applied sciences

MDPI

*Article*

# Non-Programmers Composing Software Services: A Confirmatory Study of the Mental Models and Design Challenges

**Abdallah Namoun** [1,*], **Ali Owrak** [2] **and Nikolay Mehandjiev** [2]

[1] Faculty of Computer and Information Systems, Islamic University of Madinah, Medina 42351, Saudi Arabia
[2] Alliance Manchester Business School, The University of Manchester, Booth Street West, Manchester M15 6PB, UK; ali.owrak@manchester.ac.uk (A.O.); n.mehandjiev@manchester.ac.uk (N.M.)
* Correspondence: a.namoun@iu.edu.sa; Tel.: +966-59-8009417

Received: 20 September 2019; Accepted: 4 November 2019; Published: 17 December 2019

**Featured Application: Composing software services (mashups) enables the creation of diverse new applications, yet it is a daunting activity for users who are not programmers by profession. A confirmatory study of ordinary users' service compositional activities was undertaken, resulting in a set of recommendations to guide the creation of usable tools supporting service composition. For instance, service designers are advised to use natural metaphors to represent complex logic such as conditional statements, dataflow, and parallel events between services to aid user understanding.**

**Abstract:** Ordinary web users can now create and publish online content. They even venture into "mashups," integrating information from different sources into a composite information-providing web service. This is a non-trivial design task, which falls into the area of end-user development when the ordinary users who perform it do not have programming education. In this article, we investigate the service design strategies of 12 such ordinary users and compare them against the baseline of 12 programmers. In our think-aloud study, users completed two contrasting types of tasks involved in developing service-based applications: (a) manual service composition and (b) parametric design using templates with a high degree of software support (or assisted composition). These service composition tasks were chosen to differ in respect to the level of user support provided by the tool. Our findings show that non-programmers liked, more than programmers, the template-based parametric design and did not find the tool assistance as constraining as the programmers did. The difficulty of design involved in manual service composition and the absence of user guidance hindered non-programmers in expressing and implementing accurate design solutions. The differences in the mental models and needs of non-programmers are established to be in stark contrast to those of programmers. We used the details of our findings to propose specialized design recommendations for service composition tools aligned with the profiles of their target users.

**Keywords:** service composition; mashups; non-programmers; programmers; service-oriented architecture; end-user development; design challenges; mental model; design strategies; recommendations

## 1. Introduction and Motivation

Existing service research efforts have focused primarily on advancing concepts, principles, languages, and methodologies of service-oriented architecture (SOA) [1]. Whilst the significance of such technological innovations is undeniable, their successful use within the information society requires understanding the motivations, needs, and traits that characterize end-users of the SOA technologies.

Using current SOA tools and methodologies requires strong technical and problem-solving skills [2], making them suitable for skillful programmers only. However, most of the users who engage in some sort of programming activity are not trained as programmers. For example, past projections revealed that nearly 13 million of the 90 million computer users in American workplaces were estimated to be involved in some sort of programming activities, whilst the number of trained programmers was only 3 million [3]. Other studies also suggest that at least 73% of professionals without programming experience are engaged in at least one activity, which amounts to programming [4].

This also applies to software services; for example, non-programmers can bring together information from different sources to search for a house within certain distance from a park, or to choose different modes of commuting to work depending on the rain forecast. At the same time, software services are more complex than other end-user development domains such as spreadsheets or home automation because of the diversity of interfaces, parameter types, and underlying implementation and deployment architectures. This implies there is an urgent need to simplify and disseminate the use of SOA technologies to a wider audience whose primary role is not software development.

Whilst it is widely accepted that end-users have differing characteristics, skills, and practices from professional programmers [5], little is known regarding how they reason about service-composition activities. Ko et al. [5] claim that professional programmers and end-user programmers encounter many similar software development challenges. Yet no systematic study has been performed to verify this claim in the domain of service composition. In our view, examining the decision-making strategies non-programmers adopt to compensate for their lack of technical knowledge and comparing them to a baseline expert sample would enable the understanding of end-users' strengths and weaknesses. This could lead to the creation of highly-dedicated methodologies and design guidelines, upon which service-development can be founded for a pleasant user experience.

This study investigates the mental models of non-programmers in relation to software-service-development activities and compares it to a baseline sample of professional programmers. Norman [6] defines a mental model as a person's mental representation of the real-world systems; thus, how a person perceives and uses these systems and makes decisions. In particular, our study investigated the conceptual challenges non-programmers face during service composition and how they approach two differing ways of creating service-based applications: (a) manual service composition (work-flow based) and (b) parametric design using templates with high degree of software support (assisted composition). The design tasks were chosen to differ in respect to the level of user support provided by the tools (Section 3).

### *Key Contributions*

The contributions of this study towards software service research are twofold.

Firstly, it provides a better understanding of non-programmers' mental models regarding software service composition, service design challenges, and problem-solving strategies (Sections 4 and 5). Currently, the literature contains a limited number of users' studies that employ empirical evaluations to understand the challenges and alternative decision-making strategies non-programmers adopt when indulged in service composition to compensate for their lack of technical knowledge and expertise, and report on their overall experiences and needs. Our results identify the gap between non-programmers' and programmers' mental models with respect to two prominent approaches to composing software services; e.g., workflow-based service composition [7] and artificial intelligence-based service composition [8].

Secondly, it proposes a set of practical guidelines for designing tools to support end-user service composition, which are based on our first contribution above (Section 6). These are specialized guidelines to tackle the emerging conceptual problems and facilitate different aspects of service composition for non-programmers. This contribution agrees with the recommendation of Boshernitsan et al. [9] to align development environments with developers' mental model of programming structures to accommodate for the rapidly changing requirements.

Throughout this article, we use the terms 'non-programmers' and 'programmers' to denote our two user groups. The term 'non-programmers' refers to people who do not have a software development-related education, nor do they have programming experience, and the activity of programming is fairly new to them. The term 'programmers' refers to people who have a software development-related education (at degree or to a professional level) or who have considerable programming and software development experience without a formal education. The latter group will act as an expert sample against which all non-programmers' composition activities and strategies will be compared.

This article poses two research questions focusing on the manual composition and assisted, template-based composition of software services by non-programmers. Next, educated hypotheses were generated from the findings of previous user-based studies of service and mashups composition.

