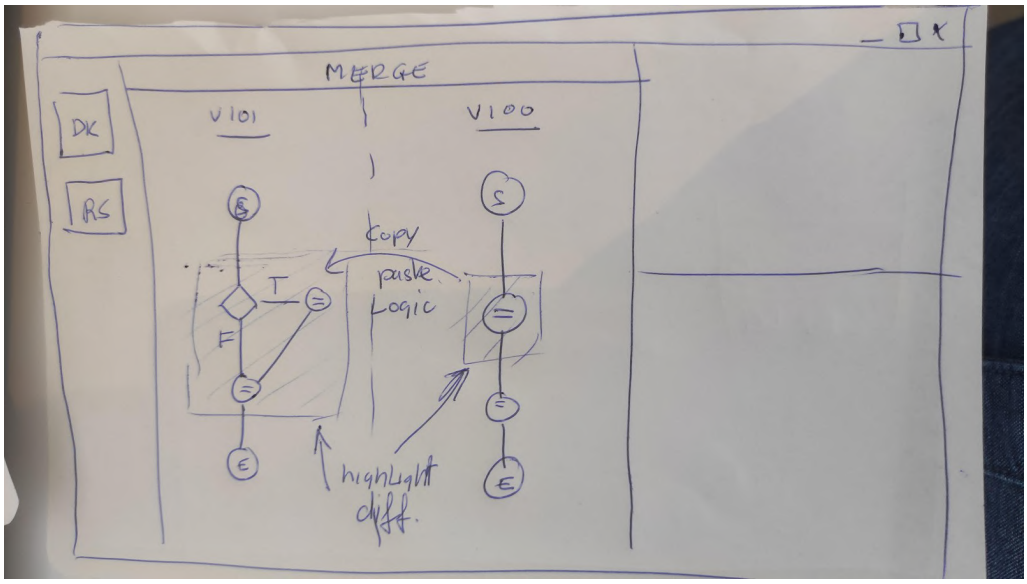
To summarize, the more promising ideas coming out of the ideation session were **"awareness of other people working in the workspace"**, **"awareness on changes being made by other developers"** and **"automatic updates to the local module"**. Together with our analysis on software development tools with collaborative features available in the market and scientific literature, we decided to move to the next stage and build prototypes of the mentioned solutions as a starting point for testing. Their respective prototypes were tested in the first iteration, as showed in Section 5.1. A "pure" and complete real-time collaboration feature (similar, for example, to Google Docs) was not initially considered, as it did not show much appreciation during the interviews and during our ideation session. However, due to user feedback, it ended up being tested in later iterations (see Subsection 5.3.1).

## 4.5 Prototype

Having generated several ideas and selected the best ones for testing, the next step is to build the models of the solution - the mockups. These mockups at first were intended for a usability test scenario where users would receive a task to complete and could interact




(a) Sketch from Participant D




(b) Sketch from Participant F

Figure 4.9: Sketches from Participant D and Participant F regarding the manual merge operation

with them to complete that task (e.g., they would have a mockup of the Service Studio application and would be tasked to determine who was online and start a call). However, we are testing features that involve many variables, and this kind of usability testing would require a way of simulating other developers simultaneously working in the same module to achieve a realistic experience. Such an experiment is not possible with the tools currently available for usability testing. We tried to devise an advanced test for this end, as it is a common practice in the area to experiment with new ways of testing. For example, in 1983 [81], IBM wanted to invest in speech recognition technology. Since it was an enormous investment, they decided to first understand if people could use and take advantage of such a feature. Therefore, they created an environment where test participants would talk to a computer and see their words written on a screen, when it was actually being written by a hidden human operator. This way, they could exactly reproduce the experience without having to implement it.

However, for our testing, we have moved away from the usability test environment (as it proved inconceivable for our scenario) and focus on concept testing (see Section 3.4.3), where we conceptually present the solutions to users with mockups, with some interactivity attached, to better illustrate the ideas. This way, users can have a better idea of the features and imagine themselves using them in their daily work, and then give us their feedback about them. We have used Figma [82], a web-based tool for building prototypes, to create the models that simulate the features we would like to test with users of OutSystems. It was chosen among several tools because the UX department of OutSystems uses it for this type of tests and, as such, we have access to an extensive library with many digital assets representing the Service Studio user interface. Thus, our work in building the prototypes can be accelerated through the use of these prebuilt components. These can show the Service Studio main window, design new sections, animations, overlays, among other things, to give the user an interactive and rich experience like the features were actually implemented (see Figure 4.10). The mockup elaboration was assisted by designers from OutSystems' UX department to evaluate the design and usability of the models before using them for testing.

Currently, Service Studio is undergoing a visual update to make its UI look more "modern", while retaining the same functionality. In July 2021, Service Studio for Mac was publicly released with this visual change. Windows users can also use the new Service Studio interface in the beta version. Although most people are still used to the "old" interface, it was decided to use the new one for the prototypes as to future-proof the research results (as the new interface can become the only version in the future). Both versions are similar in functionality and organization, and most changes are just in the look and feel (see Figure 4.11 for a side-by-side view of both UIs). For this reason, although developers have to adapt to the new interface for our tests, it is at a very small, almost negligible, cost. The decision to test with the new version was then made with the future in mind, as it is expected that eventually it will completely replace the previous one.

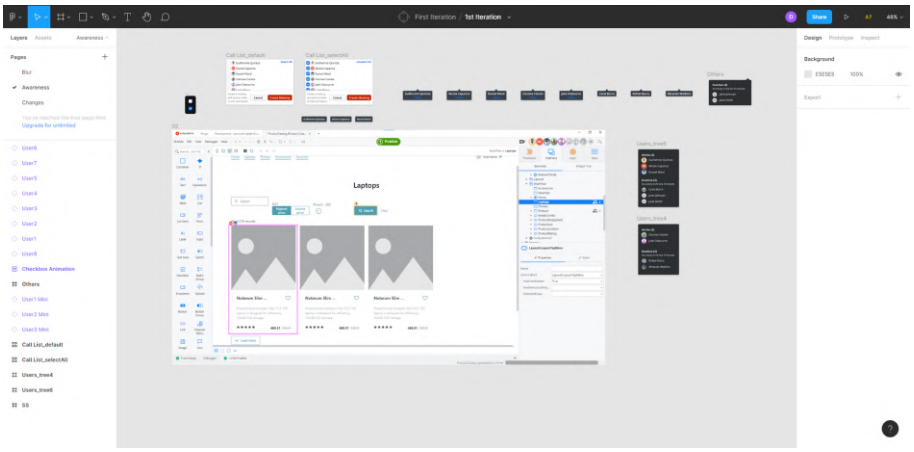
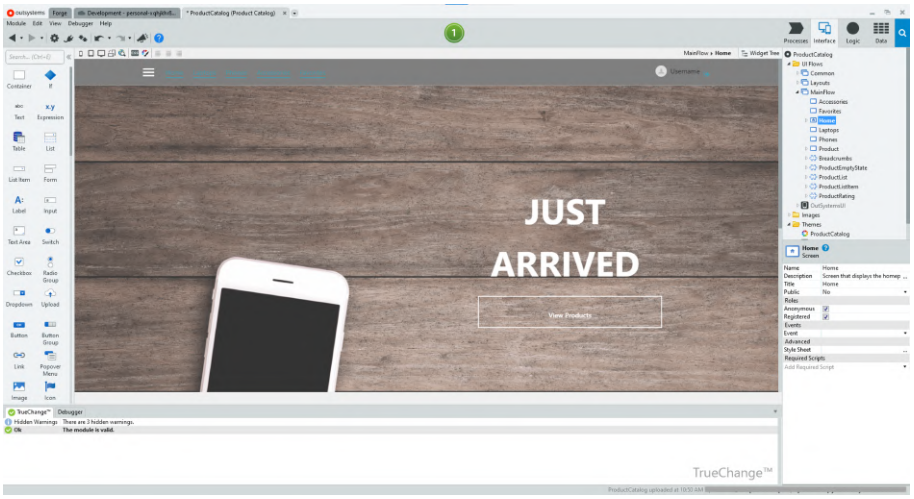
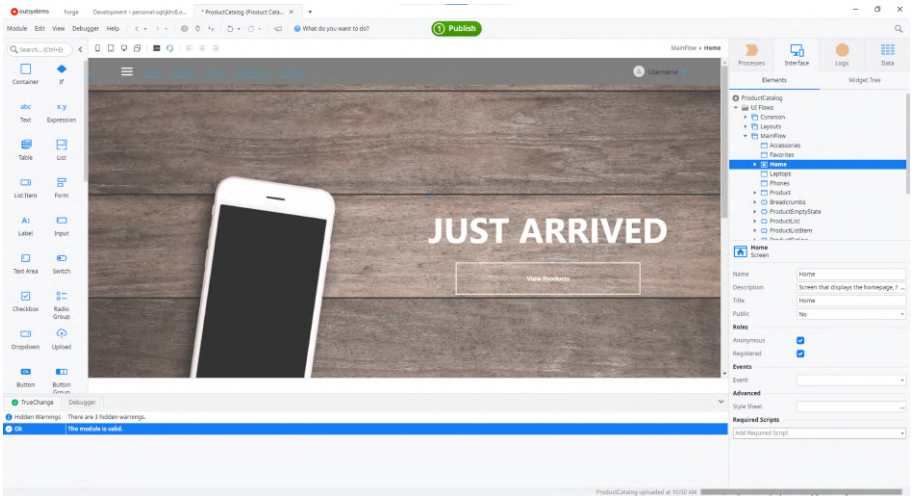



Figure 4.10: Example of a mockup being built with Figma




(a) The "old" Service Studio




(b) The "new" Service Studio

Figure 4.11: The two versions of the Service Studio UI

We have performed three test iterations, showing users three different prototypes in each iteration, making up nine prototypes in total. These are listed and described in Chapter 5.

## 4.6 Test

We have empathized with users to learn more about their concerns and issues with collaboration in OutSystems; we have combined all the information acquired to define the most troublesome problems affecting our users; we then moved our focus from the problem space to the solution space, and began to generate lots of different ideas for features to improve collaboration; finally, we grabbed the best ideas and prototyped them. The last phase of our research work is "test", which involves presenting our prototypes to users to get their thoughts and opinions regarding the potential of our ideas to solve the collaborative issues they currently face.

This is an iterative process, since it makes sense to continuously loop between the "ideate", "prototype" and "test" phases while users have something to add and suggest regarding our solutions. When a "test" phase ends with very little to no new information, and we have run out of options to test, we can conclude our research, presenting the results and discussing them. Figure 4.12 illustrates this strategy which deviates a little from the steps represented in Figure 4.2, but it remains valid due to the flexibility of the Design Thinking Methodology.

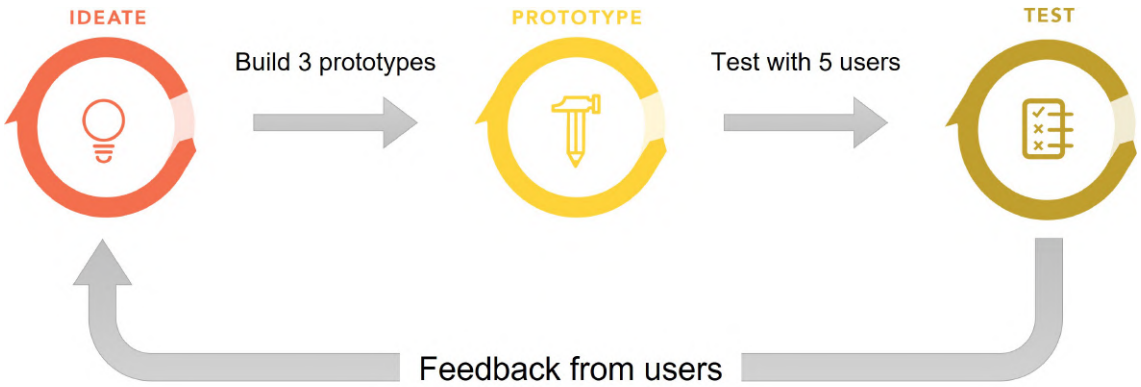



Figure 4.12: Diagram representing our test cycle. Once there is no significant feedback or options to explore, the research is over

As to avoid biases and get consistent results, we have structured our testing the same across all participants. They were selected as diversely as possible, based on age, gender, location, role, and experience. Because it can be very time-consuming and difficult to find test subjects, we have followed the UX guideline which states that five users is the optimum amount of participants for a test, since it is enough to discover most problems with our concepts [83]. A script was also redacted to serve as a guideline throughout each

test (which can be found in Annex I) introductory questions to create a profile and get some opinions before showing the concepts.

Following the principals of concept testing (described in Subsection 3.4.3), Protomonadic testing was the method chosen for our tests. It combines the benefits of Monadic testing, allowing users to openly share their thoughts about the concepts presented to them, with Comparative testing, which weights different concepts against each other. We showed three prototypes for each iteration of our test, so test participants would not get overloaded and confused with too many concepts. These were not shown in the same order to every user, thus removing the "order bias" from our testing, where the concept order can influence users' opinions. What this means is that we had three prototypes, A, B, and C, and would, for example, show them in the order ABC to some users and CBA to other users.

The prototypes were shared using a share-screen functionality on the interviewer's computer with the user having a remote control access, where he could freely move the cursor and interact with the prototype through mouse click actions. No context was provided to the user so we could realize if the prototype was self-explanatory and easy to use. During the prototype exploration process, the user was encouraged to share his thoughts out loud. When the user had concluded his analysis, the interviewer would then explain the concept and ask specific questions regarding some of its components. After analyzing all three prototypes, the user was questioned about the one they would like the most to see implemented in Service Studio.

Concept testing produces a lot of qualitative results coming from the users' thoughts and opinions. To also have some quantitative data to confirm the insights received in the testing, test subjects were asked to fill in a survey at the end with simple questions to be answered using a rating scale (a Likert Scale). It should be mentioned that each iteration was done with five users, so there are only five survey results per iteration. Although five users is enough for qualitative testing, it does not produce accurate results for quantitative data. For this reason, this data is more of a complement to the qualitative data and conclusions should not be taken solely from it. The survey and its results can be found in Annex II.

# Chapter 5

# Results

In this chapter, the results from the three iterations of our "test" phase of the Design Thinking methodology are presented. There is an in-depth analysis of the functionality and feedback received from the concept prototypes presented to the users. The test participants are also described at the beginning of each test iteration regarding the following attributes (valid at the time of testing): role, employer (whether they work at OutSystems or another external company), years of experience in developing with OutSystems and the size of project teams using OutSystems they have taken part in. This last attribute is meant to understand whether there is a connection between team size and collaboration issues.

As it was mentioned before, only five users participated in each test iteration. The goal was to collect qualitative data to understand and improve the collaborative experience of the prototyped concepts. The quantitative data are not so meaningful since five users is not a large enough population to take conclusions from those metrics. As such, the user opinions from each prototyped are not quantified in this chapter, usually referring only to the trend among the five users (using expressions like "most users liked this concept", avoiding quantifying them). However, in Annex II, it is possible to analyze the surveys filled by these users and its results. Once again, these results based on quantitative data should not be used to take any conclusions.

**DISCLAIMER**: The example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious. No association with any real company, organization, product, domain name, e-mail address, logo, person, places, or events is intended or should be inferred.

## 5.1 First Test Iteration

As mentioned in Subsection 4.4.1, the most promising ideas generated in the ideation session we organized were **"awareness of other people working in the workspace"**, **"awareness on changes being made by other developers"** and **"automatic updates to the local module"**. As such, and not to overload our test participants with too many concepts, we created three prototypes for our first test iteration based on these solutions. Test participants from this iteration had the attributes described in Table 5.1, at the time of testing. The individual test sessions took place remotely through the use of video conferencing software between June 4, 2021, and June 16, 2021.

Table 5.1: Test participants for the first test iteration

| User | Role | Employer | Experience with OutSystems | | |
|---|---|---|---|---|---|
| | | | Time | Team size | |
| | | | | Minimum | Maximum |
| A | Developer | OutSystems | 2 years | 3 people | 8 people |
| B | Developer | OutSystems | 3 years | 3 people | 6 people |
| C | Advanced Development Expert Lead | OutSystems | 10 years | Alone | 20 people |
| D | Lead Architect | External | 10 years | Alone | 12 people |
| E | Front-end & Mobile | OutSystems | 5 years | Alone | 20 people |

### 5.1.1 User Awareness Concept Prototype

Test participants were shown the prototype in Figure 5.1. In terms of added features, each number shown in the figure represents the following:

1. **Awareness Inside an Element (e.g., a Screen or Action)** - If there are other developers working in that same element, a highlight is shown with the developer's icon and color (assigned to them by Service Studio) in the area they have selected and in which they are actively making changes to. The purpose of this feature is to give developers awareness on what other developers are working on to prevent any conflicts from happening.

2. **Collaboration Bar** - People with access to the project will appear in this top bar with a name, icon (optional) and color associated to them. This is the way they will be represented within Service Studio. Figure 5.2 demonstrates the features available with this bar. Users can follow other users, which is to say that if, for example, I choose to follow "Guilherme Quintas", my Service Studio will jump from element to element according to his actions. There is also a feature which

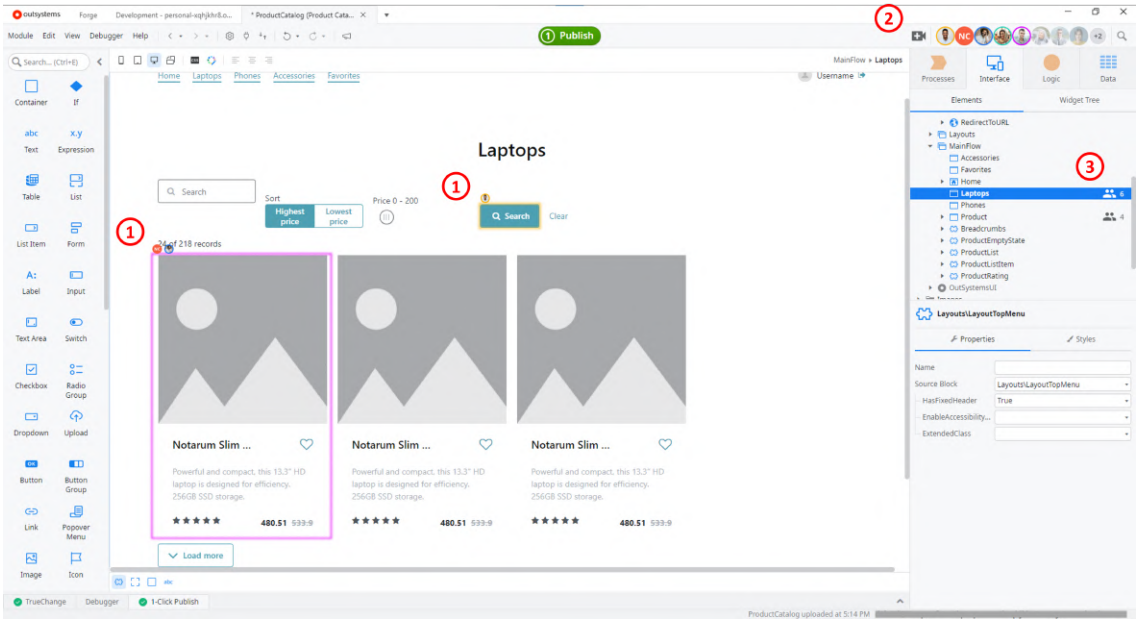


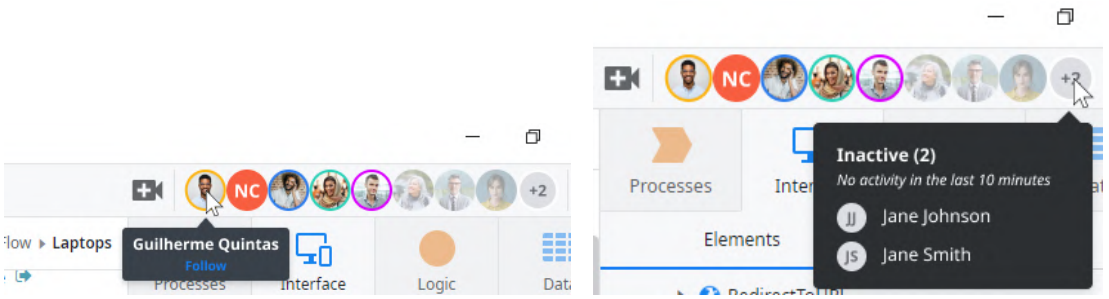

Figure 5.1: User awareness concept prototype

makes user icons more translucid according to their activity, so if they have not been changing anything in the module for more than ten minutes, their status changes to "inactive". Calls are integrated so within Service Studio it is possible to create a meeting with your peers (implementation details are irrelevant in this phase, but this could be configured to use a third-party service for the call).

3. **Module Awareness** - An icon appears next to a module element if there are other people with it open in their Service Studio. As illustrated by Figure 5.3, when hovered, it shows who is in that element and there is a distinction between who is actively editing it and who has not made a change in more than ten minutes (could be viewing it or not in front of the computer, for example).

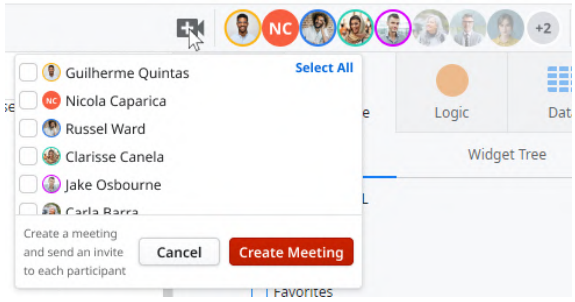
Overall, the test participants were very enthusiastic by the possibility to have a high-level awareness of where their peers are working within the module, given by feature number three. This would allow them to avoid conflicts as they would immediately contact their colleague to understand what they are doing in a certain element and coordinate their activities. Feature number one was seen as not very useful by users A, B and D as it does not show the changes happening, only the area where they are happening. This gives users the same value in preventing conflicts as feature number three, since they would always want to know what exactly is being changed within that element. User E raised some concerns regarding the possibility of the highlights of feature number one blending with the UI. The ability to follow other users around the module was seen as relevant by all users for when they are giving assistance to a colleague, but there were some concerns regarding privacy, as User A stated, "I am afraid someone from management might use

the follow feature to control what I am doing".




(a) Hovering a user's icon

(b) Distinction between active and inactive users




(c) Ability to easily create a meeting with other users

Figure 5.2: Collaboration Bar features

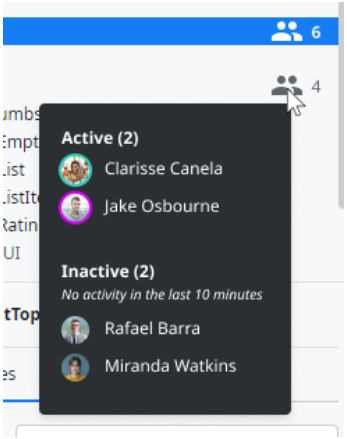



Figure 5.3: Detail of awareness of users in other elements of the module

For the next iteration, the "inactive" information should be removed since, as User C puts it regarding the module awareness menu (see Figure 5.3), "if they are not actively changing the element, I do not even need to know they are there". The higher-level awareness was really well received and should be further explored and the follow feature needs more testing to see how helpful it can really be for our objective of preventing conflicts. The call feature needs no further testing since we can conclude that it is a "nice to have" but it does not create any value for our main goal of reducing conflicts. The follow feature makes sense to keep for the next prototype, as users thought it would be

useful in a real-life situation of giving assistance to a colleague. However, the privacy issues raised by User A will be taken into account to come up with a more private way of using this feature for the next prototype iteration.