**RQ1:** *What type of design challenges are faced by non-programmers when composing services manually using a workflow-based composition approach that offers minimal tool support?*

Due to their low level of programming skills, it was expected that non-programmers would find it challenging to understand service particulars [10–14], and define conditional statements [15] and dataflow connections between services [10–13,16–18]. In contrast, programmers were anticipated to complete these manual composition tasks more accurately because of their strong mental model of programming concepts. Consequently, our hypotheses pertaining to manual composition are as follows:

**Hypothesis 1a (H1a).** *Non-programmers will find it more difficult to understand service terminologies compared to programmers.*

**Hypothesis 1b (H1b).** *Non-programmers will find it more difficult to define conditional flows (IF ... ELSE) between services compared to programmers.*

**Hypothesis 1c (H1c).** *Non-programmers will find it more difficult to create dataflow connections between services compared to programmers.*

**Hypothesis 1d (H1d).** *Non-programmers will face more challenges with manual service composition compared to programmers.*

**Hypothesis 1e (H1e).** *Non-programmers will hold a more negative perception about manual composition compared to programmers.*

**RQ2:** *What are the attitudes of non-programmers when a software tool is "taking over" their design by abstracting technical details and advising them about consequences of their choices?*

Due to their poor software development experience, non-programmers should regard assisted composition (i.e., assisted template-based design) as easy to use and satisfying [19–22]. This is because it requires little cognitive effort to operate and is close to their mental models. However, programmers are expected to regard assisted composition as inflexible, constraining, and lacking power to express complex application logic [14,18]. Consequently, our hypotheses pertaining to assisted composition are as follows:

**Hypothesis 2a (H2a).** *Non-programmers will hold a more positive perception about assisted composition compared to programmers.*

**Hypothesis 2b (H2b).** *Non-programmers will favour assisted composition over manual composition compared to programmers.*

The literature provides conflicting results with respect to creation of service-oriented compositions by non-programmers and programmers. For instance, several studies [10–14] reported major challenges during the creation of service-oriented compositions by ordinary end-users. However, recent studies show that non-programmers provide the same service composition performance as programmers [18–23]. These somewhat strange claims motivate our research to (1) test these claims and (2) understand what makes service composition a really challenging task. To answer the posited research questions, non-programmers' performances, mental models, and service composition techniques are contrasted to those of a baseline sample of programmers to identify gaps in their conceptual understanding and any alternative decision-making strategies they adopt to compensate for the lack of technical knowledge and expertise (Sections 4 and 5). Therefore, will the non-programmers create equally accurate service compositions as the programmers? And which composition issues do they normally face?

## 2. Related Works

### *2.1. End-User Development*

End-user development (EUD) is a research field concerned with creating tools and methodologies aimed at empowering end-users, who are not software professionals, to create or customize software applications that fulfil their personal and business needs [24]. Nardi [25] differentiates between end-user programmers and professional programmers in the sense that the former create software that help them accomplish tasks on their primary work. The field of EUD has steadily grown over the last decade, providing various techniques [26] and design guidelines [27]; however, most of EUD studies have focused on the development of spreadsheets, databases, and web and Internet of Things applications [28–30]. The domain of software service composition is substantially different of these domains because it is agnostic about the application domain, and is, thus, dominated by the technical details of services, their connections, underlying operating architectures, and providers' access details. Lessons learned about EUD on spreadsheets and databases, therefore, cannot be transferred easily to the domain of services.

Numerous studies have already discussed the motivations and barriers for engaging in end-user development activities. Sutcliffe [31] and Mehandjiev et al. [4], for example, propose a general framework that links motivation to perform EUD to, firstly, perceived reward, and then, to incurred cost. In web development, McGill and Klisc [32] argue that end-user web developers are aware of the associated risks and benefits and it is crucial to involve them in the development of approaches to minimize risks. In software-service development, Schulte et al. [33] surveyed 52 German banks, showing that adopting service-oriented technologies for collaboration is likely to produce more benefits (e.g., cost reduction) than risks (e.g., loss of autonomy). Namoun et al. [34] compared the perceived benefits (e.g., reusability) and risks (e.g., personal privacy, technical complexity) of service development by end-users and IT professionals. Whilst both groups were highly interested in service composition and were concerned about privacy and security of their data, the IT-professionals exhibited a more complex and comprehensive mental model of service concepts. A strand of EUD research, which is not central to our study, explores gender differences in respect to user perception and actual uptake of EUD activities [35]. The natural programming approach is another strand that tries to understand how non-developers carry out programmable tasks before exposure to programming using merely a pen and paper, such as [16].

### *2.2. Web Service Composition and Mashups Development*

The service composition field is a concerned with connecting atomic, self-contained, modular services into a composite service, and creating tools and languages that facilitate business applications' integration [36]. Typically, this process is very complex and daunting [36]. Service composition includes four key phases; namely, definition, selection, deployment, and execution [37]. The definition phase creates the abstract composition process model, which specifies the requirements, including activities, data flow, and control flow dependencies. The selection phase involves choosing the appropriate services to satisfy the requirements of each activity of the composition. The deployment phase involves deploying the composite service to enable end-users to utilize the service functionalities. The execution phase involves the execution of the composite service instance.

Owing to the complexity of traditional, syntax-based service composition languages [2,36], e.g., BPML, BPEL4WS, and WSCDL, research efforts have transformed these languages into visual languages and representations that are more user-friendly, easier to use, and less error-prone [38,39]. However, such languages are still targeted at experienced software developers [26,40]. To make service composition accessible to non-programmers, graphical tools, e.g., ServFace Builder [41], were developed based on Human Computer Interaction concepts, such as "direct manipulation of objects" and "what you see is what you get," whereby end-users specify programmable actions and behaviour in a purely graphical manner without going into arcane syntax. This approach is referred to as visual programming. In SOA, composition at the presentation layer is gaining momentum, where composite applications are integrated using merely their user interfaces [42].

The importance and benefits of SOA for software development are well recognized [1]. However, efforts to simplify its use for a wider audience, especially among non-programmers, are still timid. Namoun et al. [11,34] explored, through a set of user studies, the issues non-programmers face when performing service composition tasks. Major issues reported included: comprehending and specifying dataflow connections, linking between design time actions and runtime results, and understanding technical jargon. De Angeli et al. [17] conducted a contextual enquiry with 10 accountants to elicit requirements for the Wisdom-Aware development tool, developed to propose interactive recommendations learnt from existing compositions made by professional developers to prospective end-users. Results revealed that help is most effective when received from technical experts and automatic/contextual help was preferred over on-request help. Despite these efforts, several service-development challenges are still being unraveled, such as the difference between programmers and non-programmers design strategies.

Mashups are web applications developed by wiring information, in the form of content, application logic, and user interface components, from different sources into a unified view [43]. The continuous increase of mashups and their APIs has encouraged the development of various mashup development tools, such as Microsoft Pofly, Yahoo! Pipes, and Intel Mash Maker [44]. However, most of these tools did not follow a user-centric design approach, nor were they evaluated appropriately with the targeted end-users. Mashup development tools and approaches for creating mashups are published elsewhere [43,44].

A few attempts have tried to identify the issues end-users face when mashing up various sources of information. Jones and Churchill [45] analysed forum conversations of end-users of Yahoo! Pipes mashup tool to understand the obstacles they face and their collaborative debugging strategies. Two types of engagement were observed: core engagement where users actively asked and answered questions, and peripheral engagement were users read but did not post content. In a think-aloud study, Kuttel et al. [46] showed the usefulness of a versioning extension for Yahoo! Pipes. However, the majority of the participants were computer scientists. Wong and Hong [47] evaluated Marmite, a mashup tool for aggregating web content, with six end-users, and showed difficulty in understanding the dataflow concept, terminology, and operators. To gain further insights into the way 10 non-IT end-users create web mashups using Microsoft Popfly, Cao et al. [48] applied Schön's reflection-in-action framework and the notion of ideations from creativity literature, revealing that the design process and programming process are highly-intertwined.

Zang et al. [49] surveyed 63 mashup developers about their mashup creation experience, with the majority creating map mashups using Google Maps APIs. The developers highlighted lack of reliability of the API and documentation, and need for coding skills as the primary determent to developing mashups. In another survey study, Zang and Rosson [50] explored the requirements for

Yahoo! Pipes mashup development by web-active end-users, and highlighted that personal interest and perceived usefulness of mashups are more important than the perceived difficulty of developing mashups. Al Sarraj and Troyer [51] evaluated the usability of three mashup development tools (i.e., Yahoo! Pipes, Open Mashups Studio, and Dapper DA) with a total of 24 IT and non-IT people by applying the framework of cognitive dimensions. The results showed superior mashup performance by the IT participants.

Daniel et al. [52] proposed a domain specific approach, i.e., evaluation of research outcomes, to reduce the complexity of mashup integration. That approach attempts to overcome the generic nature of existing mashup development tools. Instead, the approach focuses on the logic that specifies domain specific functionalities as well as domain data and relevant meta-models. The user testing of this approach, however, provided little detail about the approach and challenges encountered faced during the mashup experience. Radeck et al. [53] described CapView, a visual mashup tool that can be used by non-programmers to integrate various web resources into a functional mashup application. In this approach, short, natural language annotations and distinct visual representations (i.e., colours) are used to describe the main functionality of each mashup component to improve non-programmers' technical understanding of the development process.

Aghaee et al. [54] and Aghaee and Pautasso [55] discussed the design and evaluation of NaturalMash, a mashup tool that combines a controlled natural language and the What You See Is What You Get (i.e., WYSIWYG) development approach to enable non-programmers to assume interactive mashup integration. Following the methodology proposed by Namoun et al. [11], the tool employed an iterative, user-centered design approach. The key features introduced in the tool that attempt to lower programming barriers are proactive feedback, runtime view, auto completion of compositions, and service discovery using natural language [54,56]. However, the authors did not reason through the strategies and challenges non-programmers encountered. Instead, they mainly focused on the design of the tool itself.

### 2.3. Mental Models

Mental models refer to users' views and understandings of how a system functions and how its parts are connected [6]. Mental models directly affect our daily decision making. There is evidence that non-programmers exhibit an incomplete mental model with misconceptions about service composition [11]. Intrinsically, end-users also lack interest in programming activities [57] and have low levels of self-efficacy, which refers to one's confidence in one's ability to complete a particular task [58]. Moreover, programming tasks largely depend on problem-solving skills to be solved. However, Loksa et al. [59] argue, through a controlled experiment, that explicit guidance improves productivity and programming self-efficacy. De Raadt et al. [60] suggested that expert programmers rely on a tacit set of previous strategies to extract goals from programming problems and apply plans to fulfil these goals.

### 2.4. The Research Gap

Making service composition easy is still a challenging endeavor [61,62], requiring further research to understand the obstacles encountered by end-users when composing services. Two recent surveys of service composition by non-programmers revealed insightful motivations for carrying out our research [19,63]. Barricelli et al. [63] surveyed more than 165 EUD papers, spanning from 2000 until 2017, and found that approximately 50% of those papers used the component-based technique to build software. In this context, components refer to modular units of software, such as web services and mashups. More importantly, Barricelli et al. [63] suggested, as a primary research direction, to conduct comparative studies of EUD approaches and highlighted the need for further research to investigate EUD activities so as to guide the design of future tools.

The second survey on user studies evaluated tools and approaches of service composition and showed their shortcomings with respect to the evaluation methodologies used, which subjected the findings to scientific scrutiny [19]. The coverage spanned from 2005 until 2018, yet only 47 end-user studies of service composition were reported. Moreover, only 23% of the surveyed EUD studies were confirmatory in nature, necessitating additional confirmatory studies to test the previous research claims.

Contrary to the common belief, recent user studies [18,23] showed that the technical background of users did not improve service composition performance and perception of ease of use. In a bid to eliminate programming difficulty, Cheng et al. [23] described a lightweight service mashup platform for creating telecommunication mashup-based compositions easily through a web browser where data and event flows could be defined. The approach was based on various standardised models, such as Service Data Model, Service Creation Model, and Service Execution Model. The evaluation of this composition platform showed no difference in the composition scores between programmers and non-programmers. Additionally, there have been other inconclusive observations about the differences in service composition between technical and non-technical users [12].

Moreover, there are various UI-based approaches aimed at helping end-users to compose services and objects in an easier manner, but the results are still inconclusive. Akiki et al. [64], for instance, presented a jigsaw puzzle approach to create transformations and links between services and Internet of Things objects for non-programmers. These transformations form the basis of executable workflows which are generated automatically. Evaluation results showed that non-programmers were able to define the compositions successfully. However, Cappiello et al. [18] described a WYSIWYG UI-based composition platform to glue multiple services together. In this approach, UI components are used as the building blocks of the mashups. Surprisingly, the non-programmers' performance was not statistically worse than the programmers with respect to a variety of composition tasks. Both groups also showed similar perception of ease of use towards the UI-based approach.

Trigger Action programming frameworks, such as IFTTT and Zapier, are gaining rapid popularity [65]. However, striking a balance between expressivity and usability is still an open issue for these types of frameworks. To that end, Corno et al. [15] proposed a recommender system to suggest trigger-action rules to the end-user during their mashups and service composition. Yet the efficiency of creating conditional processes by end-users has still not been investigated. Strikingly, only three service composition studies have attempted to discover the conceptual issues within composition approaches [19]. This calls for further research efforts.