### 5.1.2 Visual Awareness of Conflicting Changes Concept Prototype

Test participants were shown the prototype in Figure 5.4. In terms of added features, each number shown in the figure represents the following:

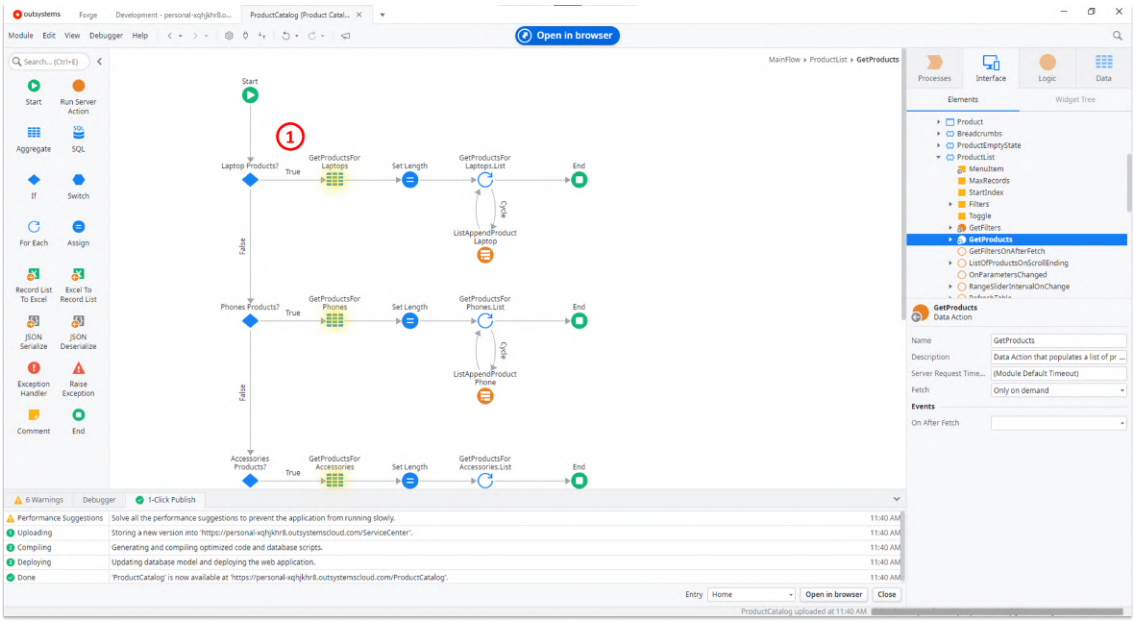



Figure 5.4: Visual awareness of conflicting changes concept prototype

1. **Highlight on Conflicts** - The purpose of this prototype is to illustrate the possibility of a visual hint (in this example, a yellow blur) appearing on sections within an element which where changed by the user but also by other developers in their respective instances of Service Studio, even if all the users involved did not publish those changes. This way, they can all immediately know that they are changing the same things and coordinate their work before publishing. This way, the conflict is avoided and not allowed to escalate in complexity. Figure 5.5 shows how right-clicking the yellow visual hint can give the developer more information about the situation, particularly the collaborators with conflicting changes.

Perhaps due to the simplicity of this prototype, users had many suggestions of how it could be further developed to bring more value to them. Improvements to the highlight could be made to show more information about the conflict (e.g., if the component was deleted or just changed, as suggested by User D; the where/what/who/when of the conflict, as User B suggested) and have a Compare & Merge window open to immediately decide if we want our changes or someone else's changes to be incorporated (suggested by User B and E). However, most users did not see this feature as very useful since a higher-level awareness, such as the awareness that there are changes being made in that element, would be enough because, as User A stated, "Just knowing that another developer is changing that same Screen or Action, I would immediately open both versions

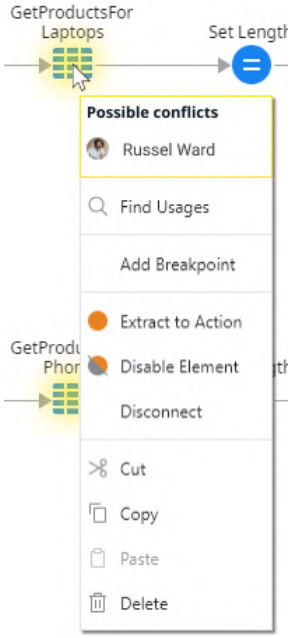


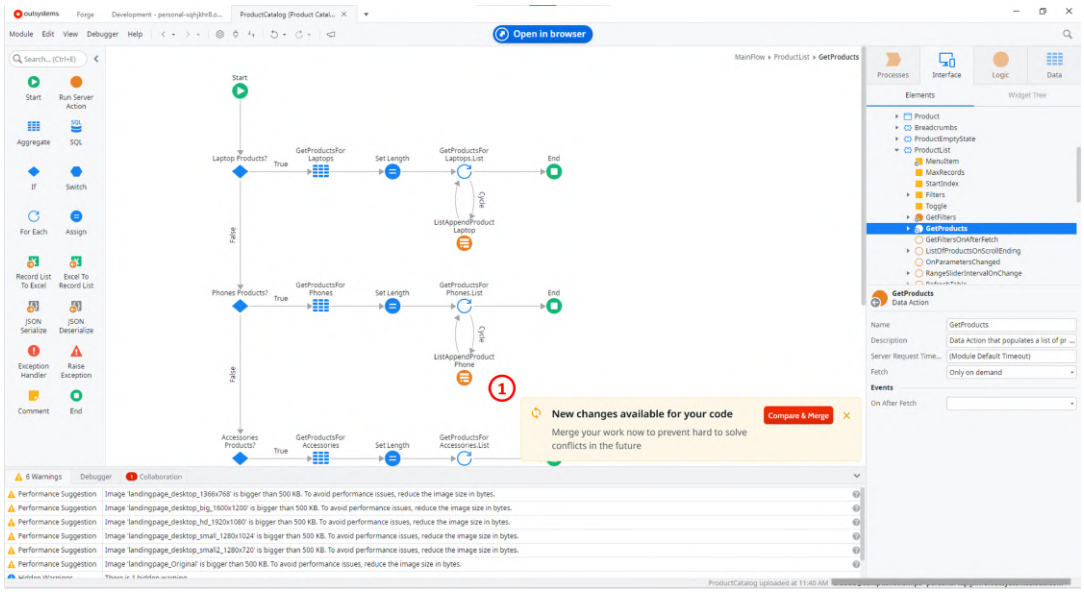

Figure 5.5: A right-click on the yellow blur marking a potential conflict with the "GetProductsForLaptops" node

side-by-side and call him to know what is going on". Developers explained that this is due to the fact that when merging their work, the platform does not have enough granularity to compare different components of the same element and realize that is not conflict (also explained in Subsection 4.2.1). As such, knowing exactly where the conflict is and ignoring it if not part of their work's scope would not avoid the conflict within the platform. We asked all users for their opinion considering a future where the merge process would be more granular and able to distinguish between these small changes, but no one's opinion changed. Other suggestions included having a way to subscribe to a notification on changes as they happen in some screens/actions of our choice (suggested by User E) and having a centralized place where all changes could be checked as they happen by all developers (suggested by User D).

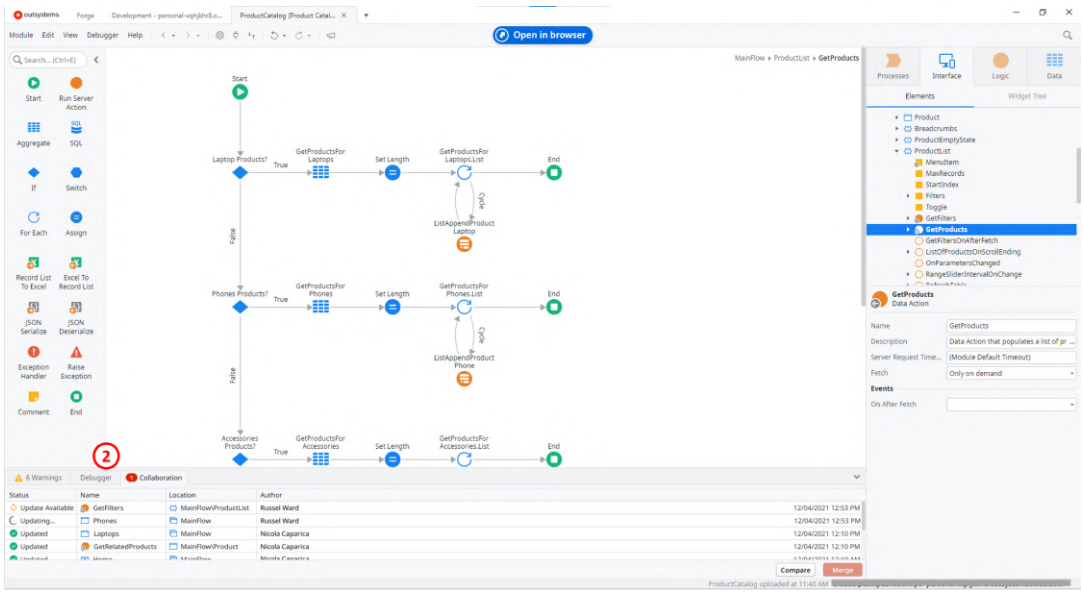
For the next iteration, we decided to drop this highlight concept, since it did not get a lot of appraise, and focus on testing the ideas regarding the centralized reporting of changes and the subscription to notifications when changes happen in certain actions/screens.

### 5.1.3 Automatic and Manual Updates with a Notification Concept Prototype

Test participants were shown the prototype in Figure 5.6. In terms of added features, each number shown in the figure represents the following:




(a) A notification pop-up appears when there are new conflicting changes published by another developer




(b) There is a collaboration tab to check on all automatic and manual updates applied to the module

Figure 5.6: Automatic and manual updates with a notification concept prototype

1. **Notification for New Published and Conflicting Changes** - Considering a scenario where a developer is working in a certain element of the module; he would get a notification whenever some other developer publishes new changes to the same

elements that he is working on. Hence, the developer is notified immediately when a conflict happens and is prompted to check and solve it.

2. **Automatic Updates and Log of All Changes Happening** - The main idea of this feature is to always have the most updated version of the module. Whenever changes are published, Service Studio will automatically update the local module with them. If there is a conflict and the user needs to manually merge them, the notification pop-up, mentioned in the previous topic, appears to warn the user, and they have to manually update it with a merge operation.

This concept created a lot of discomfort among all test participants due to the "silent changes" that happen without the user's awareness. During a task, they might be working in a certain element while assuming that other elements are in a state A when, in fact, they have silently been updated to a state B, causing even more confusion. Even though there is a collaboration tab showing the log of all changes, users could not find a use case in which this feature would be a plus. Regarding the notification feature, it was perceived as nice to have for bringing real-time awareness to what is being updated in the module. However, users demonstrated some concerns for it being "too big and intrusive" and User B even suggested the ability for it to be "opt-in". Having users desire a feature to be optional could be interpreted as a sign that it is not something they will use if it ever becomes available.

Considering the feedback received throughout testing, we decided to drop this idea of having automatic updates being done to the module since users did not feel comfortable using it. The notification feature will be reworked and tested in the next iteration, to realize if making it more discrete can have users find it more useful and pleasing to use.

## 5.2 Second Test Iteration

User feedback from the previous iteration determined that having automatic updates to the module and a visual highlight on components with concurrent changes was not the best way to improve collaboration. As such, for this iteration we decided to improve the first prototype based on what users disliked and suggested regarding user awareness in Service Studio. The second prototype tests the concept of having a more custom and discrete notification feature while also testing a new collaborative concept where users can have exclusive rights to edit an element. Finally, the third prototype takes the concept from the "Automatic and Manual Updates with a Notification Concept Prototype" of having a centralized place where users can view all changes that happened to the module and act upon them.

Test participants from this iteration had the attributes described in Table 5.2, at the time of testing. The individual test sessions took place remotely through the use of video conferencing software between June 23, 2021, and June 25, 2021.

Table 5.2: Test participants for the second test iteration

| User | Role | Employer | Experience with OutSystems | | |
|---|---|---|---|---|---|
| | | | Time | Team size | |
| | | | | Minimum | Maximum |
| F | Developer | OutSystems | 3 years | 3 people | 4 people |
| G | Tech Lead | External | 8 years | 3 people | 18 people |
| H | Developer | OutSystems | 3 years | 3 people | 40 people |
| I | Developer | OutSystems | 4 years | Alone | 6 people |
| J | Manager | External | 5 years | 3 people | 6 people |

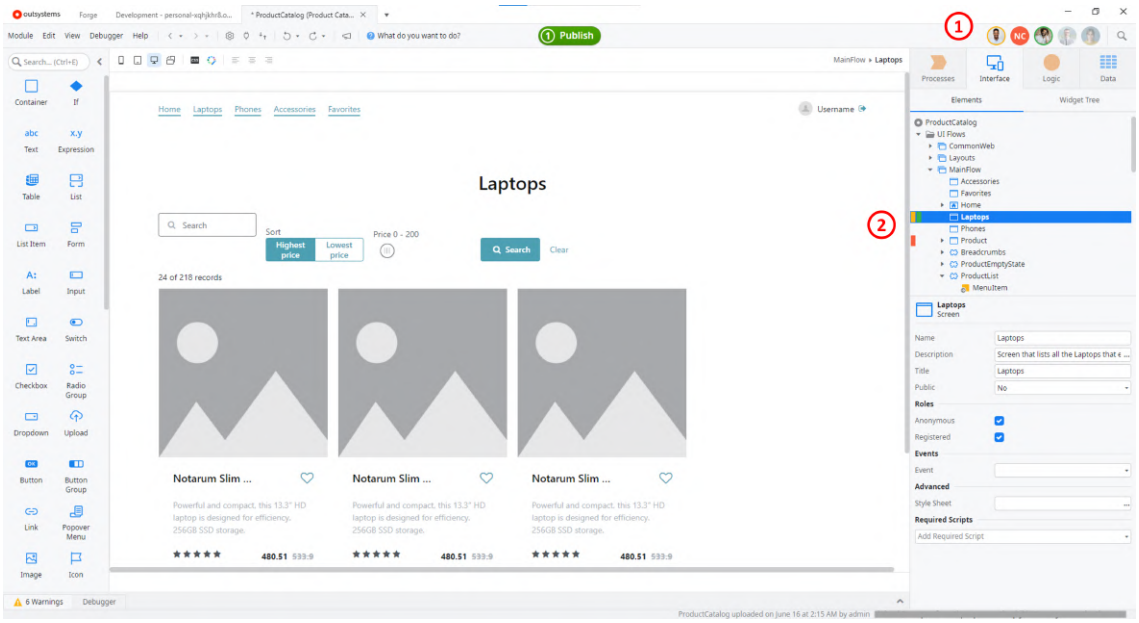
### 5.2.1 User Awareness and Follow Concept Prototype

Test participants were shown the prototype in Figure 5.7. In terms of added features, each number shown in the figure represents the following:

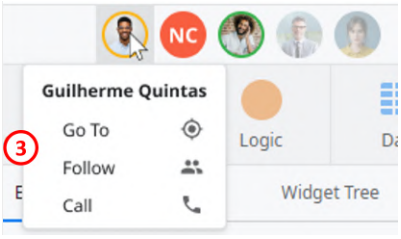
1. **Collaboration Bar** - Like in previous prototypes, it is important to have a representation of other collaborators within Service Studio to give a better sense of awareness to the user. For this concept, we maintained the icon, color and name identifying each user and added three options (see Figure 5.7 (b)): "Go to", which opens the element in which the user is working in their Service Studio; "follow", allows you to see what another user has opened in Service Studio (if given permission) and jumps to the same screens/actions as they open them; "call", to directly call that user (which could be integrated within Service Studio or use a third-party application). Offline users (meaning they do not have the respective module opened in Service Studio and are as such are not working on it) do not have any of these options, or color assigned, and their icon is more translucid. The idea behind displaying the icons of users who are offline in the collaboration bar is to bring awareness regarding who has access to the module.

2. **Module Awareness** - Same feature as in the user awareness prototype of the previous iteration (see Subsection 5.1.1), but without the explicit "active" and "inactive" distinction and with a visual overhaul. Here, active users have a bar with their color next to the element they are working in.

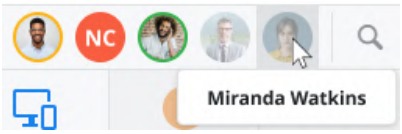
3. **Follow Feature** - When selected on a user, the follow feature now presents a rectangle around the workspace with the color of the user being followed (see Figure 5.8). It is also possible to see their cursor and selection. If they go to another element of the module, the follower's Service Studio will also switch to that element. Since this is not a concept where users can see changes happening in real-time, any changes




(a) Compared to the user awareness concept of the previous iteration, the Collaboration Bar and module awareness of other developers was kept




(b) Options available when hovering a user's icon




(c) When offline, users have no color assigned, their icon is more translucid, and there are no options available

Figure 5.7: User awareness and follow concept prototype

made to the module are not reflected in the follower's screen. There also is a "go to" feature that opens the Screen or Action where the respective user is in.

This prototype created a lot of enthusiasm among the test participants. As User G stated, "The possibility to know were each developer is working is something that can really help in big projects with a lot of people always working on the same modules, all the visual signs in the concept can help people to be careful before publishing, preventing code being lost/overridden". Therefore, this idea seemed to have a lot of potential in reducing conflicts among developers. The "follow" concept was not seen as very relevant by all users, except for User J, being perhaps more relevant for use cases involving senior management. The problem was that users could not think of any use to a follow feature where they cannot see the changes happening in real-time. For User J, it was still useful but only for code reviewing. However, everyone suggested taking the "follow" feature a step further by having a toggle for real-time collaboration to enable pair programming and to visualize live changes from other users happening to the module. The call feature

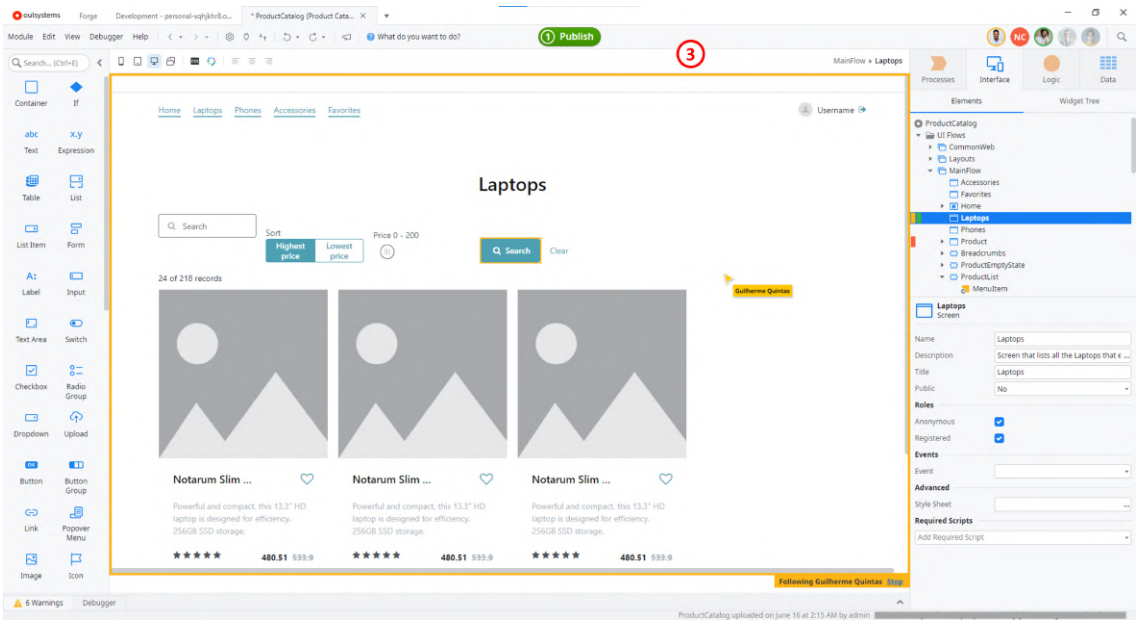



Figure 5.8: The Service Studio UI when following a user in the User Awareness and Follow Concept prototype

received mix feelings, it is considered useful but it does not add any value when you can just as easily call your peers using the project communication channels (third-party applications).

For the next iteration, it makes sense to give some final touches to the user awareness concept based on the feedback received. "Go to" was considered useful and a nice complement to the awareness experience. The call feature will be dropped as it does not add any more value. The "follow" feature does not seem useful for preventing conflicts but, however, it now makes sense to test a real-time collaboration experience since many test participants suggested several features that relate to it.

### 5.2.2 Exclusive Rights to Module Elements and Subscribe to Changes Concept Prototype

For this prototype, we decided to test a more disruptive feature. Instead of having people coordinating their concurrent work with each other, we wanted to test a concept where there is a lock mechanism that only allows one person at a time to edit a certain element of the module. This way, if someone is editing Screen A, no one else can simultaneously edit it.

Test participants were shown the prototype in Figure 5.9. In terms of added features, each number shown in the figure represents the following:

1. **Exclusive Rights to Module Elements** - When opening an element, the user will see a top banner if someone else has the lock (the exclusive rights) for that element.

Whoever "holds the lock" is the only person able to make any changes to that element. Faced with the top banner meaning they cannot edit the element, a developer then has several options: they can call or talk to the person editing the screen (who holds the lock) to figure out what is happening; they can open a "Compare and Merge" window and see what changed and even update their local version with those changes; "take over", which means that the lock is removed from the user who holds it and given to the user who requested it, making it then the only person able to edit the element; or simply wait until the lock is released by the developer who is holding it.

2. **Subscription to Changes** - For this iteration, we enhanced the notification concept by having the possibility to explicitly choose which screens/actions we would to receive a notification when changes happen (by clicking the bell icon next to the element's name). Whenever a new version is published with changes to that same element, a notification appears in the corner of Service Studio with information regarding the element and the author of those changes.