This research comes as an attempt to elaborate and conduct a sound study to test and confirm the existing service composition observations. Our user study conforms to the guidelines of the evaluation framework proposed by Zhao et al. [19], ensuring higher validity of the results. The study compared the performance of two user groups with a focus on two compositional approaches (manual and flow-based versus assisted and form based) rather than the tools themselves. Both the flow based and form-based composition approaches are the least studied forms of composition approaches [19]. Moreover, understandability has been studied in explanatory studies only and is the least issue to be studied in EUD studies. Finally, most the of the studies focus on identifying the usability issues instead of the trying to understand the underlying root causes for the complexity of service composition. In this study, we go beyond the perceived usability of tools by shedding light on the conceptual issues of service composition.

## 3. Tools and Methods

### *3.1. Sampling Technique*

Purposive sampling was used to select our participants. It is a non-probability sampling approach that does not involve random selection of participants [66]. Instead, participants are evaluated and selected to match the characteristics of a specific target group. Our primary selection criterion was programming education and software development experience, leading to two homogeneous groups: non-programmers and programmers. One shared requirement for both groups was the need to be familiar with the domain of the tasks to be completed using the SOA4All tools. We selected a "Student Registration" process, and for this we targeted students and staff from the University of Manchester. This targeted group represents the current and future business workforce that will potentially be challenged with the task of composing services. We provided incentives to reduce self-selecting bias. Initially, a web-based screening questionnaire was distributed to University mailing lists to collect technical background information of potential participants. This questionnaire measured participants' programming/development experience, service modelling experience, knowledge of software development environments and modelling tools, and general demographic information (see Appendix A).

## *3.2. Participant Profile*

From over 200 respondents, we selected a well-differentiated sample totalling 24 students to take part in our study. Among those, 12 participants (7 males and 5 females) came from non-computer science disciplines, such as business and management, international human resource management, marketing, and managerial psychology, and all had no previous programming or service modelling experience. In contrast, the other 12 participants (9 males and 3 females) came from computer science disciplines and had extensive programming and service-development experience using a variety of programming languages, such as Java, C, and C++, and software development environments, such as Visual Studio, NET, NetBeans, and Eclipse. These participants constituted the expert sample against which all design activities and outcomes were compared.

Participants rated their software development background and experience on a 5-point Likert scale, where 1 signifies 'Extremely poor', 3 signifies 'Average', and 5 signifies 'Excellent'. All background scales showed excellent reliability (Cronbach alpha > 0.92). We calculated an average index of the background questions and performed an independent-samples *t*-test to assert differences between non-programmers' and programmers' development experiences. Results showed a statistically significant difference between users' development skills and experience depending on their background, $t$ (21) = 4.79, $p < 0.001$. Non-programmers' software development experience and service composition skills (M = 1.67, SD = 0.30) averaged significantly lower than programmers' (M = 3.33, SD = 0.18) for all measures, as depicted in Figure 1.

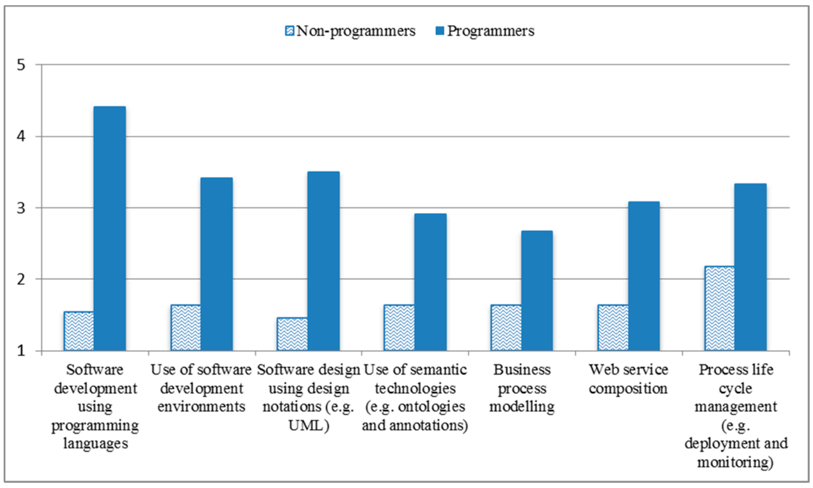
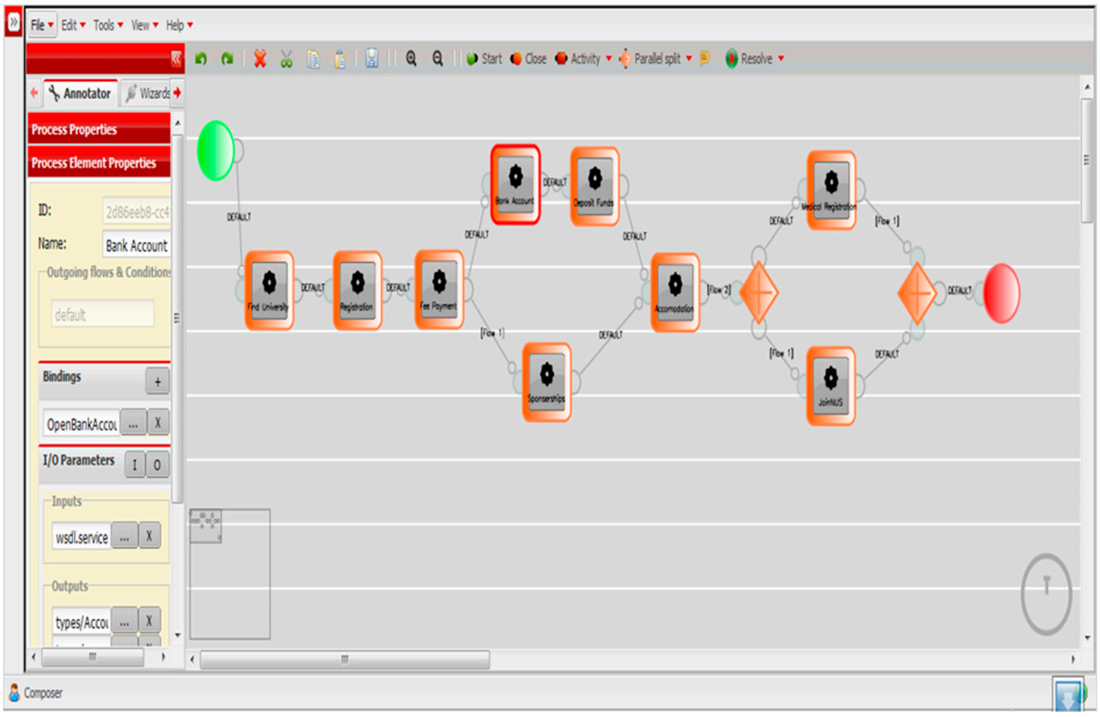


**Figure 1.** Software development experience of our sample.

Additionally, non-programmers' interest ($t$ (21) = 5.55, $p < 0.001$) and likelihood ($t$ (21) = 5.26, $p < 0.001$) to develop applications in the future were rated significantly lower than programmers' (M = 2.63 versus M = 4.5 and M = 1.90 versus M = 4.33 respectively).

### 3.3. SOA4all Service Composition Tools

SOA4All Studio, produced by the SOA4All project, is an integrated visual service-development environment, aiming to support the building of service-based applications without coding. The studio offers a number of complex tools covering the full service-development lifecycle, including service discovery, annotation, composition, execution, and monitoring. However, this study focused on the composition aspect only, and to this end the participants used (1) the Process Editor tool (as in Figure 2) and (2) the User Assisted Service Composition tool (as in Figure 3) of the SOA4All Studio.

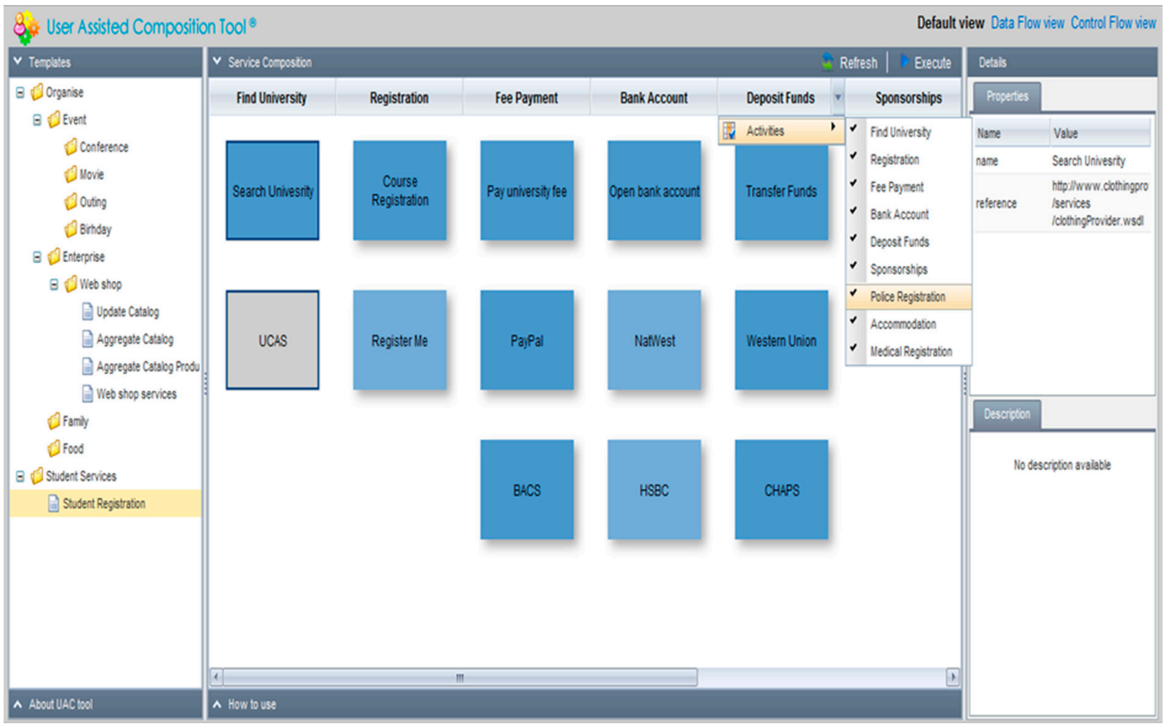


ID: 2d86eeb8-cc4
Name: Bank Account
Outgoing flows & Conditions
default
OpenBankAccou
wsdl.service
types/Accou

**Figure 2.** SOA4All process editor.

Process Editor is a service composition tool, which empowers users to model business processes visually using annotated services in pursuit of their goals without writing programming code or focusing on low-level technical details. We refer to this approach as manual service composition, where little design support is offered to the end-user by the tool. This involves representing and specifying the flow of a particular process, e.g., student registration (Section 3.4.2), which is a set of collected activities or tasks, with a notation. In the process editor, these activities or tasks are represented using lightweight, visual notations; namely, activities, parallel splits, parallel merges, and start and end notations. These notations have different roles and represent events/actions.

To create the flow for the process "University Registration," the user starts by adding the start event (left circle) and close event (right circle) and placing them on the canvas. Next, the user adds activity notations in-between the start and close events. These activities, such as "Find University" and "Registration," represent the events that are required to fulfil the University Registration process. He then connects these activities together to specify their order of execution; i.e., control flow. The user can edit activity details, including its name and input and output parameters from the left-hand panel. The user then binds a service operation for the activity to achieve it.



| Name | Value |
|---|---|
| name | Search Univesrity |
| reference | http://www.clothingpro /services /clothingProvider.wsdl |

**Figure 3.** SOA4All user-assisted service composition tool.

The user can define complex logic and flow for the process by adding a conditional statement to signify that under a certain condition the application should follow one flow or the other. For instance, the "Fee Payment" activity has two outgoing flows, one to the "Bank Account" activity and the other to the "Sponsorship" activity. This means to pay university fees, students may pay either using their bank account (default flow) or using a sponsorship letter (if they do not have a bank account). The user can parallelize flows when (s)he wants to specify that two activities may occur at the same time; this is achieved using the parallel split and parallel merge notations (diamond shapes). In Figure 2, "Medical Registration" and "JoinNUS" activities occur after "Accommodation" activity, and represent parallel flows, meaning they occur at the same time after students have found an accommodation. Furthermore, users can define dataflow relationships between activities by associating outputs of activity to inputs of another using a dedicated dataflow editor. The visualization in Figure 2 specifies the control flow of the University Registration process.

The User Assisted Service Composition Tool is a specialized plug-in developed to empower non-programmers to compose services by simply customizing design templates, without the need to model processes manually or write programming code [21]. We refer to this as parametric design using templates or assisted composition. This tool offers a high level of design support to the end-user. The end-user in this case does not specify the logic, control flow, or data flow of the composite service as this is handled by the tool itself using artificial intelligence techniques [21]; (s)he only needs to customize the relevant design template (from the left menu) to create a service composition. Design templates are goal-oriented, pre-defined service compositions containing a list of activities and corresponding services organized in a tabular manner (see Figure 3). Each design template exists to achieve a particular user goal and consists of activities which are essentially actions/steps that contribute to fulfilling that goal. Each of these activities may have a choice of competing services to achieve the activity. For example, the "Student Registration" template enables new students to register for a university course and includes the following activities/steps: "Find University," "Registration," "Fee Payment," "Bank Account," "Deposit Funds," and "Sponsorships." The Fee Payment activity; for instance, can be achieved via either of these services: "PayUniversityFee," "PayPal," or "BACS."