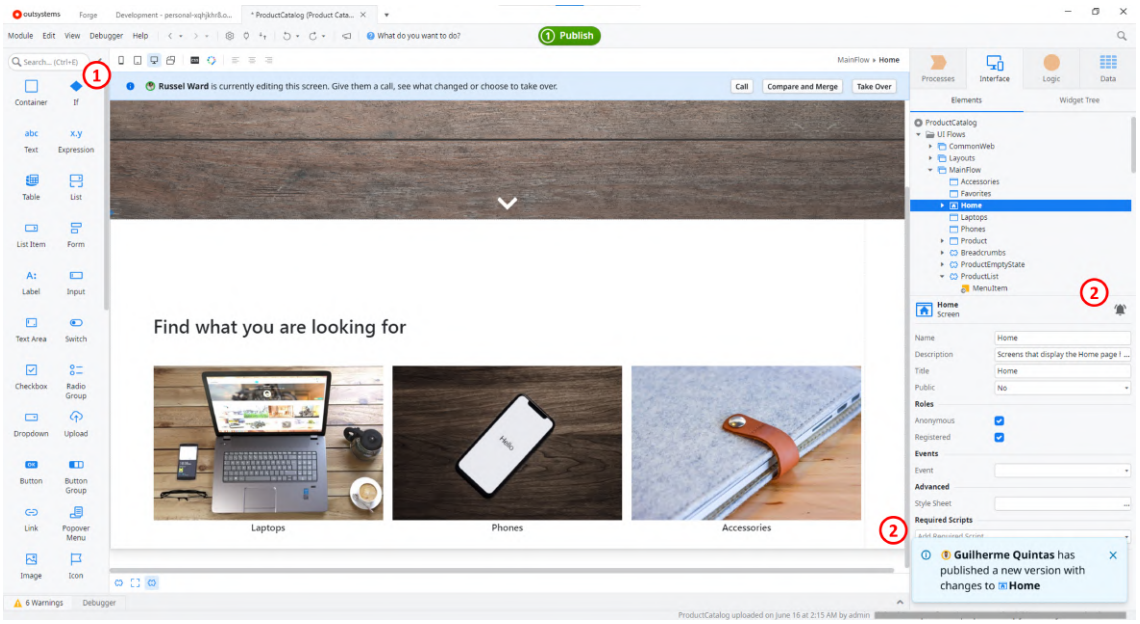



Figure 5.9: Exclusive rights to module elements and subscribe to changes concept prototype

The lock mechanism only received negative feedback since it is too limiting and blocking, as User G stated "it does not promote collaboration, it is anti-collaboration". However, the top blue banner received a lot of appraise and users reported they would very much enjoy if it could be used to indicate changes being made or previously done to that element and prompt the user to communicate with the author of the changes, compare them and possibly have a way of quickly merging. Regarding the subscription to changes, users thought it was an interesting feature, but no user could find a real use case for it

and all users except for User J expressed their discomfort in seeing it being used by senior management to control what developers are doing.

Because the exclusive rights to module elements feature was not well received, but users very much appreciated having a top banner to give them awareness if someone else is editing that same element, the banner will be used to provide module awareness in the next prototype iteration. Regarding the subscribe function, it was dropped as it did not seem to be aiding in the task of reducing conflicts between collaborators.

### 5.2.3 Collaboration Tab with Version History Concept Prototype

Test participants were shown the prototype in Figure 5.10. In terms of added features, each number shown in the figure represents the following:

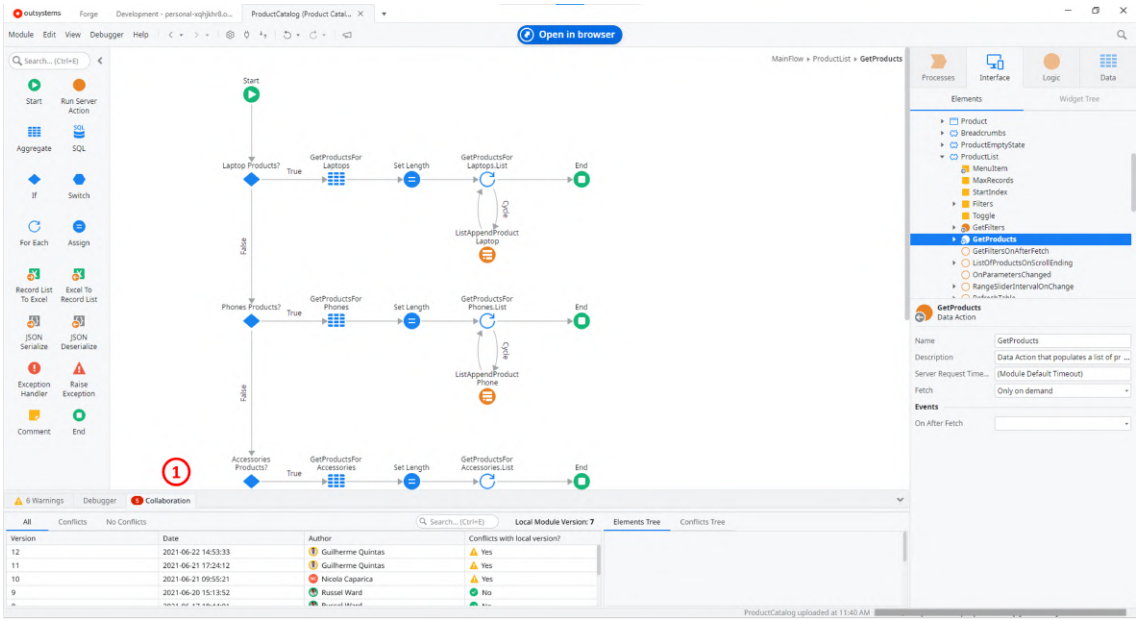



Figure 5.10: Collaboration tab with version history concept prototype

1. **Collaboration Tab with Version History** - A centralized place easily accessible to the user with the history of all published versions of the module, alongside information regarding date, author and if it conflicts with the current local version. It is also possible to filter the publishes according to the existence of conflicts and to quickly compare your local version with any other version and merge them.

This concept was not seen as very useful by users, except for User B and J who see it more relevant for senior management use cases, like reviewing the work of their team members. It does not actually aid collaboration in any way, looking more like a VCS functionality. "It does not prevent conflicts, it only reports them better", as User H put it. User F pointed out two situations where there can be a lot of publishes happening, depending on the number of developers and the project's development stage or even if

something is not working and there are publishes being made all the time to test the application. Hence, this concept could be too overwhelming as users might not be able to keep up with the updates being listed. User H and I thought that a simple notification stating that there are new updates to certain screens or actions would be a much more useful and pleasant feature.

Since this concept did not seem to help users with collaboration in any way and there does not seem to be much space to improve, we have disregarded the version history concept for the next iterations but developed on the suggestion regarding a notification on the actions and screens with changes made by other developers (see Subsection 5.13).

## 5.3 Third Test Iteration

In this iteration, which ended up being the last one due to no new ideas to test, a real-time collaboration concept was prototyped since users suggested such a feature throughout the previous test iterations. A more final version of the user awareness concept is tested, taking advantage of the blue banner bar from the "Exclusive Rights to Module Elements and Subscribe to Changes Concept Prototype" which in here is adapted to give users awareness on changes made to the module elements. Finally, the Visual Awareness of Conflicting Changes Concept Prototype tested in the first iteration (see Subsection 5.1.2), was further improved in an effort to confirm if adding some improvements to it could make it more appealing to users.

Test participants from this iteration had the attributes described in Table 5.3, at the time of testing. The individual test sessions took place remotely through the use of video conferencing software between July 1, 2021, and July 2, 2021.

Table 5.3: Test participants for the third test iteration

| User | Role | Employer | Experience with OutSystems | | |
|---|---|---|---|---|---|
| | | | Time | Team size | |
| | | | | Minimum | Maximum |
| K | Developer | OutSystems | 3 years | 3 people | 25 people |
| L | Front-end & Mobile | OutSystems | 7 years | 3 people | 10 people |
| M | Tech Lead | OutSystems | 5 years | 3 people | 25 people |
| N | Developer | External | 3 years | 3 people | 8 people |
| O | Developer | OutSystems | 5 years | 3 people | 25 people |

### 5.3.1 Real-time Collaboration Concept Prototype

During testing, we realized that it became reasonable to test live collaboration with users as a solution for promoting collaboration and preventing conflicts when developing applications in OutSystems. We decided not to test it at first, to allow for other diverse ideas to be tested. However, and without any bias from us, users along the test sessions suggested collaborative features that were actually related with real-time collaboration. As such, it made perfect sense to test this concept. It has some similarities to the real-time collaboration solution implemented in a previous research work done in OutSystems[24] but, unlike it, our prototype has been designed without considering implementation limitations, and it is based on the feedback received from users in previous test iterations.

Test participants were shown the prototype in Figure 5.11 (a) and were asked, without any further context, to create a collaborative session with their colleagues. In terms of added features, each number shown in Figures 5.11 and 5.12 represent the following:

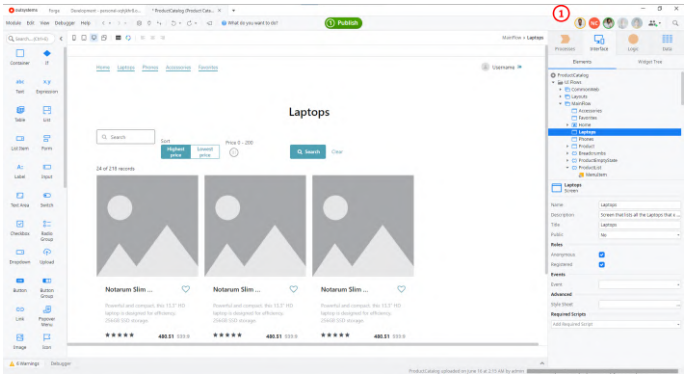
1. **Collaboration Bar** - As with previous concepts, there is a Collaboration Bar where call collaborators of the project are represented by their icon, name and color. Added to this prototype is a button which enables the creation of a real-time collaborative session with invites sent to the selected team members (Figure 5.11 (b))

2. **Real-time Collaboration** - Here users have the chance to collaborate live with other developers. The idea is to have developers working in the module with awareness of each other's movements and actions. Our hope is that this will promote collaboration by allowing users to better coordinate their actions and promote pair programming. As such, in this prototype and unlike all other, changes made do the module would immediately be reflected to everyone participating in the session, as they are all sharing the same version of the module. Figure 5.11 (c) demonstrates the loading screen shown to users while the real-time collaborative session is being created.

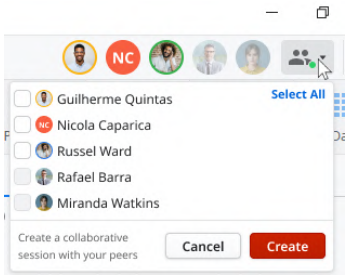
3. **Module Awareness** - If two people do not have the same screen open, module awareness allows them to see in which Screen or Action they are in. This way, they get a sense of where others are within the module.

4. **Element Awareness** - To simulate a physical context, where people can see each other and what they are doing, users can see the cursor (and name and color) of who is working on the same Screen or Action as them. There is also a colored box around the component other users have selected. This complements the fact that everyone can see the changes happening to the module elements as they are being done.

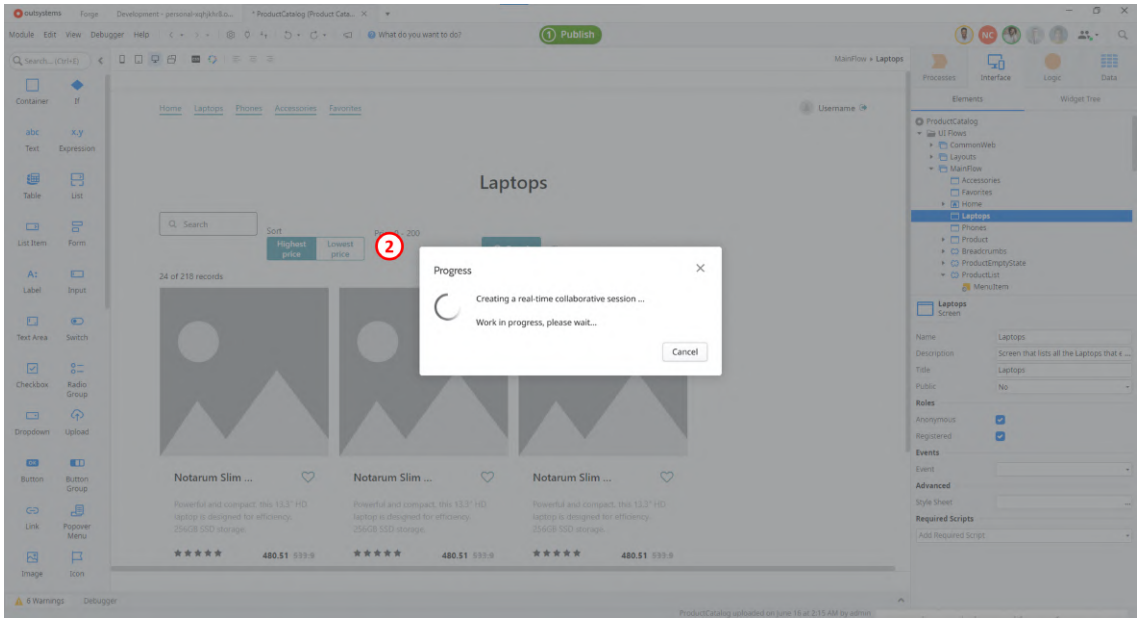
5. **Component Awareness** - In OutSystems, screens can be complex with many components within them. For this reason, a user might open a screen where a Developer




(a) As with previous concepts, there is a Collaboration Bar but with an extra button for creating a real-time collaboration session




(b) Selected users will receive an invitation to join the session



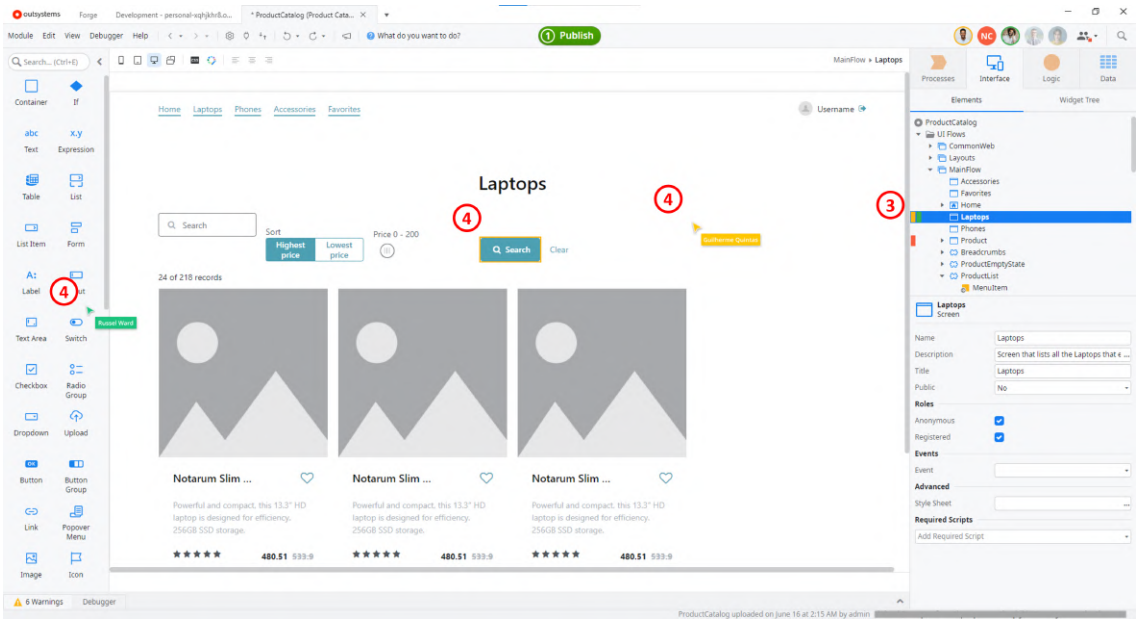

(c) Loading screen while the session is being created

Figure 5.11: Setting up a real-time collaboration session

A is working in and not see their cursor because they are actually working in a component within that screen. To signal that situation, a colored box is shown around the component to demonstrate there is someone "inside" it.

6. **Follow Feature** - A complement to the experience, the follow and "go to" feature have the same functionality as explained for the User Awareness and Follow Concept (see Subsection 5.2.1).

All test participants enjoyed the concept as it was seen as a solution for some current problems, especially at the beginning of projects where conflicts are harder to avoid, and with potential to change the way things are done (promoting more collaboration, exchange of ideas, and discussion of solutions) in a remote work reality. All of them also thought that real-time collaboration is something that does not make sense to happen all




(a) When users join the session, their cursor and selection is displayed to all participants with the same element opened, as well as awareness on their location in the module

(b) If a user is currently "inside" a component of the screen, that information is transmitted to everyone with that screen open

(c) "Go to" opens the element where the respective user is in, and the Follow feature follows the user around the module as he "jumps" from one element to the other

Figure 5.12: Real-time collaboration concept prototypes

the time, so being able to create a session when required was well received. The "go to" and Follow feature were seen as a plus by users L and N. User O specifically described real-time collaboration as "an ideal match for accelerated development of applications in which we only have two weeks to build an app". In these situations, developers end up working in the same places of the module due to the pressure of time, which leads to problems, conflicts, stressful situations and delays.

The following use cases were mentioned by users when asked about situations when this real-time collaboration concept could be useful: pair programming, code review, workshops, coaching, assisting a colleague, the beginning of projects when screens/actions are still being created, brainstorming, and collaborative bug fixing. Because users made no suggestions to improve the concept, and we could not think of any meaningful

changes to test the prototype, we decided that we had tested everything we needed to regarding this concept.

### 5.3.2 User Awareness Final Concept Prototype

Awareness of other users in the module is a main concern of our research work. For this iteration, we included a compilation of the tested features that received positive feedback and added some more, now with a more final concept regarding user awareness.

Test participants were shown the prototype in Figure 5.13. In terms of added features, each number shown in the figure represents the following:

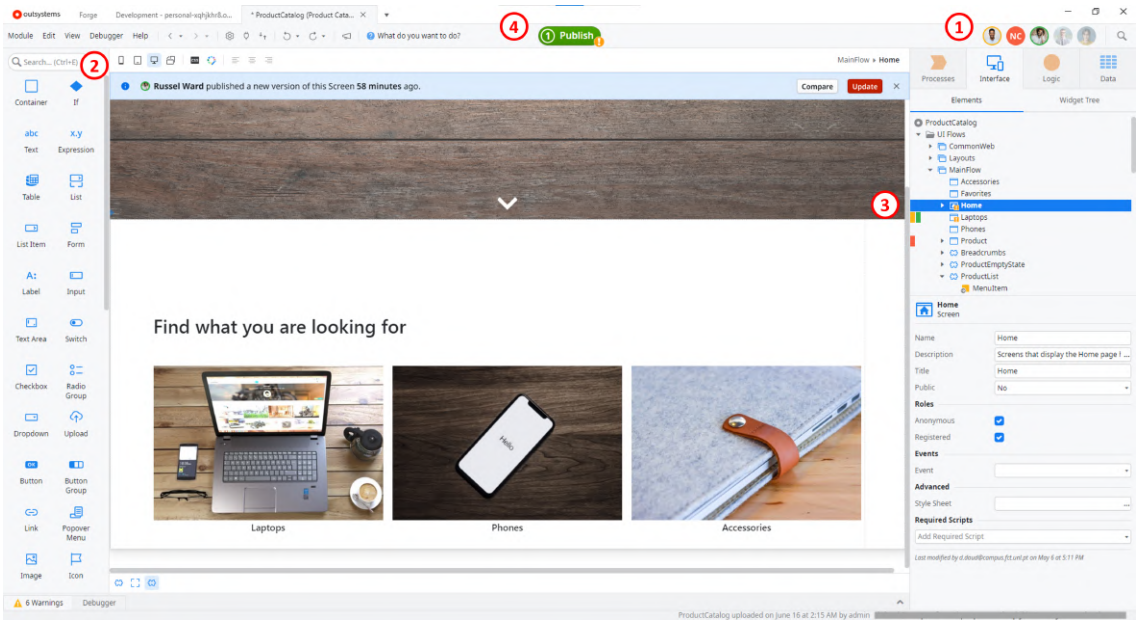
1. **Collaboration Bar** - The visual representation of other collaborators in the module.

2. **Element Changes Awareness** - A top blue banner appears when opening an element to indicate if the local version is out-of-date, with information regarding the time and author of the last publish and prompting the user to compare (the latest version with his own version) or update (merge their local version with the latest one) it. This way, users are aware of new published versions of the screens and actions before making any changes to them. This anticipates conflicts and improves coordination between colleagues.

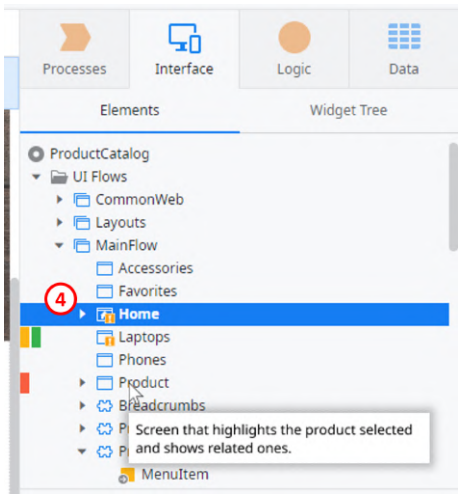
3. **Module Awareness** - Active users have a bar with their color next to the module element they have open in their Service Studio, so users can know where in the module their colleagues are working.

4. **Module Changes Awareness** - An exclamation point next to the Publish button indicates that there is a new published version of the module that conflicts with the local version. The objective is for users to take action immediately when they see this, so the conflict does not escalate. In the elements tree, there also is an exclamation point next to the elements with an update that conflicts with our local version, as demonstrated by Figure 5.13 (c).

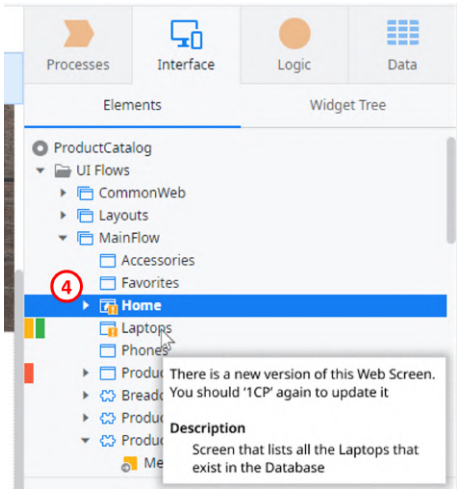
Users were pleased with this concept because it gives them the ability to have awareness on the changes happening to the module and, more specifically, to the elements they are working on. It facilitates communication between developers since it is possible to know where everyone is in the module and if there are any unexpected changes to its elements, causing conflicts. This way, conflicts are anticipated and immediately resolved before they can worsen. All test participants believe that if this concept were to be implemented, it could immediately bring a lot of value to application development. User K had a relevant suggestion regarding the options for the blue banner: "I think the compare option would only make sense to show up if I changed something, otherwise there is nothing to compare. The update option could show up if I did not make any changes so that Screen or Action would overwrite my local version. So both options could even be




(a) The user awareness final concept with visual indications for new changes to the module elements and where other users are working in the module




(b) Hovering an element typically shows a tooltip with the element's description




(c) Screens/actions with an exclamation point have conflicting changes as indicated by the tooltip

Figure 5.13: User awareness final concept prototype

exclusive, have either a Compare button or an Update button". We thought it could be a better way of doing things, so we asked all test participants for their opinion regarding this option, to which everyone agreed to make more sense than having both buttons.