The customization process involves assigning an appropriate service (represented as blue boxes) to each activity (represented as headings of columns), arranging the order of activities, and removing any unwanted activities according to user needs. Further information about each service is available on the right-hand side to inform the selection process of services.

Finally, the user clicks the "execute" button to run and test the composite service "Student Registration." As demonstrated in Figure 3, users are not required to handle any low-level data to achieve a composition. Although simple to use, this tool may be limited by its inability to express complex relationships between activities, such as defining conditional statements and dataflow specifications.

Rationale behind the Design of the Composition Tools

Table 1 highlights the key features of the service composition tools that we used in our comparative study.

**Table 1.** Features of SOA4all service composition tools.

| **Composition Tool and Aim** | **Features** |
|---|---|
| Process Editor:<br>Models service processes using visual notations—manual design of service compositions | (1) Uses graphical notations to visualize and represent services for higher level of abstraction and increased understandability;<br>(2) Relies on graphical manipulation, e.g., drag and drop, to connect notations together for expressing process flow and data flow connections;<br>(3) No coding is required to produce process models;<br>(4) Composition relies on workflow-based notations. |
| User Assisted Composition Tool:<br>Composes services using activity-based templates—parametric design using templates | (1) Focuses on the goal of the composition (i.e., intent of the end-user developer);<br>(2) Relies heavily on selection of services;<br>(3) Services are represented as black boxes, hiding the technical details from the end-user;<br>(4) No visual modelling or coding is required;<br>(5) Composition uses artificial intelligence techniques to assist the user. |

## 3.4. Study Procedure and Tasks' Description

Our participants undertook user testing, which was comprised of the following three main phases:

### 3.4.1. Training Phase

All participants attended a 1-h training session outlining the purpose of the SOA4All studio and detailing the features of its composition tools. Moreover, participants were given a tutorial demonstrating how to develop a composite software service that contacts friends to organize an outing using both composition tools (i.e., manual and assisted composition).

### 3.4.2. Service Composition Phase

First, we conducted a pilot study, including three participants, to ensure that the composition tasks were well described and interpreted in the intended manner. Subsequently, we refined the descriptions of the tasks so that they accurately described what needed to be achieved without detailing the fine-grained steps required to solve the tasks. We also ensured the tasks were sufficiently complex to enable full coverage of the different service composition activities supported by the SOA4All tools.

At the start, the participants were given an evaluation workbook containing the “University Registration” scenario, along with a description of the development tasks and a list describing the available services and their operations that may be used to achieve the scenario. Participants then interacted with the SOA4All tools and performed a range of service composition tasks, as outlined in Table 2.

**Table 2.** Description of service-development tasks.

| Task Type and Relevant SOA4All Tool | Service Composition Task |
|---|---|
| Manual Composition using Process Editor | • Specify the following condition in the process model: “you can pay university fee in one of two ways. If you have a sponsorship letter you submit it to the university, otherwise you pay the fee using a bank account”;<br>• Add an activity “JoinNHS” to the process model. This activity should run parallel to the ”Medical Registration” activity;<br>• Bind the service “RegisterNHS” to the appropriate activity in the process model;<br>• Define the following dataflow connection: for one input “TransferDetails”, map it to the output of a preceding service;<br>• Save the process model you have created. |
| Assisted Composition using User Assisted Composition Tool | • Navigate to and load the “Student Registration” design template;<br>• Remove the activity “Police Registration” from the design template;<br>• For each activity in the template, select the appropriate service according to description in the student registration scenario. |

We selected a scenario that our participants, non-programmers and programmers alike, would find familiar, since it related to the process they went through when applying to study at a particular UK university. The overall description of the test scenario was as follows:

“This scenario describes the registration process that overseas students go through while getting admission into UK universities. Your goal is to complete an overseas student registration process. For this you need to develop a software application which allows you to search for a UK university, register for a course in the university and find an accommodation. There are two ways for paying the university fee, the first way is to open a bank account and get funds transferred into that account. The bank account can be used to make a payment for the university fee. In the second way, you can request a letter from a sponsor and submit that letter to the university. You must choose only one way to pay the university fee. After paying the university fee, you will register with the NHS.”

This scenario helps participants to think about the aim and context of the composition, and the control flow of the composite service to be developed.

The service-development tasks were of two types: (a) manual service composition and (b) parametric design using templates with high degrees of software support (or assisted composition). The SOA4All composition tools offered two varying levels of support when interacting with the users, ranging from very little support during manual service composition to a high level of support during the parametric design with templates, where users only have to select the correct template, and then those services which fit their circumstances and preferences. This is illustrated in Figure 4.

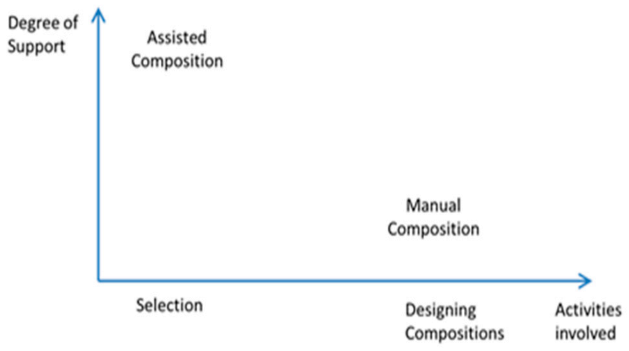


**Figure 4.** Types of service-composition tasks tested.

#### 3.4.3. Subjective Evaluation Phase

On completion of each type of development activity, participants evaluated the respective SOA4All tool, and reflected on their service composition experience by:

1. Completing a post-test questionnaire rating the overall service-development experience, overall usability, and satisfaction towards the relevant SOA4All tool. The questionnaire contained question items about the usability and preferences, such as "ease of use and ease of learning, etc.," to which participants expressed their degree of agreement on a 5-point Likert rating scale, where 1 signified a strong disagreement and 5 signified a strong agreement with the sentence.
2. Completing a de-briefing interview reporting on their impressions about the development experience and SOA4All tools and discussing design challenges and suggestions for future improvements.