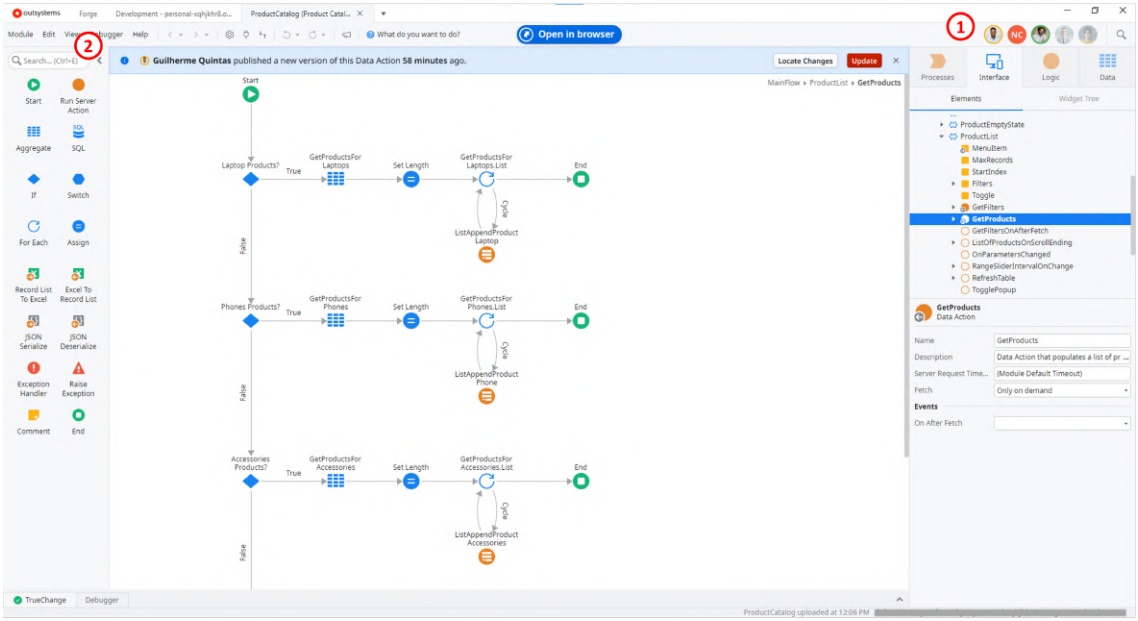
We believe this prototype is mature enough and there are not any more significant changes that could be done to test it (besides the mentioned option of having only a single button in the blue banner, which was tested and received positive feedback). As such, we believe this concept's testing was final.

### 5.3.3 Locate Conflicting Changes Concept Prototype

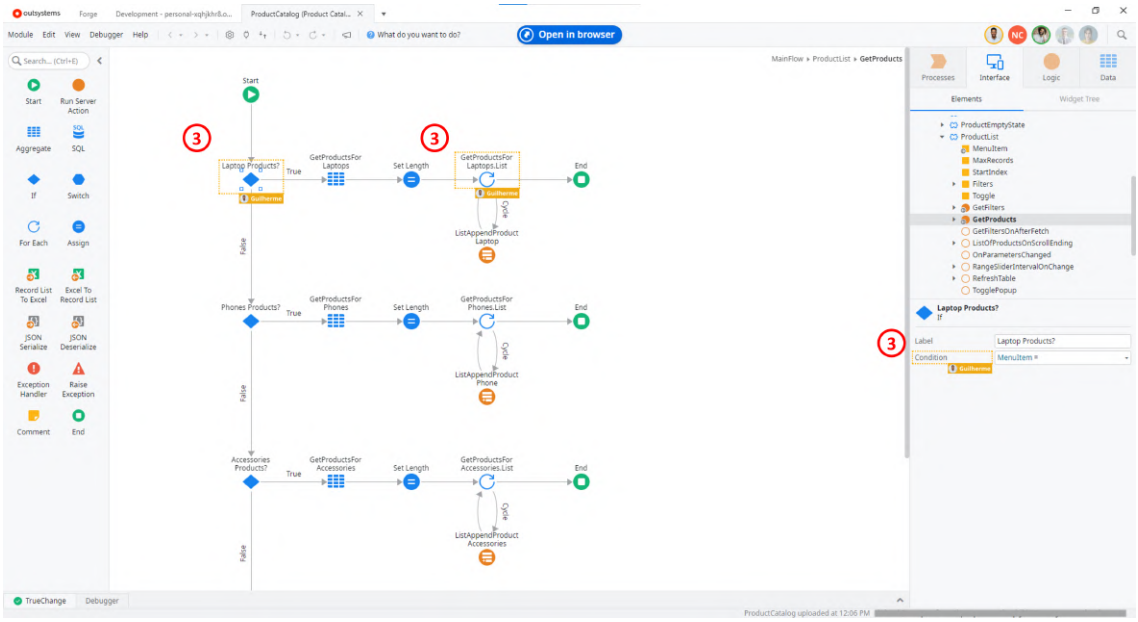
To finalize the testing of having a more granular way of showing changes within an element, as demonstrated in Subsection 5.1.2, we decided to try and tackle this solution one more time. Test participants were shown the prototype in Figure 5.14. In terms of added features, each number shown in the figure represents the following:

1. **Collaboration Bar** - The visual representation of other collaborators in the module.
2. **Element Changes Awareness** - As in the previous prototype, a top blue banner appears when opening an element to indicate if the local version is out-of-date, with information regarding the time and author of the last publish and prompting the user to compare or update his local element.
3. **Visually Locate Conflicting Changes** - When a user chooses to "Locate Changes", a box locates the components of the element with changes that conflict with our local version (meaning more than one developer is changing that component). This give a faster and more visual way of knowing where the conflicts are.

Having this granularity in identifying changes did not seem advantageous to all test participants because, when merging their work, the platform does not differentiate between the changes within the same element, forcing a developer to identify the changes and merge them within that element, even if they are independent of each other. If there are changes in that element, everything needs to be merged. We asked all users for their opinion considering a future where the merge process would be more granular and able to distinguish between these small changes, but no one's opinion changed. This was because they would still be working on a version of the element which is out-of-date, raising concerns over hidden changes (e.g., if something new was added, it will not be displayed since it is not a conflicting change). These were the same reasons given by users for the concept described in Subsection 5.1.2. User O complained about not being able to see the changes (just be notified about them): "if you cannot see the changes, what is the point?". Some users thought that having the ability to see those new changes as well could be more interesting, describing an ideal experience as a real-time collaboration feature (like the one prototyped before, described in Subsection 5.3.1, reiterating that a real-time collaboration feature could indeed bring value to developers). Other users thought this




(a) In this prototype, users are informed of a new version of an element available and then have a more visual way of knowing where the conflicts are




(b) Clicking the “Locate Changes” button, reveals a box around the components with changes conflicting with our local changes

Figure 5.14: Locate conflicting changes concept prototype

concept could be a way of having a better, more visual, compare feature, but it would require more details. As stated by User K, "it is a more visual and incomplete Compare operation". Most people would just rather have a higher level of awareness regarding changes.

# Chapter 6

# Conclusion

This chapter concludes this dissertation's work with a discussion on the research results, presented in the previous chapter, and elaborates on some points that could be further analyzed in the future.

## 6.1 Discussion

Like traditional programming technologies, collaborative development in OutSystems is based on an asynchronous process that relies on developers working in isolation and synchronizing their work periodically using merge operations. As real-time collaboration becomes a standard demanded by users for many other tasks, such as document editing, it is strategic for OutSystems to support this trend and research on the possibilities of having it implemented in its platform. Additionally, the increased adoption of remote work as a standard for many companies having their employees working from home only increases the motivation for our research.

Following the Design Thinking Methodology, our procedure was based on these four steps: empathize, define, ideate, and prototype. To empathize, we analyzed the scientific literature to find out what had already been researched regarding this topic; we looked for similar features in other software to understand the industry trends; finally, we talked to OutSystems users to determine how collaborative work currently happens and what issues it faces. Moving to the definition phase, we categorized all issues that we identified as being part of the collaborative process within the OutSystems ecosystem and selected the one we believed were more important to solve. Having found the problem to tackle, the "ideate" phase allowed us to brainstorm and discuss several options for solutions to solve our defined problem. Having our ideas more matured, we created several prototypes and tested them with users to understand if they were able to solve the issues they

currently face when collaboration with each other. We ended up by having three test iterations, which involved nine different prototypes.

During our test iterations, we collected a lot of feedback regarding our concepts. Looking at what the users commented during testing (qualitative metrics, available in the previous sections of this chapter) and what they answered in our survey (quantitative metrics available in Annex II), we believe that there are two ways to solve the problem "Delays and inefficiencies with excessive communication between developers (from different teams or the same one) to make sure no conflicts happen (e.g., calls, extra daily meetings)". The first solution can be applied in the short-term while creating a lot of value for users, as it is easier to implement and understand. The solution involves providing user awareness context inside of Service Studio, as described by the User Awareness Final Concept in Subsection 5.3.2. Users gave a lot of positive feedback over the prototypes with user awareness features. This happened because Service Studio currently does not provide any awareness as to what other users are doing or about changes being published. This creates a fear among developers of having their work mistakenly deleted by others and requires them to communicate at all times with each other to coordinate their actions. When they do not communicate, mistakes happen more often. This is why a solution with this type of implicit communication through awareness in Service Studio can bring so much value to end-users of the platform.

The second solution is real-time collaboration, and it focuses on improving collaboration by changing how teams work. This feature would take a lot more effort to implement but, as users said during testing, it could very much improve how collaboration occurs in the OutSystems reality. Users working remotely feel the need to discuss ideas with their peers, to show elements of the module to them, to debate, and to be able to coordinate their actions while all working on the same application. Real-time collaboration can be a solution as it enables all this, while also removing the need for developers to merge their work. This way, they can work at the same time on the same application while they see the changes immediately happen. Unlike what we initially expected, developers were not scared of this but actually enthusiastic about the prospect of being able to work in this manner. However, it makes sense for developers to work in isolation at times and, as such, a real-time collaboration feature would only make sense in certain phases of a project. During testing, having a way of creating real-time collaborative sessions with other users and being able to transfer between this way of working and the current isolated method, was considered the most efficient way to work and collaborate.

We then have two solutions for solving the collaboration constraints affecting OutSystems developers. These have different time frames since the User Awareness solution has a short-term implementation effort making it a quick solution which brings a lot of value, and the Real-time Collaboration solution with a long-term effort to implement and create sensibility with users on how to integrate it in their team's processes and work but with the potential of really improving the way applications are developed with OutSystems.

## 6.2 Future Work

Having achieved this thesis research objective, to create the experience of a solution to improve collaboration, the next steps would be to finalize the last step of our Design Thinking methodology: implement. The future work we propose would be to implement the proposed solutions in Service Studio and the OutSystems platform, and then have those implementations tested with users. This would generate more and better feedback from users, as they could actually interact with the concepts in a real-life scenario (e.g., have a development team use these features throughout the execution of a project and collect metrics regarding the time improvements on communication and development). Having these tests done by many people (e.g., at least fifteen people) could produce quantitative data from these tests that could be used to reach to conclusions regarding the usability and user experience of the implemented prototypes, which was not possible in this thesis focused on gathering qualitative data.

During testing, there was feedback concerning other use cases which are not in the scope of this research work but could be relevant for the further innovation and development of OutSystems' products. Many of the features tested were discarded for being only useful for senior management use cases, like the Follow feature described in Subsection 5.2.1, the Subscription to Changes feature described in Subsection 5.2.2 and the Version History feature from Subsection 5.2.3. These feature could be more worth exploring to develop better ways of having team leaders and seniors perform the activities which are part of their work in Service Studio.

Another common topic throughout testing was the compare and merge operation. Our research goal was to explore better alternatives to this model of working, focusing on real-time collaboration features. However, future work could explore better ways of performing these operations. A common suggestion among users, especially when hearing feedback from the concept described in Subsection 5.3.3, was to have a more visual Compare operation, where it could be possible to have a side-by-side view of the local and most recent published version of that Screen or Action, and be able to drag-and-drop from the most recent version to the local one. This idea was also demonstrated in the ideation session by Participant F (see Figure 6.1).

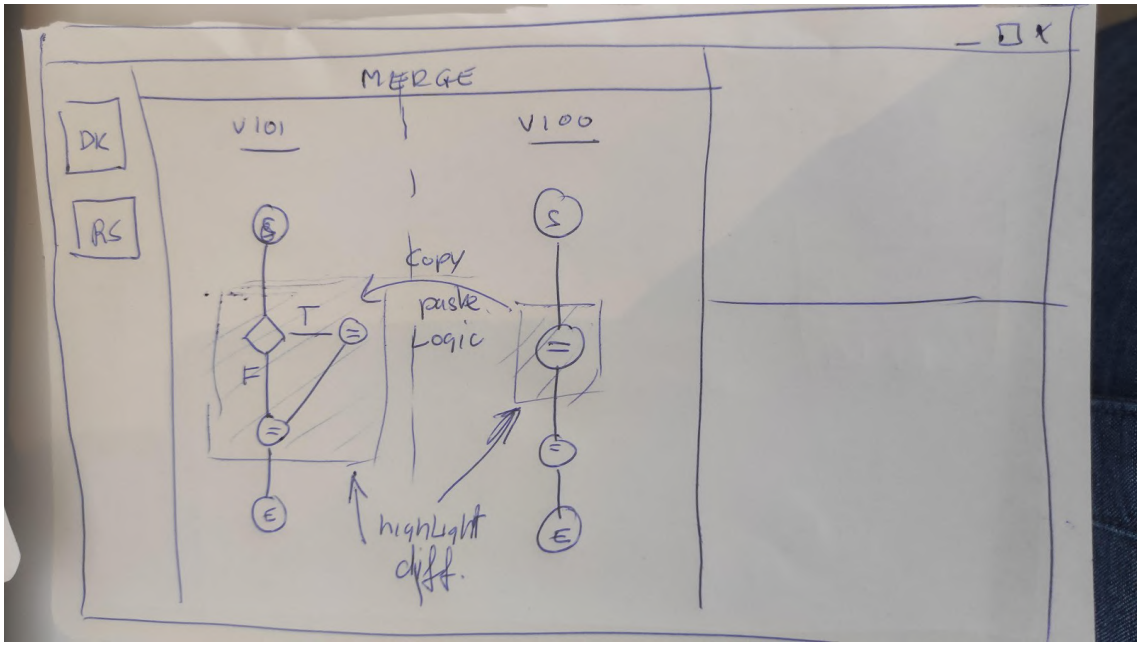



Figure 6.1: Sketch from Participant F, suggesting a more visual way of comparing and merging changes

# Bibliography


[1] *4 Capabilities App Dev Platforms Need for Whole-Team Cross-Functional Collaboration*. URL: https://www.outsystems.com/blog/posts/whole-team-cross-functional-collaboration/ (visited on Feb. 12, 2021).

[2] L. Waizenegger et al. "An affordance perspective of team collaboration and enforced working from home during COVID-19". In: *European Journal of Information Systems* 29.4 (2020), pp. 429–442.

[3] C. A. Ellis, S. J. Gibbs, and G. Rein. "Groupware: some issues and experiences". In: *Communications of the ACM* 34.1 (1991), pp. 39–58.

[4] P. Dourish and V. Bellotti. "Awareness and coordination in shared workspaces". In: *Proceedings of the 1992 ACM conference on Computer-supported cooperative work*. 1992, pp. 107–114.

[5] T. Gross, C. Stary, and A. Totter. "User-centered awareness in computer-supported cooperative work-systems: Structured embedding of findings from social sciences". In: *International Journal of Human-Computer Interaction* 18.3 (2005), pp. 323–360.

[6] *Common Use Cases - Visual Studio Live Share | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/visualstudio/liveshare/reference/use-cases (visited on Feb. 11, 2021).

[7] P. Ralph et al. "Pandemic programming". In: *Empirical Software Engineering* 25.6 (2020), pp. 4927–4961.

[8] D. Ford et al. "A tale of two cities: Software developers working from home during the covid-19 pandemic". In: *arXiv preprint arXiv:2008.11147* (2020).

[9] A. W. Bartik et al. *What jobs are being done at home during the COVID-19 crisis? Evidence from firm-level surveys*. Tech. rep. National Bureau of Economic Research, 2020.

[10] *Low-Code: The full guide to low-code platforms and development | OutSystems*. URL: https://www.outsystems.com/low-code-platforms/ (visited on Jan. 26, 2021).

[11] F. Alexander. *What Is Low-Code? [2021 Update]*. URL: https://www.outsystems.com/blog/posts/what-is-low-code/ (visited on Jan. 26, 2021).

[12] P. Vincent et al. "Magic quadrant for enterprise low-code application platforms". In: *Gartner report* (2019).

[13] *Developing with OutSystems | Evaluation Guide | OutSystems*. URL: https://www.outsystems.com/evaluation-guide/developing-with-outsystems/ (visited on Jan. 26, 2021).

[14] *OutSystems development and management tools | Evaluation Guide | OutSystems*. URL: https://www.outsystems.com/evaluation-guide/development-and-management-tools/ (visited on Jan. 26, 2021).

[15] *Visual Studio IDE, Code Editor, Azure DevOps, & App Center - Visual Studio*. URL: https://visualstudio.microsoft.com/ (visited on Feb. 11, 2021).

[16] *C Sharp docs - get started, tutorials, reference. | Microsoft Docs*. URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (visited on Nov. 14, 2021).

[17] *Possible setups for an OutSystems infrastructure | OutSystems*. URL: https://success.outsystems.com/Documentation/11/Setting_Up_OutSystems/Possible_setups_for_an_OutSystems_infrastructure (visited on Feb. 22, 2021).

[18] *Components and Tools*. Taken from the "Lesson Materials". URL: https://www.outsystems.com/training/lesson/2183/components-and-tools (visited on Jan. 27, 2021).

[19] *OutSystems Platform Services | Evaluation Guide | OutSystems*. URL: https://www.outsystems.com/evaluation-guide/platform-services/ (visited on Jan. 26, 2021).

[20] *Service Studio Overview - OutSystems*. URL: https://success.outsystems.com/Documentation/11/Getting_started/Service_Studio_Overview (visited on Jan. 26, 2021).

[21] *How does OutSystems enable team collaboration? | Evaluation Guide | OutSystems*. URL: https://www.outsystems.com/evaluation-guide/how-does-outsystems-enable-team-collaboration (visited on Feb. 6, 2021).

[22] *The merge feature and team collaboration - OutSystems*. URL: https://success.outsystems.com/Documentation/11/Developing_an_Application/Merge_the_Work/The_merge_feature_and_team_collaboration (visited on Feb. 6, 2021).

[23] *Merge the Work - OutSystems*. URL: https://success.outsystems.com/Documentation/11/Developing_an_Application/Merge_the_Work (visited on Feb. 7, 2021).

[24] T. A. G. de Almeida. "Real-time collaborative editing of OutSystems DSL models". MA thesis. NOVA University Lisbon, 2013.

[25] T. Huff. *Agile and Scrum: Understanding the Differences*. URL: https://www.outsystems.com/blog/posts/agile-and-scrum/ (visited on Feb. 7, 2021).

[26] *Team Overview*. Screenshot by author. URL: https://www.outsystems.com/training/lesson/1297/team-overview (visited on Feb. 7, 2021).

[27] *Tech Lead*. URL: https://www.outsystems.com/training/lesson/1296/tech-lead (visited on Feb. 8, 2021).

[28] *Tech Lead*. URL: https://www.outsystems.com/training/lesson/1298/developer (visited on Feb. 8, 2021).

[29] J. R. Wallace, S. Oji, and C. Anslow. "Technologies, methods, and values: changes in empirical research at CSCW 1990-2015". In: *Proceedings of the ACM on Human-Computer Interaction* 1.CSCW (2017), pp. 1–18.

[30] J. Grudin. "Computer-supported cooperative work: History and focus". In: *Computer* 27.5 (1994), pp. 19–26.

[31] L. J. Bannon and K. Schmidt. "CSCW: Four characters in search of a context". In: *ECSCW 1989: Proceedings of the First European Conference on Computer Supported Cooperative Work*. Computer Sciences Company, London. 1989.

[32] D. Damian et al. "Awareness in the wild: Why communication breakdowns occur". In: *International Conference on Global Software Engineering (ICGSE 2007)*. IEEE. 2007, pp. 81–90.

[33] C. Gutwin et al. "Supporting group awareness in distributed software development". In: *IFIP International Conference on Engineering for Human-Computer Interaction*. Springer. 2004, pp. 383–397.

[34] C. Gutwin, S. Greenberg, and M. Roseman. "Workspace awareness in real-time distributed groupware: Framework, widgets, and evaluation". In: *People and Computers XI*. Springer, 1996, pp. 281–298.

[35] C. Gutwin and S. Greenberg. "A descriptive framework of workspace awareness for real-time groupware". In: *Computer Supported Cooperative Work (CSCW)* 11.3 (2002), pp. 411–446.

[36] R. Johansen. *Groupware: Computer support for business teams*. The Free Press, 1988.

[37] V. M. R. Penichet et al. "A classification method for CSCW systems". In: *Electronic Notes in Theoretical Computer Science* 168 (2007), pp. 237–247.

[38] M. Goldman. "Role-based interfaces for collaborative software development". In: *Proceedings of the 24th annual ACM symposium adjunct on User interface software and technology*. 2011, pp. 23–26.

[39] C. M. Pilato, B. Collins-Sussman, and B. W. Fitzpatrick. *Version control with subversion: next generation open source version control*. "O'Reilly Media, Inc.", 2008.

[40] *Apache Subversion*. URL: https://subversion.apache.org/ (visited on Nov. 16, 2021).

[41] *Git*. URL: https://git-scm.com/ (visited on Nov. 16, 2021).

[42] *Azure DevOps Server | Microsoft Azure*. URL: https://azure.microsoft.com/services/devops/server/ (visited on Nov. 16, 2021).

[43] *Low-code Application Development Platform - Build Apps Fast & Efficiently | Mendix*. URL: https://www.mendix.com/ (visited on Jan. 10, 2021).

[44] *CRM Software & Cloud Computing Solutions - Salesforce EMEA*. URL: https://www.salesforce.com/ (visited on Jan. 10, 2021).

[45] *Kony Is Now Temenos - Temenos*. URL: https://www.kony.com/ (visited on Feb. 11, 2021).

[46] *Use Claris FileMaker to Build Business Applications — Claris*. URL: https://www.claris.com/filemaker/ (visited on Feb. 11, 2021).

[47] *Appian: Low-Code Automation | Business Apps | BPM | RPA*. URL: https://www.appian.com/ (visited on Jan. 10, 2021).

[48] *Low-code app development for enterprises | Pega*. URL: https://www.pega.com/products/pega-platform/low-code-app-development (visited on Jan. 10, 2021).

[49] *Eclipse desktop & web IDEs | The Eclipse Foundation*. URL: https://www.eclipse.org/ide/ (visited on Feb. 11, 2021).

[50] *AWS Cloud9 – Amazon Web Services*. URL: https://aws.amazon.com/pt/cloud9/ (visited on Feb. 11, 2021).

[51] *IntelliJ IDEA: The Capable & Ergonomic Java IDE by JetBrains*. URL: https://www.jetbrains.com/idea/ (visited on Feb. 11, 2021).

[52] *Atom*. URL: https://atom.io/ (visited on Feb. 11, 2021).

[53] *CodePen: Online Code Editor and Front End Web Developer Community*. URL: https://codepen.io/ (visited on Feb. 11, 2021).

[54] *Google Drive Blog: A rebuilt, more real time Google documents*. URL: https://drive.googleblog.com/2010/04/a-rebuilt-more-real-time-google.html (visited on Feb. 14, 2021).

[55] *Create, find, or download a file - Computer - Docs Editors Help*. URL: https://support.google.com/docs/answer/49114 (visited on Feb. 11, 2021).

[56] *Find what's changed in a file - Computer - Docs Editors Help*. URL: https://support.google.com/docs/answer/190843 (visited on Feb. 11, 2021).

[57] *Chat with others in a file - Computer - Docs Editors Help*. URL: https://support.google.com/docs/answer/2494891 (visited on Feb. 11, 2021).

[58] *Visual Studio Live Share | Visual Studio - Visual Studio*. URL: https://visualstudio.microsoft.com/services/live-share/ (visited on Feb. 11, 2021).

[59] K. Baker, S. Greenberg, and C. Gutwin. "Empirical development of a heuristic evaluation methodology for shared workspace groupware". In: *Proceedings of the 2002 ACM conference on Computer supported cooperative work*. 2002, pp. 96–105.

[60] D. Norman and J. Nielsen. *The Definition of User Experience (UX)*. URL: https://www.nngroup.com/articles/definition-user-experience/ (visited on Feb. 20, 2021).

[61] *Ergonomics of human-system interaction — Part 11: Usability: Definitions and concepts*. Standard. International Organization for Standardization, 2018.

[62] W. Quesenbery. *Using the 5Es to Understand Users - Whitney Interactive Design*. URL: https://www.wqusability.com/articles/getting-started.html (visited on Feb. 20, 2021).

[63] C. M. Barnum. *Usability testing essentials*. Elsevier, 2010.

[64] A. Joyce. *Formative vs. Summative Evaluations*. URL: https://www.nngroup.com/articles/formative-vs-summative-evaluations/ (visited on Feb. 20, 2021).

[65] C. M. Barnum. *Usability testing essentials: ready, set... test!* Morgan Kaufmann, 2020.

[66] A. Holzinger. "Usability engineering methods for software developers". In: *Communications of the ACM* 48.1 (2005), pp. 71–74.

[67] J. Nielsen and R. Molich. "Heuristic evaluation of user interfaces". In: *Proceedings of the SIGCHI conference on Human factors in computing systems*. 1990, pp. 249–256.

[68] *Concept Testing: Moving Forward with the Right Ideas | Maze*. URL: https://maze.co/guides/concept-testing/ (visited on Aug. 16, 2021).

[69] *Rapid, remote testing for agile teams*. URL: https://maze.co/ (visited on Aug. 18, 2021).

[70] *Concept Testing: Moving Forward with the Right Ideas | Maze*. URL: https://maze.co/guides/concept-testing/survey (visited on Aug. 16, 2021).

[71] *Concept Testing: Moving Forward with the Right Ideas | Maze*. URL: https://maze.co/guides/concept-testing/product (visited on Aug. 16, 2021).

[72] M. G. Haselton, D. Nettle, and D. R. Murray. "The evolution of cognitive bias". In: *The handbook of evolutionary psychology* (2015), pp. 1–20.

[73] J. Liedtka. "Perspective: Linking design thinking with innovation outcomes through cognitive bias reduction". In: *Journal of product innovation management* 32.6 (2015), pp. 925–938.

[74] *Decision Frames: How Cognitive Biases Affect UX Practitioners*. URL: https://www.nngroup.com/articles/decision-framing-cognitive-bias-ux-pros/ (visited on Aug. 21, 2021).

[75] T. Brown. "Design Thinking". In: *harvard business review* (2008), p. 1.

[76] *Design Thinking 101*. URL: https://www.nngroup.com/articles/design-thinking/ (visited on Aug. 22, 2021).

[77] *Design Thinking 101*. URL: https://www.nngroup.com/articles/design-thinking/ (visited on Feb. 21, 2021).

[78] M. E. Conway. "How do committees invent". In: *Datamation* 14.4 (1968), pp. 28–31.

[79] *RICE Prioritization Framework for Product Managers [+Examples]*. URL: https://www.intercom.com/blog/rice-simple-prioritization-for-product-managers/ (visited on Dec. 10, 2020).

[80] J. Knapp, J. Zeratsky, and B. Kowitz. *Sprint: How to solve big problems and test new ideas in just five days*. Simon and Schuster, 2016.

[81] J. D. Gould, J. Conti, and T. Hovanyecz. "Composing letters with a simulated listening typewriter". In: *Communications of the ACM* 26.4 (1983), pp. 295–308.

[82] *Figma: the collaborative interface design tool*. URL: https://www.figma.com/ (visited on Aug. 28, 2021).

[83] J. Nielsen and T. K. Landauer. "A mathematical model of the finding of usability problems". In: *Proceedings of the INTERACT'93 and CHI'93 conference on Human factors in computing systems*. 1993, pp. 206–213.

Appendix A

# Identification and classification of collaboration issues

Table A.1: Reported issues and their RICE score (in descending order)

| Reported issues | RICE score | | | | |
|---|---|---|---|---|---|
| | **Reach (0-100)** | **Impact (0.5;1;2;3)** | **Confidence (0-100%)** | **Effort (0-100)** | **Total** |
| Delays and inefficiencies with excessive communication between developers (from different teams or the same one) to make sure no conflicts happen (e.g., calls, extra daily meetings) | 90 | 2 | 90% | 40 | 4.05 |
| Conflicts happening with insignificant changes like moving an object of an Action, due to the merge granularity | 70 | 2 | 60% | 50 | 1.68 |
| Difficult to identify changes when trying to resolve merge conflicts in more complex screens of the Interface layer (e.g., the Web/Reactive Screen) | 65 | 2 | 80% | 65 | 1.6 |

| | | | | | |
|---|---|---|---|---|---|
| Pair-programming is essential for training new employees with Service Studio. Because of the pandemic, pair-programming has to be done remotely with screen sharing, which can be inconvenient at times (e.g., slow internet connection) and is very limited (only works one way, there cannot be more than 1 person editing) | 30 | 2 | 80% | 45 | 1.07 |
| Product architecture's focus should be on bringing value to the customers and the business. However, the architecture of a product also has to be planned in order to avoid conflicts between developers during development, decreasing the value it can deliver in the process | 80 | 2 | 50% | 85 | 0.94 |
| Not possible to have large teams taking part in OutSystems projects, because conflicts would make it impossible to work | 60 | 2 | 50% | 65 | 0.92 |
| Difficult to identify changes when trying to resolve merge conflicts in more complex actions of the Logic layer (e.g., big Actions) | 30 | 2 | 80% | 65 | 0.74 |
| Lack of pair-programming capabilities in Service Studio makes job interviews more difficult | 20 | 0.5 | 80% | 45 | 0.18 |
| Code being deleted because of another developer's published changes after executing the merge operation | 5 | 3 | 60% | 55 | 0.16 |

APPENDIX B

# Sketches from the Ideation Session

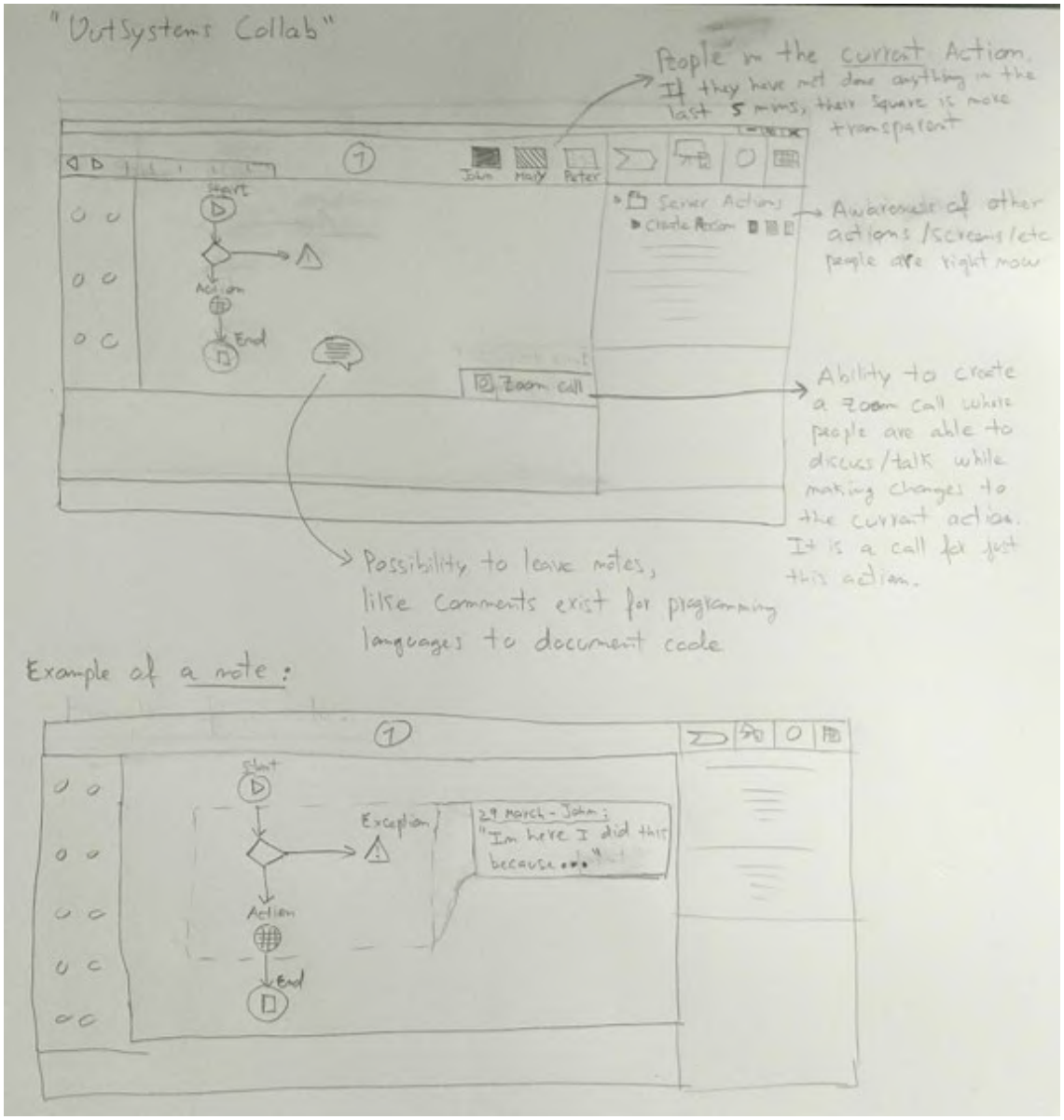
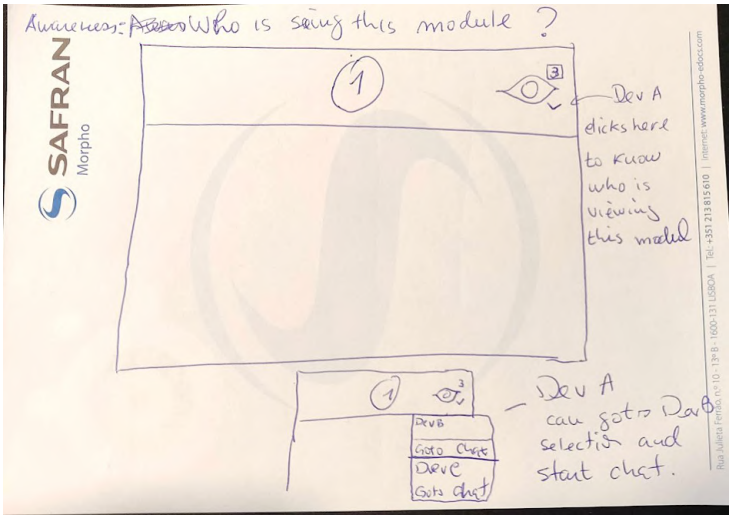
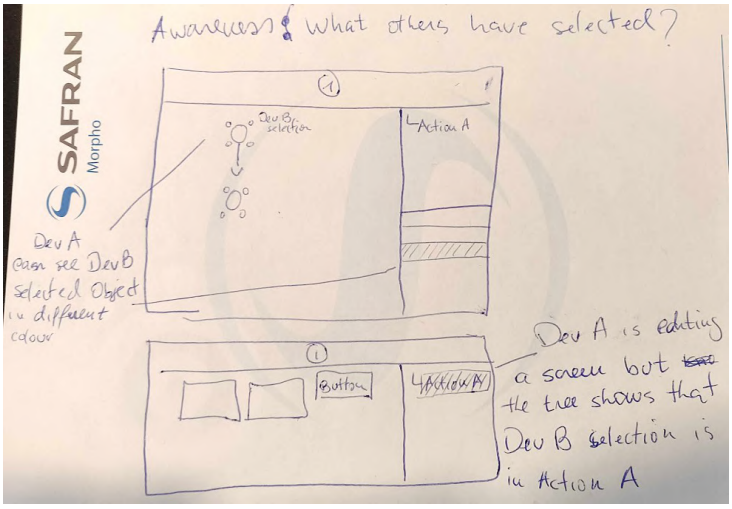


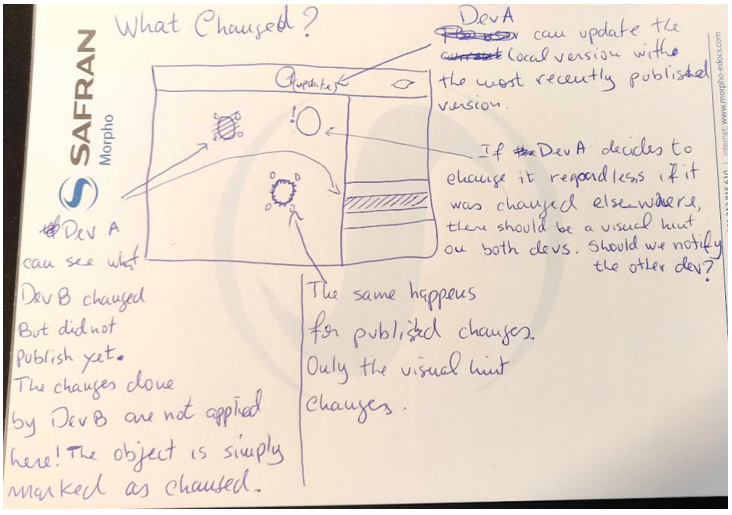

Figure B.1: Sketches from Participant A




(a)




(b)




(c)

Figure B.2: Sketches from Participant B

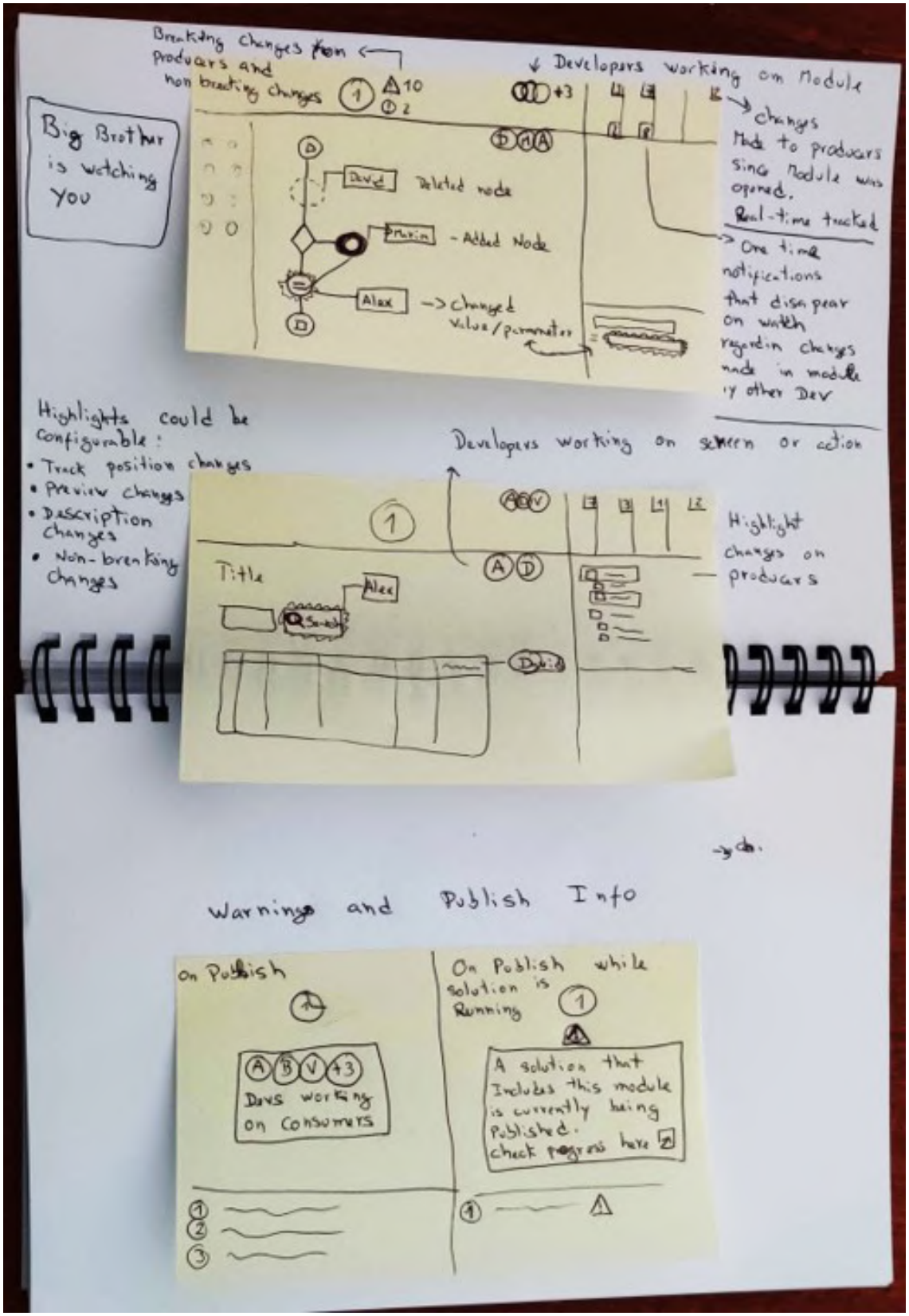



Figure B.3: Sketches from Participant C

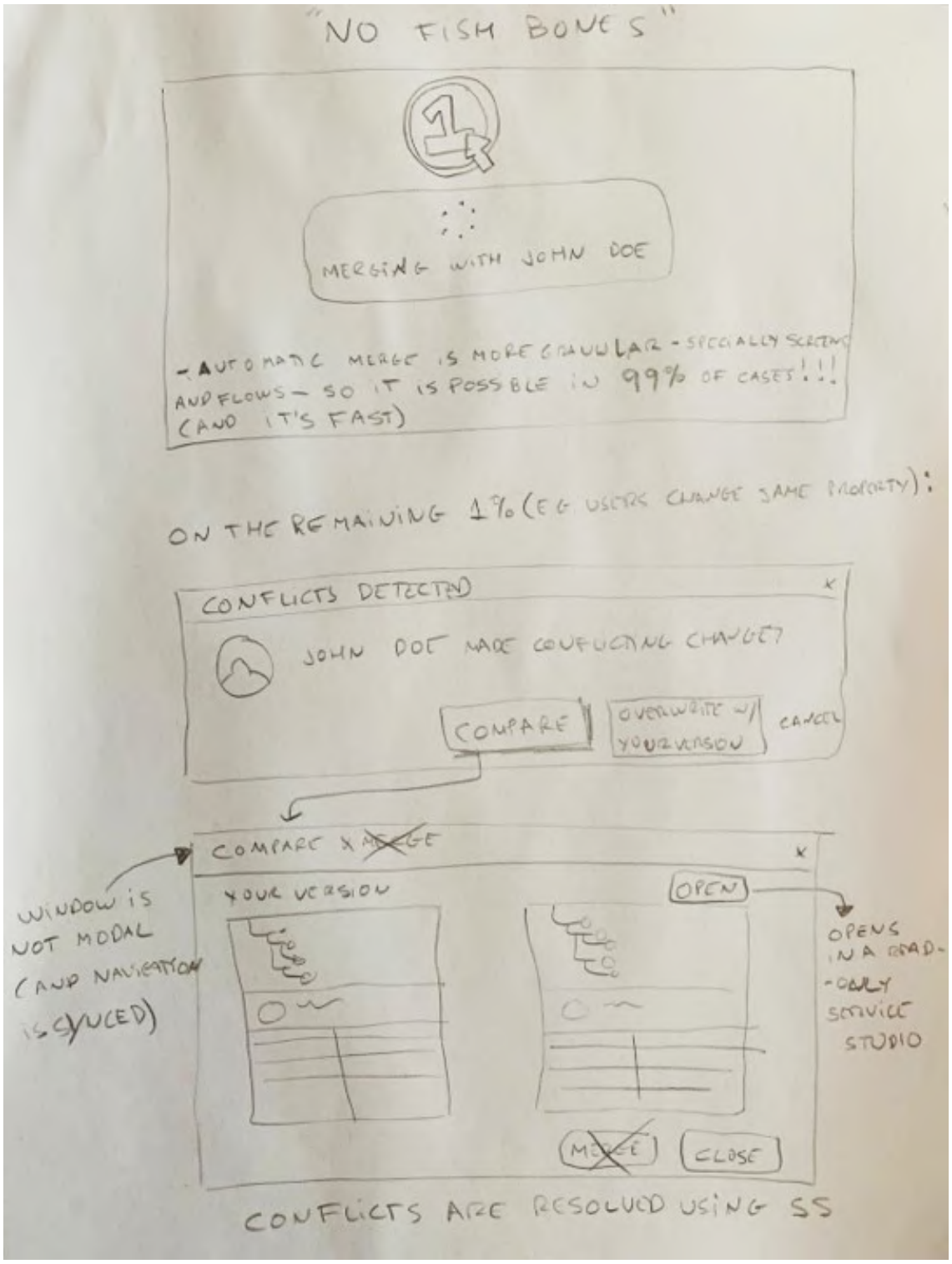
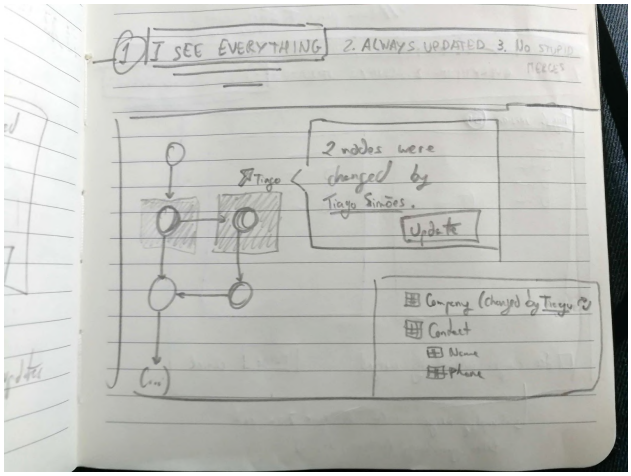
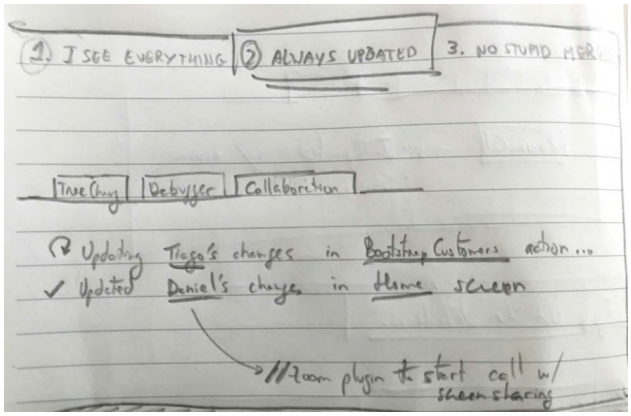