### *3.5. Data Collection and Methods of Data Analysis*

Data were collected throughout the study using a concurrent think-aloud protocol and de-brief interviews. The think-aloud protocol [67] encouraged our participants to verbalize their assumptions, thoughts, and problems encountered during the composition process. This protocol, thus, enabled capturing the participants' mental models and reasoning for their service-design strategies. Moreover, self-report data and interaction behaviour were recorded using a screen capturing software, and later transcribed, permitting noteworthy actions to be inserted into a spreadsheet for subsequent analysis.

The quantitative data included objective measures of service-development performance and subjective measures of user satisfaction for each user group. Objective measures were calculated from the interaction videos, and included task completion time (in seconds), completion rate, and correctness of responses. Subjective measures were collected using post-study questionnaires and included perceived usability ratings (on a 5-point Likert scale with 1 = disagree, 3 = neutral, and 5 = agree) of core functionalities of SOA4All tools and development tasks. The questions used for those satisfaction questionnaires are included in Figures 5 and 6.

The qualitative data included each participant's conceptual understanding, development challenges experienced, and feedback about the service-design strategies. Initially, a data analysis scheme was devised whereby data items were divided into three main categories: actions, comments, and problems. Actions refereed to user activities performed to achieve the development tasks; for example navigation, execution, etc. Comments referred to verbal statements and remarks about the development tasks, and included "positive feedback and improvement suggestions." Problems refer to design challenges users encounter during the development tasks and include "conceptual and usability" problems. Conceptual problems signified weaknesses in conceptual understanding of composition concepts and in design strategies to solve composition tasks, whereas usability problems signified user interface related difficulties that hindered user interaction with the tools. The comments and problems were further coded and classified into finer-grained themes using the thematic analysis technique [66]. The thematic analysis technique is a coding methodology whereby patterns and similarities emerge from the data. The resultant themes are presented in tables, in Sections 4 and 5, along with the percentage of occurrence of each theme.

Two researchers coded user comments and problems independently to split the workload and reduce bias. In order to assess rating agreement, both researchers initially coded 10% [68] of the overall sample data achieving 80% inter-coder reliability [69]. Coding disagreements were discussed and reconciled. Inter-coder reliability is an important step in thematic analysis and allows researchers to divide work among coders. Ensuing this, each researcher coded 50% of the remaining qualitative data using the same coding procedure. On completion, the researchers integrated their results by revisiting the emerging themes and resolving any conflicts.

## 4. Results: Manual Service Composition

We varied the manual service composition tasks to include defining a condition, parallel activity, data flow connection, and service binding. Defining a condition consisted of selecting the right condition operator and selecting the right parameters for this operator to express certain actions given a certain condition (i.e., if condition X is true, do action A; otherwise, do action B). Defining a parallel activity consisted of adding a new activity, a parallel split notation, and a parallel merge notation and connecting those to the rest of the process model diagram to achieve parallel execution of two events. The data flow connection task consisted of associating the appropriate outputs of a service to the inputs of another service to specify how data passes from one service to another. Not all participants were able to complete these tasks independently; thus, two levels of assistance were offered by the experimenter, as follows:

- Some help: in this case participants were given hints about, e.g., where some actions can be performed or where some visual notations can be found within the user interface.
- Substantial help: in this case, participants were given explanations about how certain tasks can be solved, e.g., which operator and parameters to choose for the condition, and the exact steps to be undertaken.

### *4.1. Performance during Manual Composition*

A total correctness score was calculated for the design compositions created by the participants to measure their performance (as shown in Table 3). This score added together the individual scores, where each user's solution was graded using the following scheme: a complete solution with no help receives three marks, a solution with some help receives two marks, a solution with substantial help receives one mark, and a wrong solution or no solution receives 0 marks. The manual composition tasks yielded varying performance results between the two user groups, with the non-programmers performing worse than the programmers. All 10 non-programmers who managed to define the conditional branch, required substantial help from the experimenter. In contrast, only two out of nine experts required substantial help to complete the same task. Indeed, these low completion rates and scores support hypothesis H1b. All non-programmers successfully bound an activity with the correct service, with only three of them requiring some level of assistance. The programmers exhibited similar behaviour.

Specifying data flow connections between two services triggered the highest number of failures in both groups, with five non-programmers unable to complete the task. Five of the seven non-programmers who completed the data flow task required some assistance, as opposed to only three out of seven experts. The non-programmers saved their compositions without any help.

Moreover, the non-programmers spent significantly more time adding the parallel branch to the composition, ($t$-test (16) = 2.94, $p$ = 0.009), and specifying the data flow connections between services, ($t$-test (12) = 3.37, $p$ = 0.006), than the programmers. These figures confirm our hypothesis H1c. Both user groups spent the longest time creating the conditional branching, parallel branching, and specifying dataflow, (F (4, 84) = 22.74, $p < 0.001$). These tasks encompass more steps and require strong analytical skills; thus, ensuring longer completion time.

**Table 3.** Task completion time and number of participants who completed manual service composition; statistical differences are marked by **.

| Manual Composition Task | Non-Programmers | | | Programmers | | |
|---|---|---|---|---|---|---|
| | Average Task Completion Time (seconds) | # Users Who Completed the Tasks | Total Correctness Score | Average Task Completion Time (Seconds) | # Users Who Completed the Tasks | Total Correctness Score |
| 1. Conditional branching, operator and parameters selection ** | 300 (SD = 114.11) | 10 (10 received substantial help) | 10 | 268 (SD = 104.79) | 9 (4 received some help, 2 received substantial help) | 19 |
| 2. Binding activity with an appropriate service | 88 (SD = 38.06) | 12 (2 received some help, 1 received substantial help) | 32 | 93 (SD = 69.33) | 12 (1 received some help) | 35 |
| 3. Adding a parallel branch ** | 288 (SD = 113.26) | 10 (6 received some help, 4 received substantial help) | 16 | 164 (SD = 57.11) | 9 (6 received some help) | 21 |
| 4. Specifying data flow connections ** | 186 (SD = 57.35) | 7 (5 received some help) | 16 | 101 (SD = 34.10) | 7 (2 received some help, 1 received substantial help) | 17 |
| 5. Saving the composition process | 22 (SD = 11.90) | 12 | 36 | 31 (SD = 13.31) | 12 | 36 |

## 4.2. Manual Composition Problems and Overall Perception

A user's technical profile had no significant effect on the number of problems, positive statements, and suggestions reported during manual service composition (*t*-test not significant). However, the number of conceptual problems encountered was significantly higher than that of the usability problems, (F (1, 46) = 20.24, $p < 0.001$), which reveals the complexity and challenging nature of manual composition (Table 4).