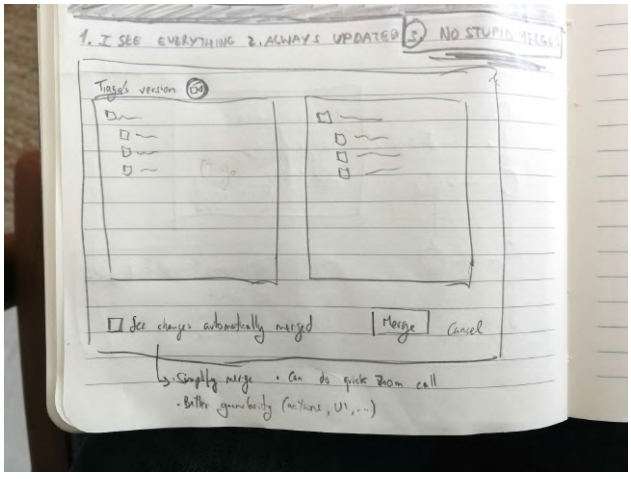

Figure B.4: Sketches from Participant D




(a)

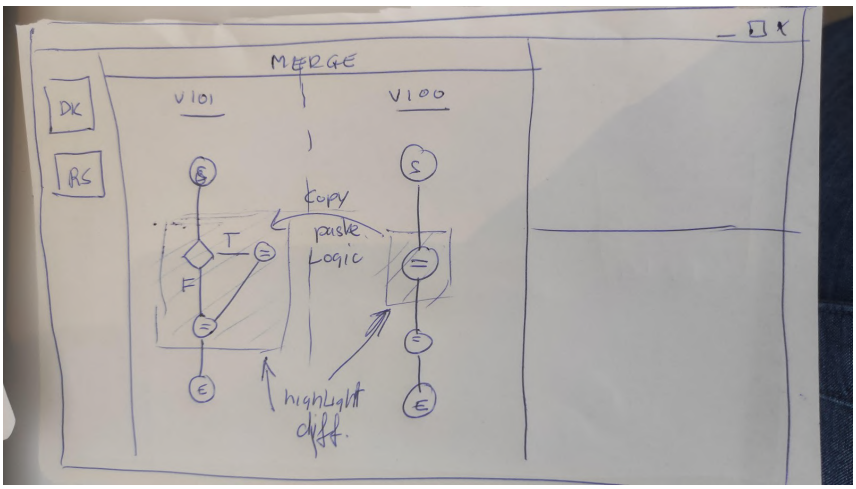



(b)

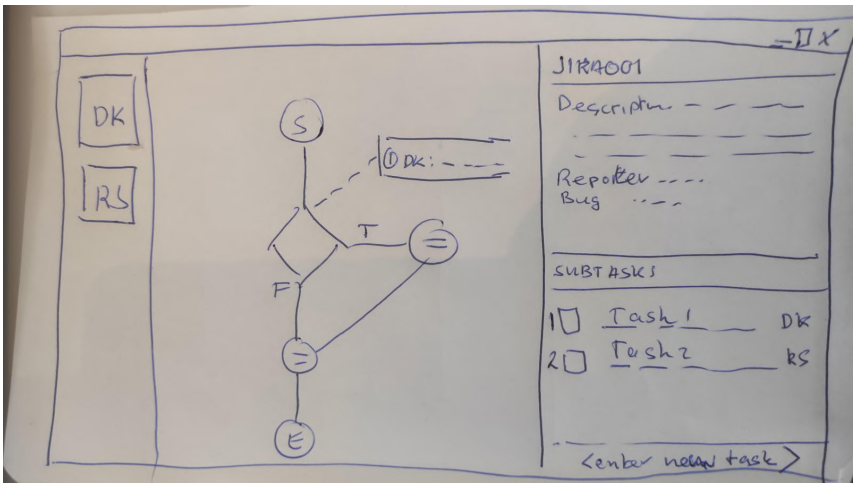



(c)

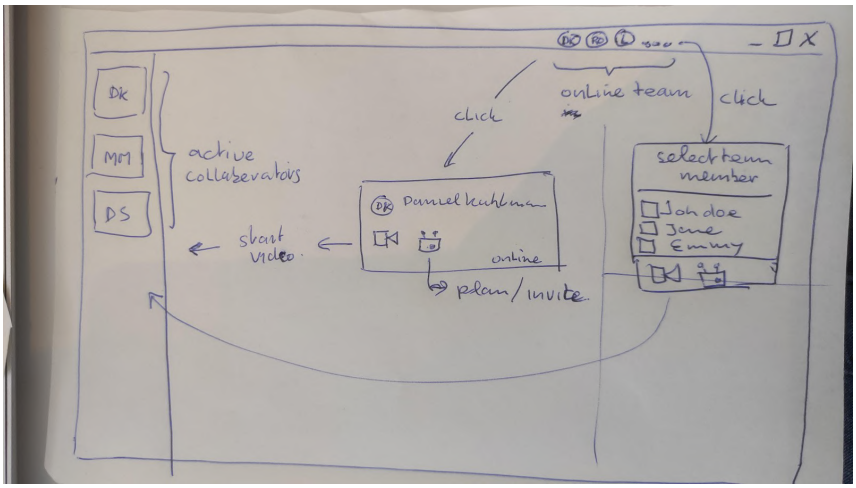
Figure B.5: Sketches from Participant E




(a)




(b)




(c)

Figure B.6: Sketches from Participant F

ANNEX I

# Concept Testing Script

Below is the script used as reference throughout the concept testing interviews, part of the "test" phase of our Design Thinking methodology.

outsystems

# Script

## Real-Time Collaboration - **Concept** testing for awareness and synchronous changes in Service Studio

## Introduction - Goals and Procedures of the session

Hello, my name is David and I am currently doing my Master's thesis and working at R&D at OutSystems.

I'd like to begin by thanking you for taking the time to speak with us. Your feedback is really valuable, and will help us determine if some of the features that you will see today bring any benefits to users and work as intended.

I would like to keep this session to about 1 hour. Does that work? Great.

During this session we will:

1. Start by making a few introductory questions about you and your previous experience with Service Studio
2. Then, we will present you some concepts we have developed to leverage collaboration and reduce the number of conflicts when developing in OutSystems. We will first explain to you the concept and then show you some examples. Please bear in mind that the images you will see are just illustrative of the concepts and do not represent any final or definitive version of them.

3. We then have a small questionnaire that I will ask you to answer after this session. And we will also have some time for any final comments or questions that you may have for us.

Some more protocolar issues:

- With your permission, I'd like to record this call. The recording will only be used to help us to improve the tools and features, and it won't be seen by anyone except those with a need-to-know. Recording this call also helps us, because I don't have to take as many notes! Do I have your permission to record the call?

- It's also really important to highlight that we are only testing the concepts, not you. There are no right or wrong answers. You can't do or say anything wrong here. Please feel free to let me know at any time if there's something you like, dislike, if you're confused, etc. I promise you won't hurt my feelings or the team. We want to improve, so please express what you really think.

- Also, I'd like you to "think aloud" as much as possible during the meeting. By that, I mean that I'd like you to speak your thoughts as often as you can. For example, you may be looking at a page, and suddenly see something you didn't see before. In that case, saying something like "oh, this caught my eye because..." would be very useful.

Do you have any questions for me before we start? Great.

# Part I - Warm-up, background, get comfortable

1. Now I would like to ask you to present yourself. Can you tell me about your role and what you do in your daily work?
2. How many years of experience with OutSystems?
3. What projects have you taken part of? What was the size of the biggest **team**?

# Part II - Scenario and concepts

Before presenting the concepts, I would like to start with a couple of general questions.

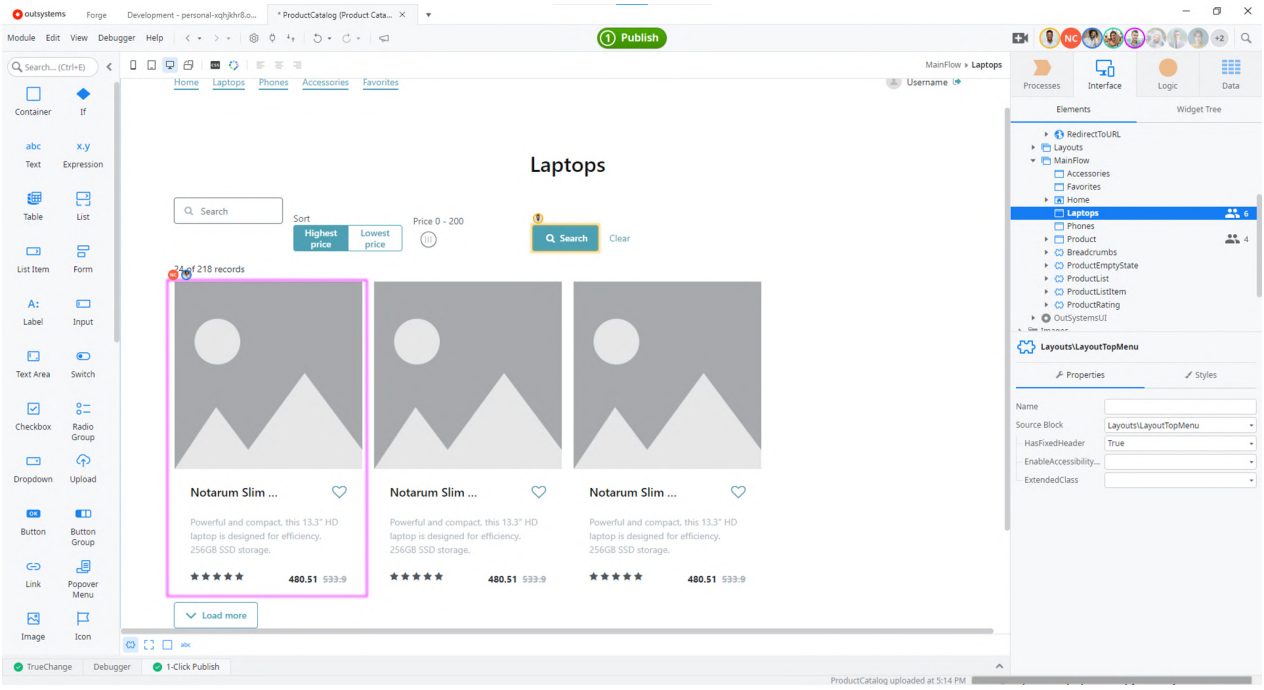
**Q1**. How would you describe your current experience of working with other developers in Service Studio in the same project? How does it compare to working in the office?

**Q1.1**. If he/she mentioned problems when collaborating - If you were to design an ideal solution to those problems, please describe what that solution would look like.

**Q1.2.** If he/she did not mention any problems when collaborating **-** Did you have any problems in your last project? How do you normally work during a project, do you have workarounds in place?

**Q2**. Lets say OutSystems wanted to integrate Real-Time Collaboration into Service Studio, what kind of experience comes to mind?

## Prototype (repeat the following steps for each prototype shown)




**(Let the user play around and speak about what he thinks he is seeing)**

**(Clarify the concept)**

**Evaluate as a whole:**

**Q1**. What are your initial thoughts on this concept?

**Evaluate specific characteristics of the concept:**

**Q2**. What do you think about this X attribute? (Continue for each attribute)

**Q3**. What are the benefits? Further probing questions might be:

**Q3.1**. Would it help you to do something more easily or better?

**Evaluate the connection of the user with the concept:**

**Q4**. How would it impact your work?

**Q5**. When and where would you use this?

**Q6**. What do you like the most and the least regarding this concept?

**Q7**. If you had a magic wand and could change something from this concept, what would it be?

**(If more than one prototype was shown, ask the user to order all prototypes according to his/her preference and to justify his/her choices.)**

# Part III - Post-test questions

To end this test, I have a very small questionnaire that should only take 2 minutes to complete that I will send to you and would ask to fill it out immediately after this session. Would that be alright?

# Wrap-up and thanks

We finished the test. Thank you for your participation!

Is there anything else you would like to share? Thank you for your time! Please do not hesitate to talk to me or the team if you think of additional points that we should be aware of or if you have any questions.

ANNEX II

# Concept Testing Survey

Below are the three surveys, and their respective results, sent to users of each corresponding test iteration.

## II.1 First Iteration Survey

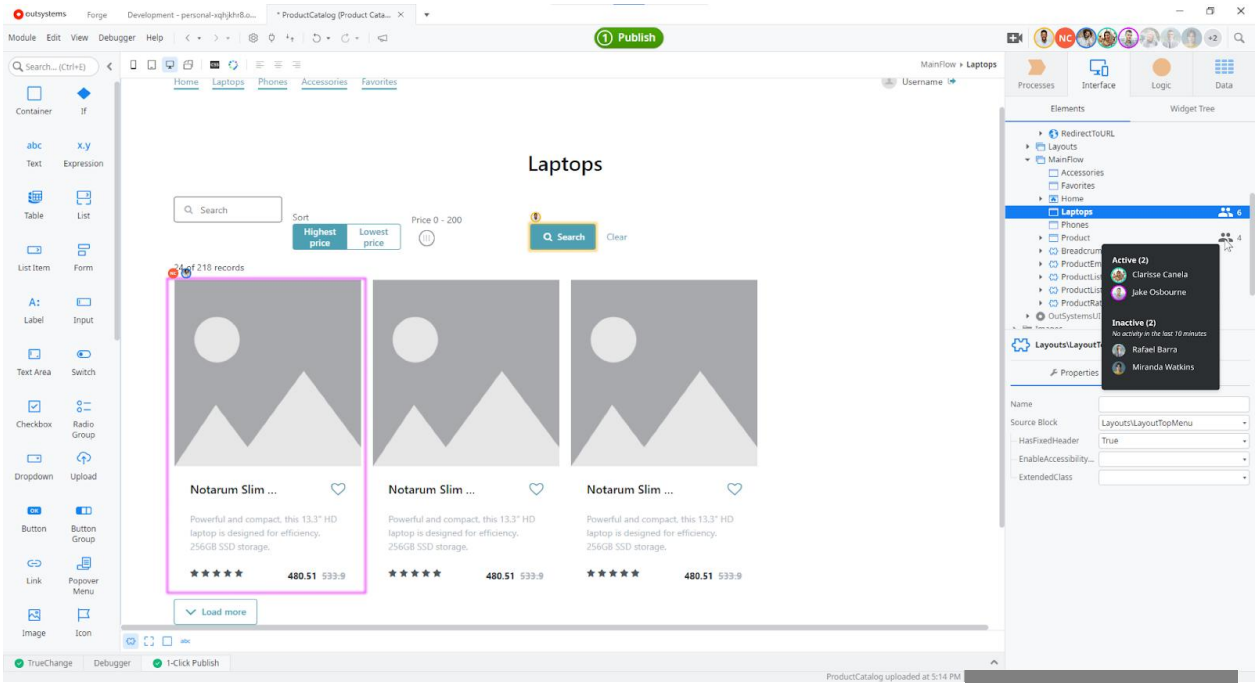
# Real-Time Collaboration Concept Testing Survey

Time to complete: 2 mins

Thank you for being a part of the concept test. To finalize, please fill out this short form.

* Required

## User awareness concept



1. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| This concept improves communication between developers working on the same project | | | | | |
| This concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

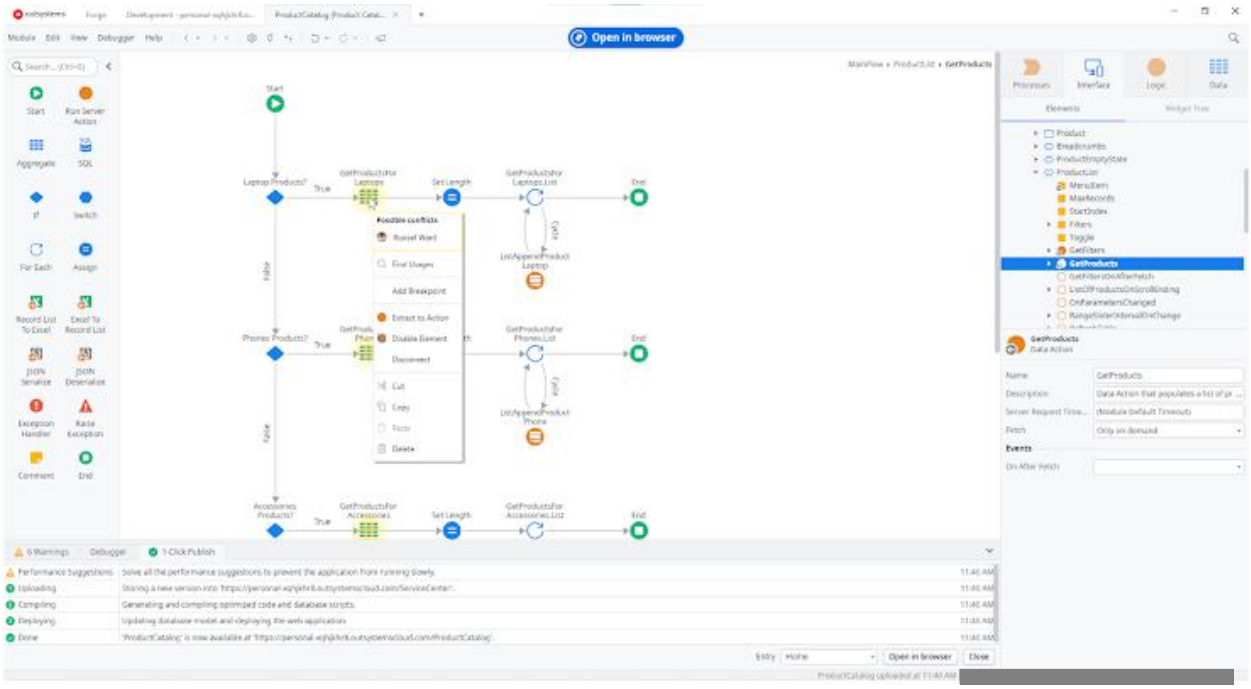
2. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

3. Any comments about this concept?

Visual awareness of conflicting changes concept



4. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| The concept improves communication between developers working on the same project | | | | | |
| The concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

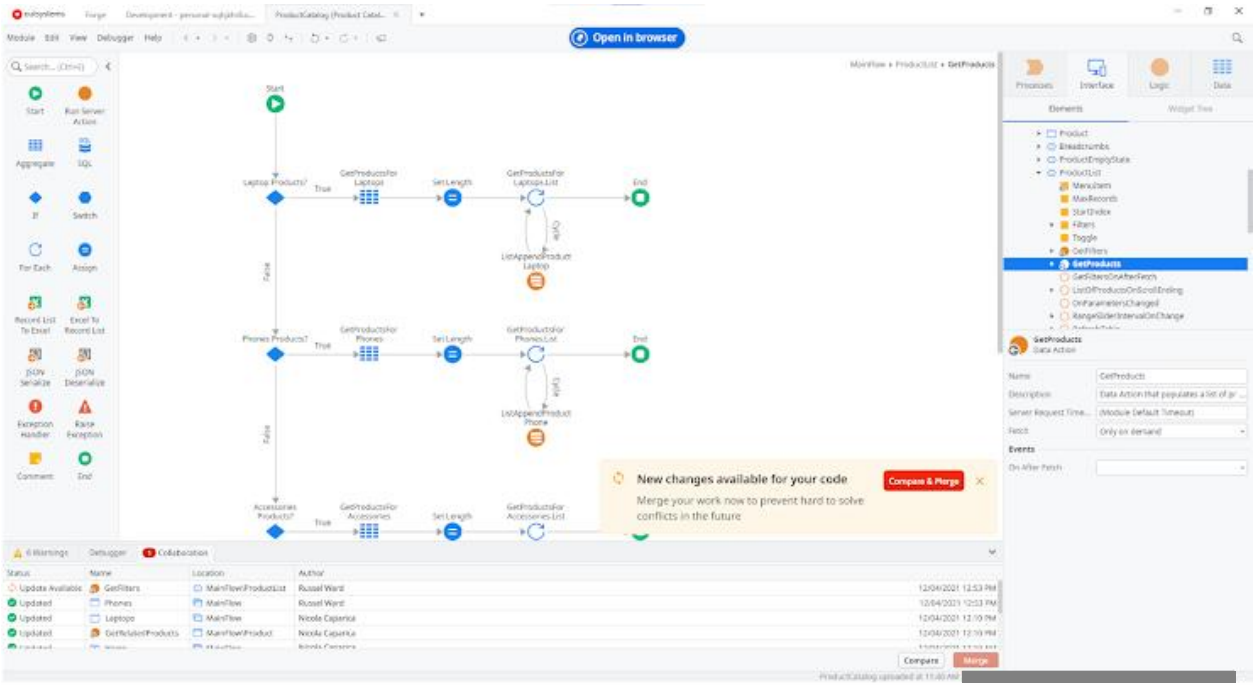
5. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

6. Any comments about this concept?

Automatic and manual updates with a notification concept



7. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| The concept improves communication between developers working on the same project | | | | | |
| The concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

8. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

9. Any comments about this concept?

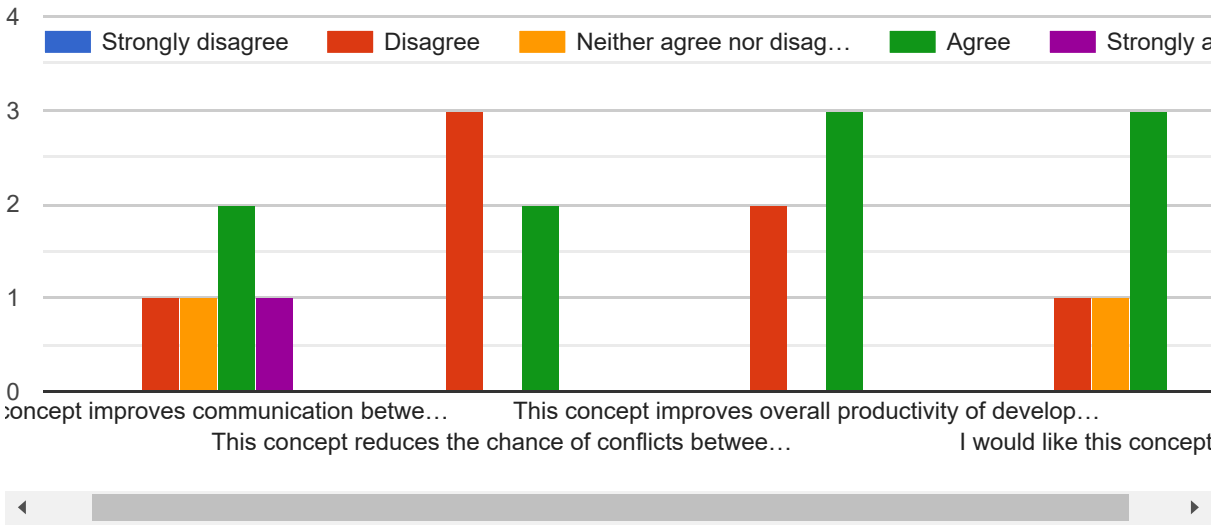
### II.1.1 Results

# Real-Time Collaboration Concept Testing Survey

5 responses

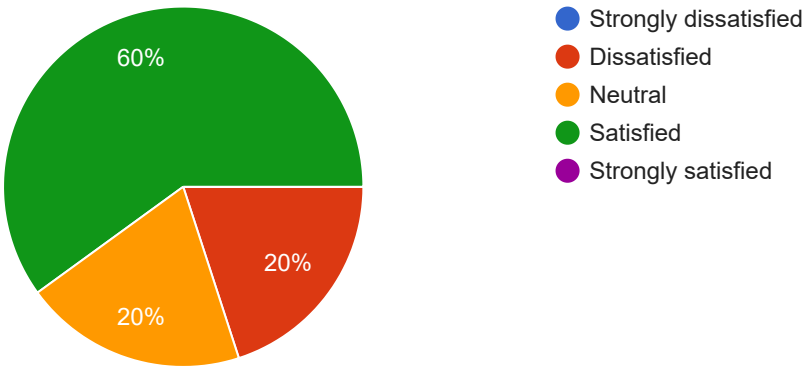
Publish analytics

## Rate your level of agreement with each statement




## How do you feel regarding this concept?

5 responses



## Any comments about this concept?

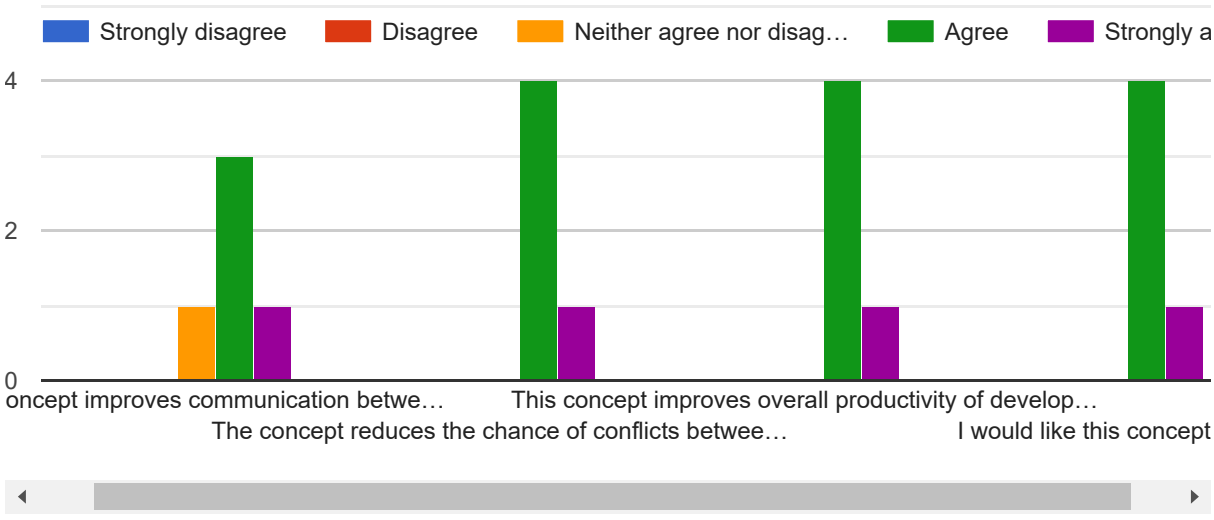
3 responses

Knowing exactly where the user is working doesn't not reduce the amount of conflicts. A history or (*) mark where changes have been made would be more useful. The call system, I don't think it's necessary as teams already have a better tool to communicate, either Slack or Teams for example.
The highlight systems works well and it's very clear to the user where the team is working on

Looking forwards to see this in the future, in my opinion it still needs some iterations to provide more valuable feedback, but it's going in an interesting direction.

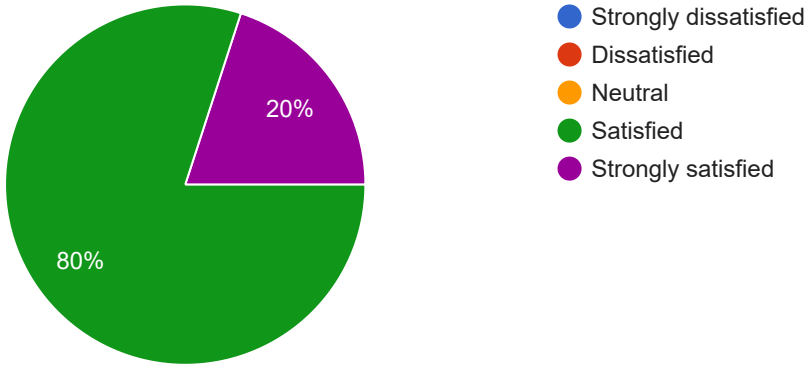
I have some fears regarding Service Studio crashes with this kind of features, also think that some points need more detail but it's getting very nice and useful

## Rate your level of agreement with each statement



## How do you feel regarding this concept?

5 responses




## Any comments about this concept?

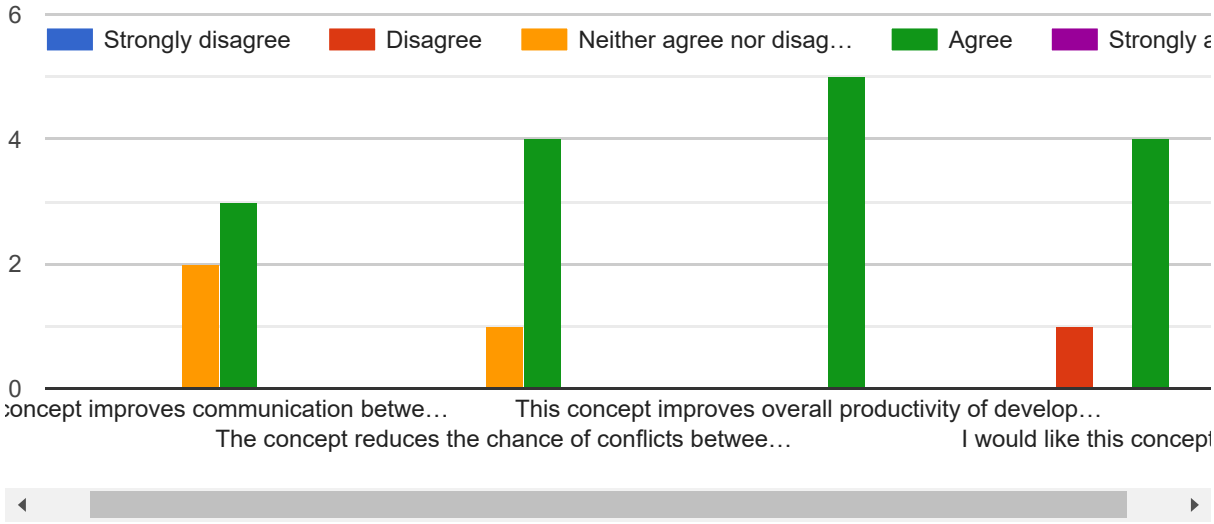
3 responses

I really like this concept as it gives you a clear idea where changes have been made and in what specific place. The yellow blur is not clear, not enough contrasts (maybe not even accessibility complaint), maybe use the highlight from the first Test.
For a first release of this feature, knowing exactly where the user published is awesome but to much, for First Value I would like to know which action/screen/block was changed instead knowing that that specific aggregate was changed. If we go to the UI level, knowing exactly where it was changed does not reduce the amount of conflicts in case I also did changes there, but if I knew changes were made beforehand, I would wait for my colleague to be done before doing changes myself.

Same as before, in my opinion it still needs some iterations to provide more valuable feedback, but it's going in an interesting direction.

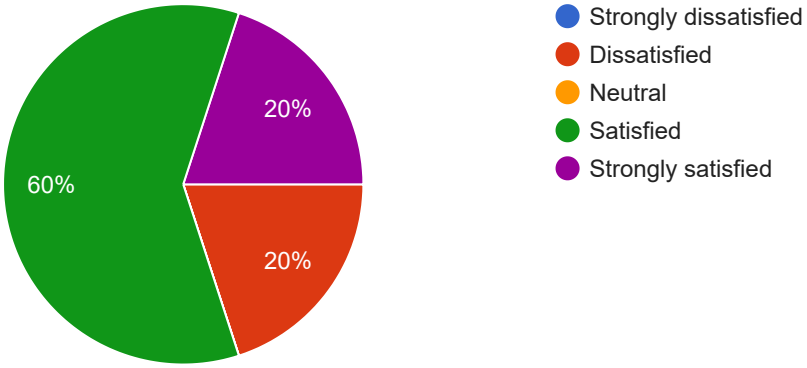
Need some improvements in order to get the detail of the change and clear the highlight

## Rate your level of agreement with each statement




## How do you feel regarding this concept?

5 responses



## Any comments about this concept?

3 responses

This concept still needs to be fine tuned, receiving live updates on my version it's odd as I lose control of my code and a lot of questions are raised in case the changes that my colleague did had errors.
What I like about this feature is knowing that changes were made before I publish, so I don't get caught off guard in case of big conflicts

Looks good from what I saw. I feel it's closer to final version, comparing with the previous ones.

Two fears here: to many notifications can reduce productivity the other is related with the "silent" changes can be missed in situations where it shouldn't

## II.2 Second Iteration Survey

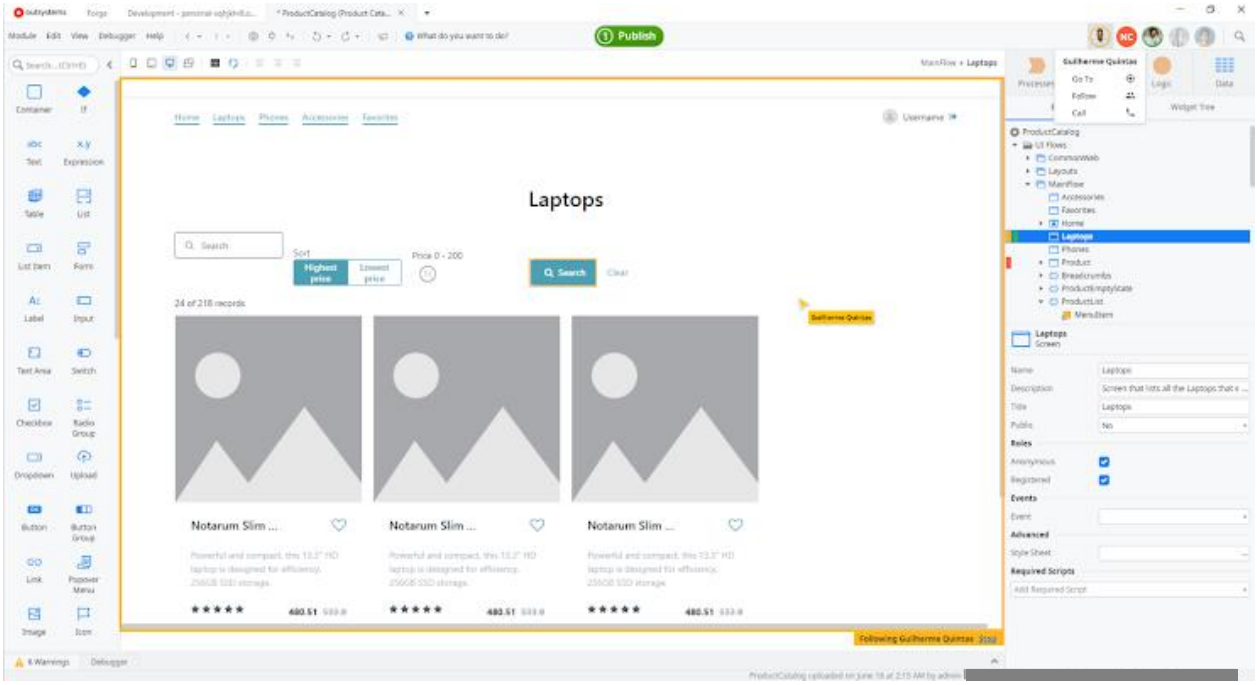
# Real-Time Collaboration Concept Testing Survey

Time to complete: 2 mins

Thank you for being a part of the concept test. To finalize, please fill out this short form.

* Required

## User awareness & follow concept



1. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| This concept improves communication between developers working on the same project | | | | | |
| This concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

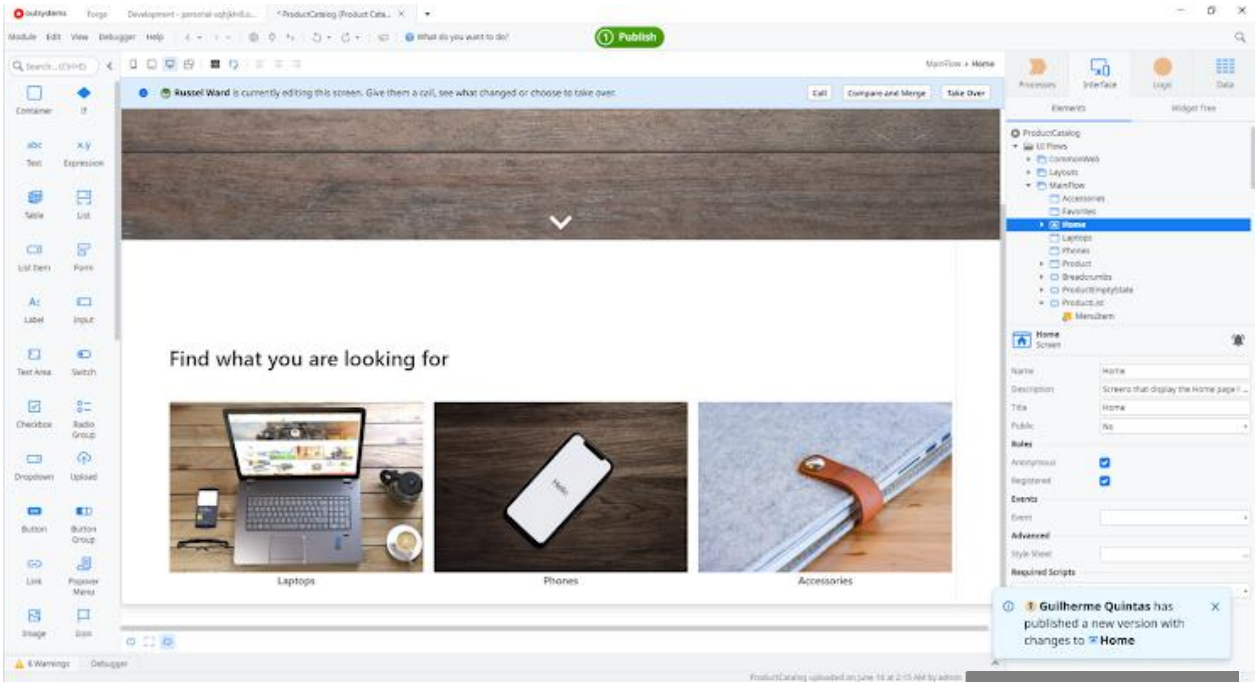
2. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

3. Any comments about this concept?

Exclusive rights to module elements & subscribe to changes concept



4. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| The concept improves communication between developers working on the same project | | | | | |
| The concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

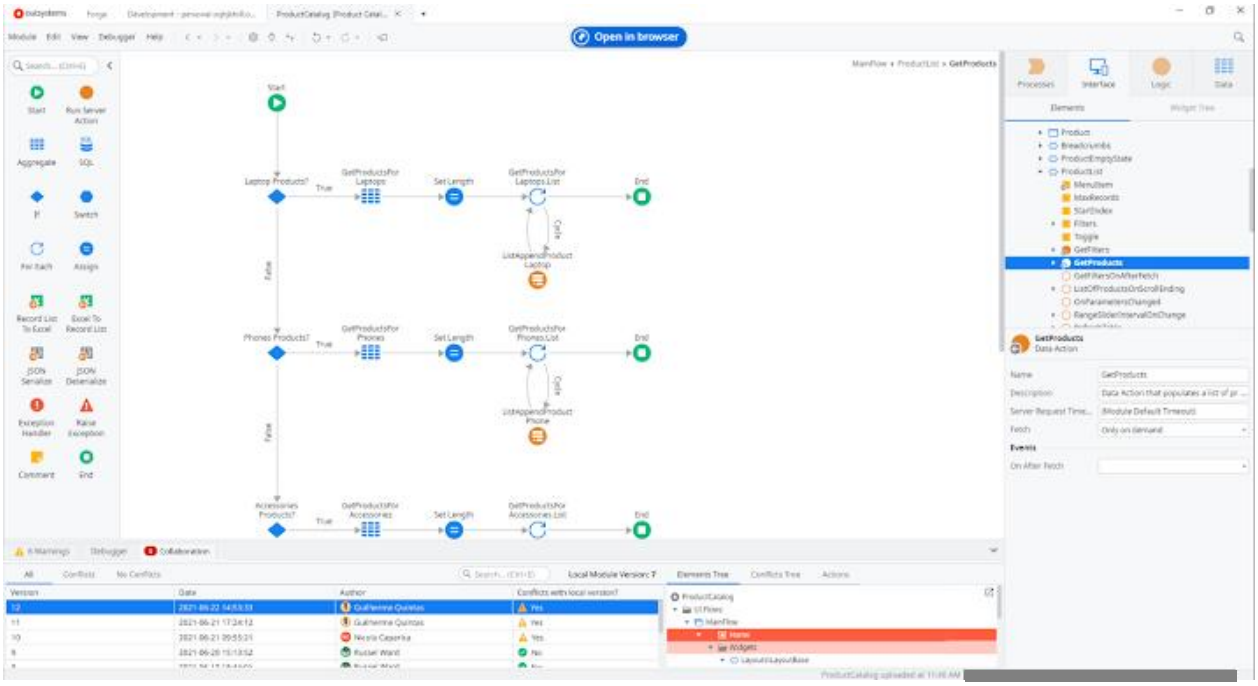
5. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

6. Any comments about this concept?

Collaboration tab with version history concept



7. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| The concept improves communication between developers working on the same project | | | | | |
| The concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

8. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

9. Any comments about this concept?

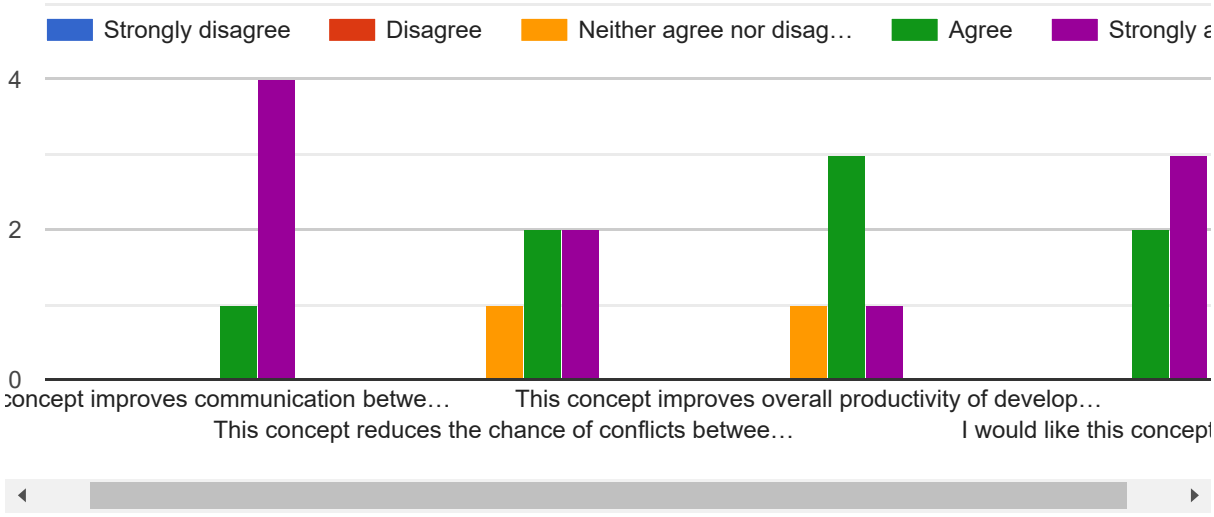
### II.2.1 Results

# Real-Time Collaboration Concept Testing Survey

5 responses

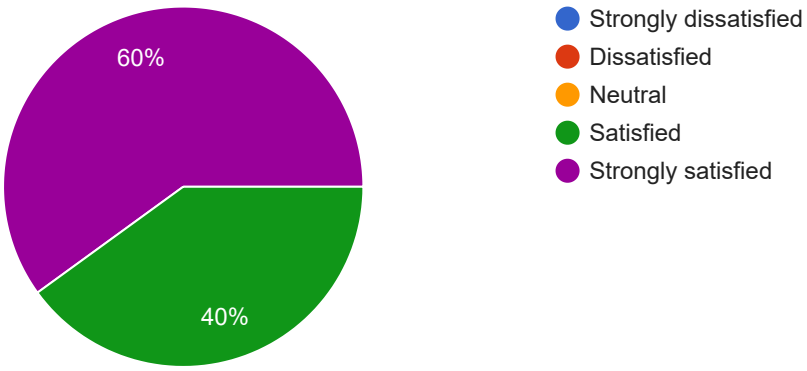
Publish analytics

## Rate your level of agreement with each statement




## How do you feel regarding this concept?

5 responses



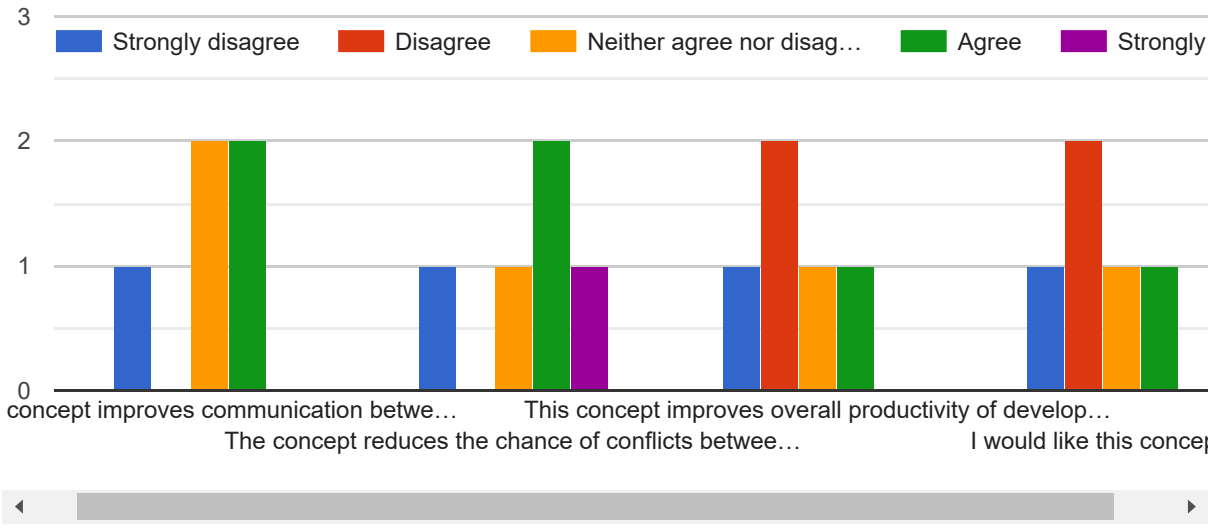
## Any comments about this concept?

2 responses

The possibility to know were each developer is working is something that can really help in big projects with a lot of people always working in the same modules, all the visual signs in the concept can help people to be careful before publishing, preventing code being lost/overridden

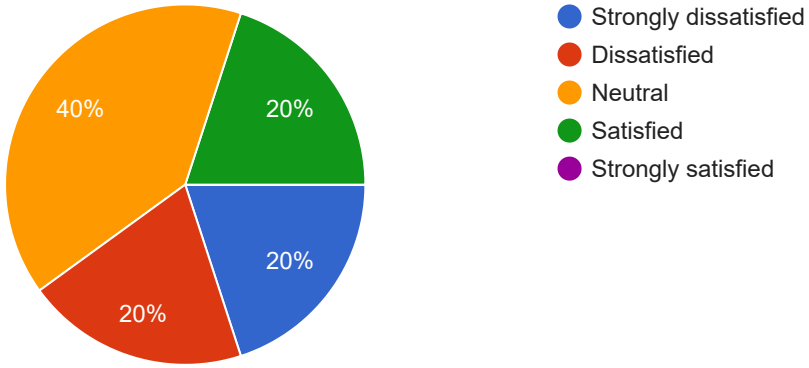
This concept would make our live easier to answer questions such as "where is my colleague working at this moment?" or "can I work on this action / screen or will I be clashing with someone?". Although this would assist in this regard, communication is critical in any project and would see as lack of governance if developers are constantly working on the exact same things.

## Rate your level of agreement with each statement



## How do you feel regarding this concept?

5 responses




## Any comments about this concept?

4 responses

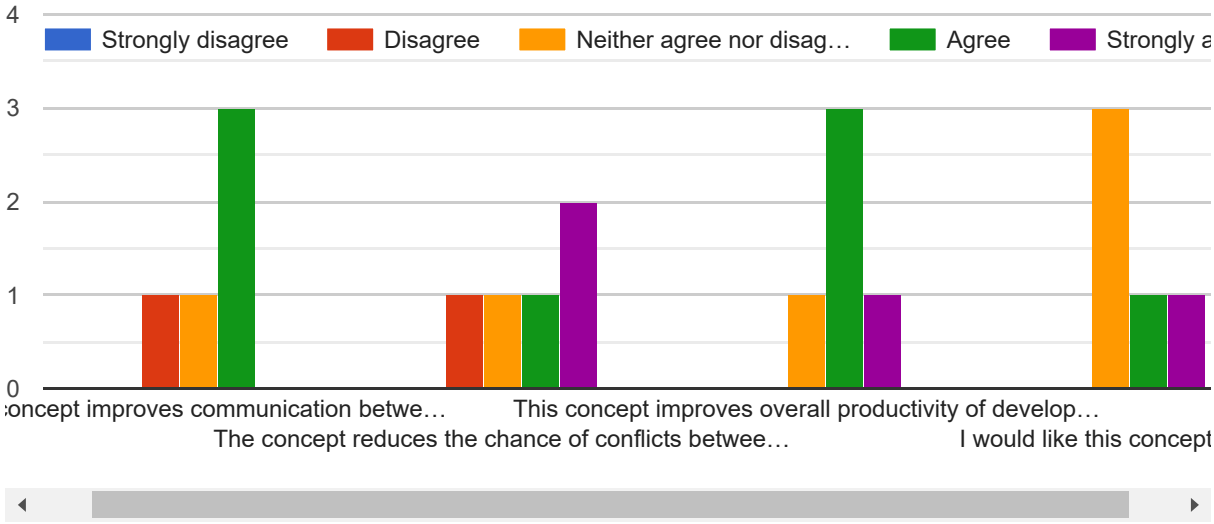
Exclusive rights to module elements is something that I don't want to see implemented, this would prevent people from working in the same screens/blocks, this is a situation that can become a problem but on a team with good communication is something that can happen and speed thing up;
Subscribe to changes concept is a kind of micro management situation, for a developer I don't see a lot of positives, maybe on management situation.

The phrase "Take over" was very confusing to me, I did not relate it to checkout/lock the element to me. Also the Notification bell is very confusing to me, I could not figure out what the purpose was.

It is implicit a lock on a screen / action when one is editing and it will force one user to change at a time. Although good governance and communication, and good partition of code in blocks and actions already avoids developers changing the same exact action or screen, there can be scenarios where both have to change the same thing (e.g. two developers fixing CSS issues of a PoC) where they need to do it, even if at their own risk, and they wouldn't able to do it anymore.

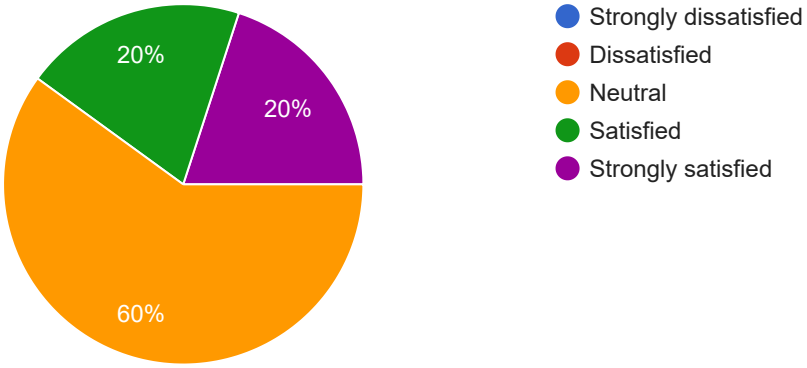
Take over button is a big question on how it would be received.

## Rate your level of agreement with each statement




## How do you feel regarding this concept?

5 responses



## Any comments about this concept?

3 responses

The ability to have the information regarding conflicts available is a huge improvement, right now, a developer can only know this after trying to publish and wait for it to compare with the last published version, this is something that depending on the server/environment can take a lot of time.

I am not convinced about this concept, the tree in always needs to be expanded to be readable. Also, it does not avoid conflicts, it only reports them better. I personally would rather have a right menu option on each element (screen, action, etc.) and select 2 versions (current and other, or 2 other versions) and then have a difference window displayed.

This concept doesn't answer the question "where is my colleague working", it just warns that I should get / merge with the latest version before starting to edit this action. It's also not clear if the changes occurred when a user is editing an action would be visible in another user's session before they are published or only when they are published.

## II.3 Third Iteration Survey

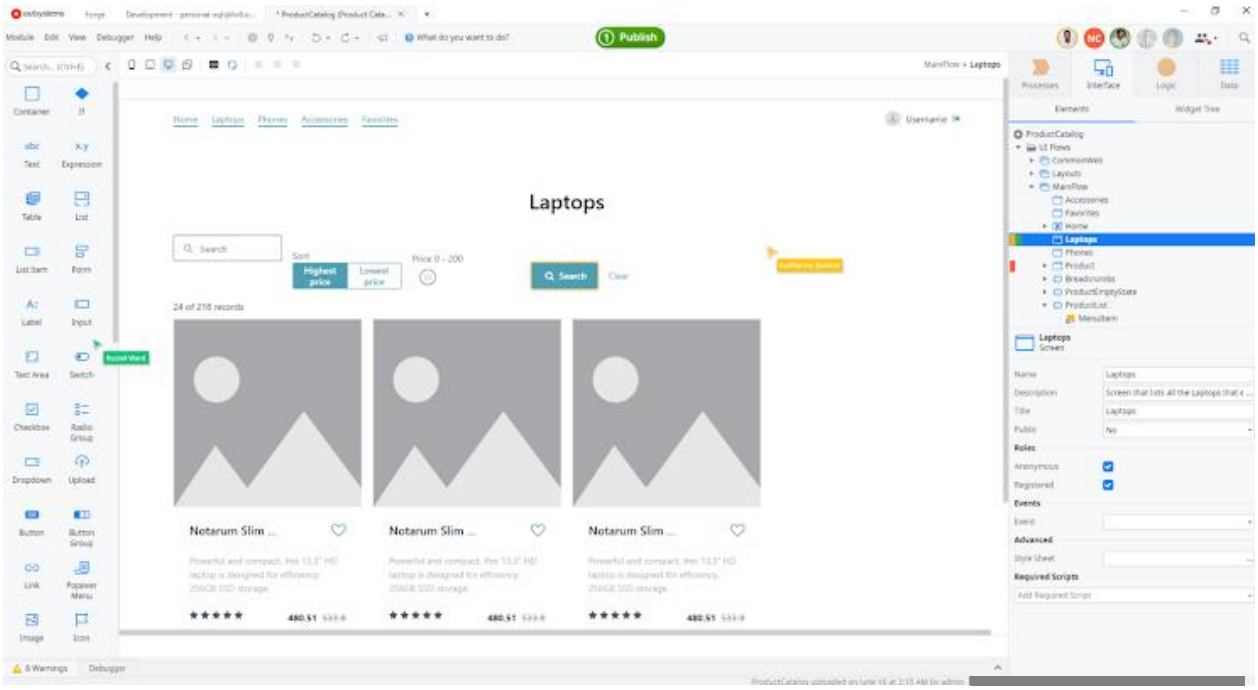
# Real-Time Collaboration Concept Testing Survey

Time to complete: 2 mins

Thank you for being a part of the concept test. To finalize, please fill out this short form.

* Required

Real Time collaboration concept



1. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| This concept improves communication between developers working on the same project | | | | | |
| This concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

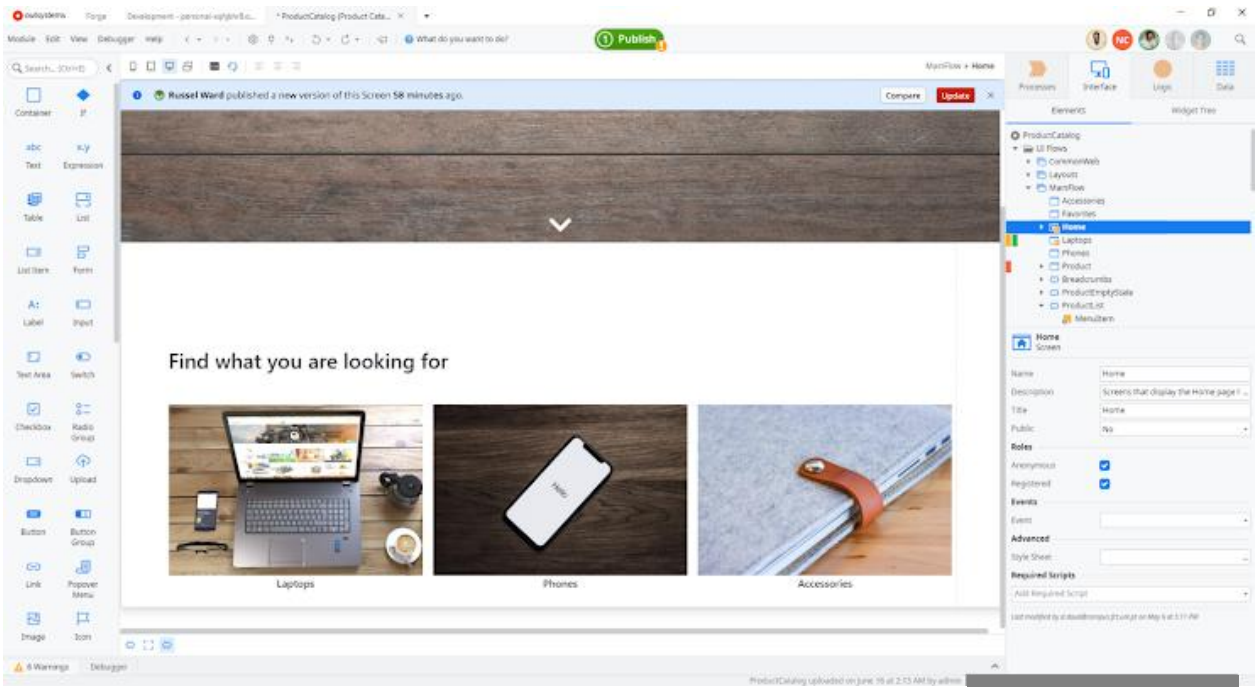
2. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

3. Any comments about this concept?

User awareness concept



4. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| The concept improves communication between developers working on the same project | | | | | |
| The concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

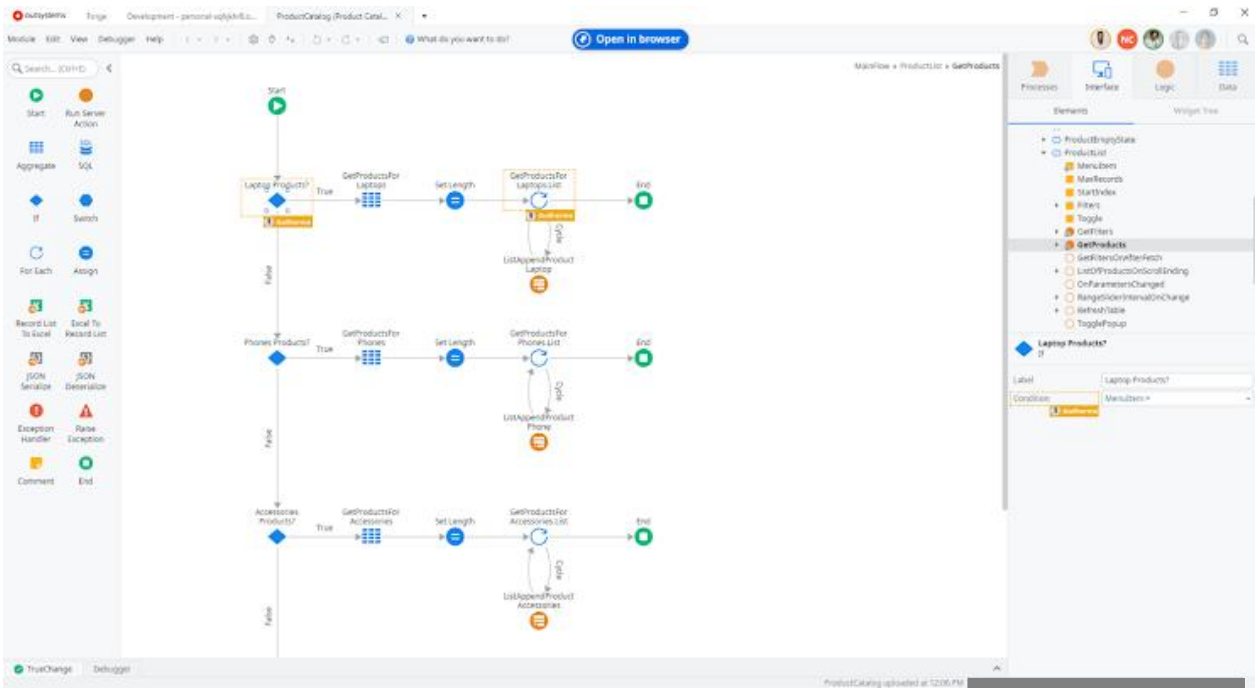
5. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

6. Any comments about this concept?

Locate conflicting changes inside a screen/action concept



7. Rate your level of agreement with each statement *

*Mark only one oval per row.*

| | Strongly disagree | Disagree | Neither agree nor disagree | Agree | Strongly agree |
|---|---|---|---|---|---|
| The concept improves communication between developers working on the same project | | | | | |
| The concept reduces the chance of conflicts between developers working on the same project | | | | | |
| This concept improves overall productivity of developers working on a project | | | | | |
| I would like this concept to be implemented | | | | | |

8. How do you feel regarding this concept? *

*Mark only one oval.*

- Strongly dissatisfied
- Dissatisfied
- Neutral
- Satisfied
- Strongly satisfied

9. Any comments about this concept?

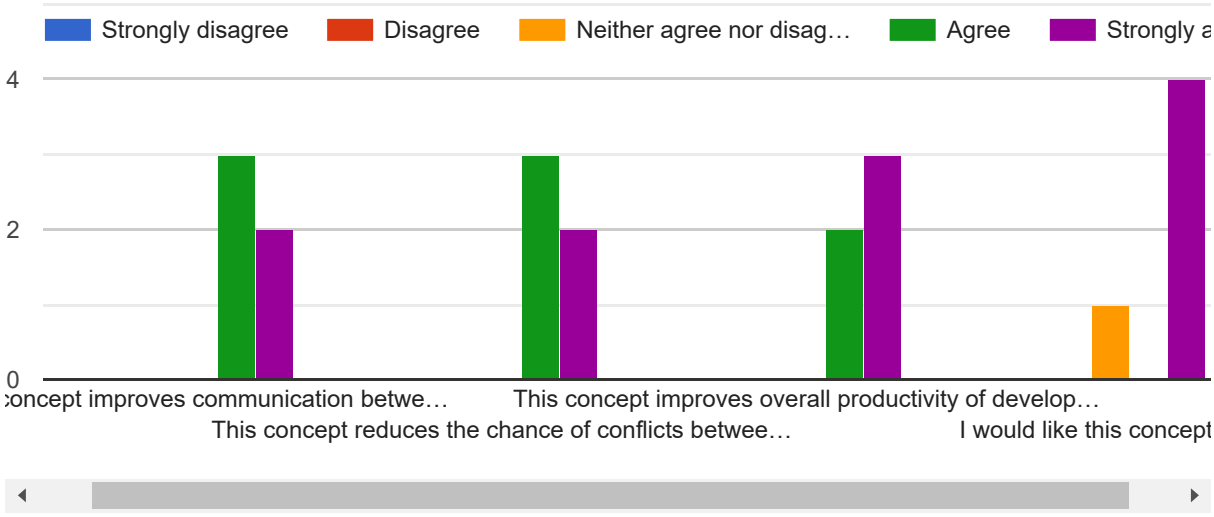
### II.3.1 Results

# Real-Time Collaboration Concept Testing Survey

5 responses

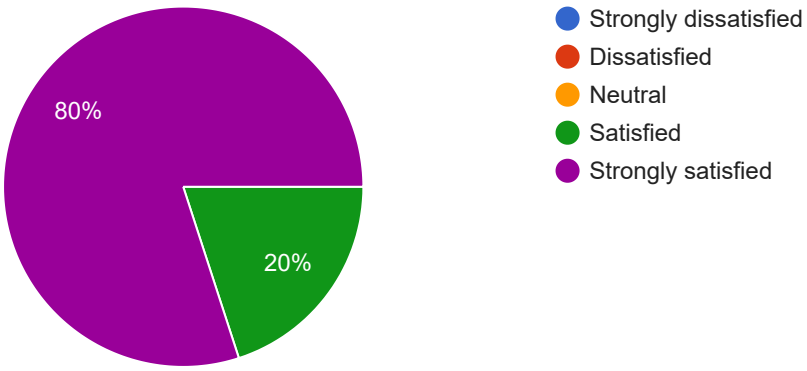
Publish analytics

## Rate your level of agreement with each statement




## How do you feel regarding this concept?

5 responses



## Any comments about this concept?

3 responses

It may help in a lot of scenarios.

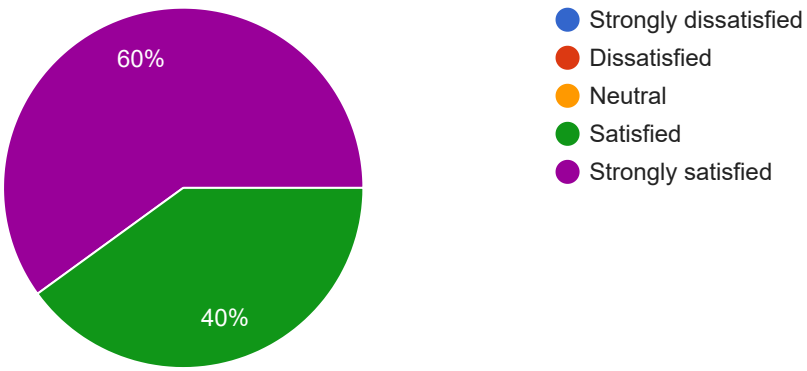
Great ideas!

This is amazing because I would definitely use this for my work to collaborate with other people. I really think this could be a game changer for Outsystems

## Rate your level of agreement with each statement


## How do you feel regarding this concept?

5 responses



## Any comments about this concept?

2 responses

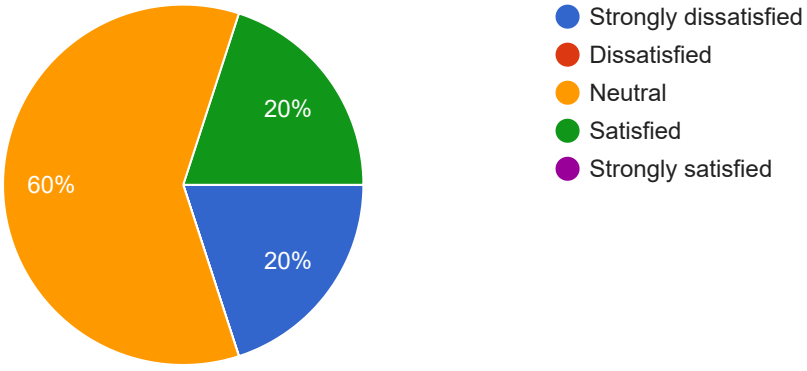
Has a different perspective from the first option, but it's also a nice to have, both are useful together.

With this we could finally have some awareness when working!

## Rate your level of agreement with each statement


## How do you feel regarding this concept?

5 responses



Any comments about this concept?

2 responses

I don't see any value on this concept. At least for my use cases.

I do not see the value in this