**Table 4.** Average number of problems, positive comments, and suggestions during manual service composition (per user; cells in grey color are significantly different).

| Category | Non-Programmers | | Programmers | |
|---|---|---|---|---|
| | M | SD | M | SD |
| Overall problems | 5.99 | 2.74 | 6.33 | 3.07 |
| -Conceptual problems | 4.91 | 2.29 | 4.00 | 1.85 |
| -Usability problems | 1.08 | 1.26 | 2.33 | 1.83 |
| Positive statements | 3.00 | 2.67 | 3.33 | 2.62 |
| Suggestions | 1.58 | 1.31 | 1.33 | 1.08 |

Table 5 summarizes the main conceptual problems that non-programmers encountered during the manual service composition. In line with the performance results, the non-programmers reported that defining the conditional statement was the main source of difficulty, which supports our hypothesis H1b. This problem was also reported by the expert sample. Non-programmers were then hindered by complexity of workflow-based composition approach and their weak understanding of low-level terminology, but this was not a concern for the expert sample, arguably owing to their technical background. Instead, the expert sample was more critical about specifying data passing between services than the non-programmers, which is an interesting and unexpected finding. So, what makes these manual service composition tasks challenging to complete by non-programmers?

**Table 5.** Conceptual issues (and percentages of occurrence) during manual service composition.

| Conceptual Problem | Non-Programmers | Programmers |
|---|---|---|
| Definition of the conditional branches | 34% | 35% |
| Complexity of manual service composition and low-level terminology | 32% | 6% |
| Creation of parallel branches | 15% | 6% |
| Specification and understanding of data flow | 10% | 22% |
| Other disperse issues | 9% | 31% |

Next, we delved into the factors that led to the emergence of the problems mentioned above. The most recurring problem highlighted by our participants revolved around expressing choice between two services using a conditional branch. A conditional branch is equivalent to creating an "if (condition); . . . else . . . ;" statement using a traditional programming language. For non-programmers, the issue may be attributed to their weak understanding of the meanings of operators and parameters, and poor selection of the appropriate operator and parameters of the condition. This in turn, may be justified by their unfamiliarity with the syntax and the technical language used. For programmers, condition definitions were challenging, owing to their unfamiliarity with the syntax (e.g., how many parameters are needed) for defining a condition in a workflow-based composition, and to the inability to see the immediate effects of conditional statements once defined.

**Conceptual problem one:** Definition of a conditional branch.
E.g., "Parameter is very confusing; I don't understand it and it is too technical; it is beyond me," P11, non-programmer.
"I don't know which operator to choose because I don't understand the language," P1, non-programmer.
"Setting a condition is like programming; it is too technical and I am not happy about that," P6, non-programmer.
"I wasn't sure how many parameters I had to select or where it should be placed," P11, programmer.
"Do I need to specify the condition for other route?" P1, programmer.
"I do not see any changes to the main model once I have set the condition," P12, programmer.

Approximately 32% of non-programmers' comments complained about the programming nature of manual service composition, and about the syntactic language used to refer to service-composition concepts. We believe that low level language aggravates the perceived complexity of manual service composition and self-efficacy of non-programmers. Expectedly, however, this was not reported as a problem by the programmers, who were acquainted with such vocabulary.

**Conceptual problem two:** Technical nature of service composition and terminology.
E.g., "These words like binding sound too technical . . . The vocabulary needs to be changed a bit; sometimes it is too technical," P3, non-programmer.
"All these things on the left hand side seem very complicated and technical; I don't understand what they mean and it is intimidating," P11, non-programmer.
"It was a little technical and I would never be able to figure it out without training or assistance," P12, non-programmer.
"This is like programming and it is too technical," P7, non-programmer.

Some non-programmers struggled to create parallel branches to express the simultaneous execution of a new activity/event within the service composition. When adding the parallel branch, participants did not know how to use the relevant (i.e., split and merge) notation and where to place it within the composition process model, probably due to unfamiliarity with this type of task (i.e., parallel events).

**Conceptual problem three:** Definition of a parallel branch,
E.g., "I am trying to find where to create the parallel step," P3, non-programmer.
"So how do I define parallel processes and splits?" P6, programmer.

In respect to dataflow connections, fewer complaints were received from non-programmers than from programmers. The non-programmers failed to understand the meaning of inputs and outputs of services, whilst the programmers failed to match the outputs of one service to the inputs of the next service. This is an unexpected and intriguing effect for programmers, highlighting the mismatch between their mental model of service composition, likely based on control flow as dominant programming paradigm, and the dataflow specification in our tool.

**Conceptual problem four:** Specification and understanding of data flow connections.
E.g., "I do not understand the difference between these input types," P5, non-programmer.
" . . . am not 100% sure which output to select," P11, programmer.
"The only misleading part is that you need to specify the output of the previous service and I did not know that," P7, programmer.

The remaining conceptual problems were dispersed and included weak understanding of service binding (i.e., attaching a service to an activity notation), not knowing the meaning of some workflow-based notation, and confusion throughout the compositional process. Notably, the programmers doubted that manual service composition was achievable by non-programmers; e.g., "This is supposed to be for non-programmers; I think it will be difficult for them," P1, programmer; "The notation is not easy to understand for non-technical users," P6, programmer.

With respect to usability, the non-programmers were more judgmental about the inability to directly manipulate visual notation and perform certain actions, such as removing unwanted services (54% of themes), than the programmers (36% of themes). Direct manipulation is an important feature to non-programmers since they use it heavily to program the behaviour of composite services. They were more concerned about the immediacy and complexity of the feedback which was beyond their technical knowledge (23% of themes) than the programmers (11% of themes).

Two interesting and unexpected results emerged in relation to manual service composition by non-programmers. Firstly, despite the difficulties experienced, some non-programmers claimed that manual service composition is easy to perform (Table 6). Secondly, several non-programmers praised manual service composition for providing more details and offering more freedom to develop powerful personalized applications; e.g., "This way provides much more detail and gives you freedom," said P6, non-programmer. "It is good to have this application, as I can make powerful and complex tools with it," P8, non-programmer. He highlighted the fact that the service composition can be manipulated and personalized according to personal needs; e.g., "The diagram is very good, as you can add many things and make it specific," P8, non-programmer. The ability to see what is happening using the process editor was appreciated even by the non-programmers; e.g., "I like this one as I can see what is going on," P9, non-programmer.

**Table 6.** Positive features of manual service composition.

| Positive Aspect | Non-Programmers | Programmers |
|---|---|---|
| Ease of use of manual service composition | 20% | 23% |
| Control and freedom of manual service composition | 20% | 15% |
| Use of notations to represent activities and services | 3% | 15% |
| Other disperse features | 57% | 47% |

As expected, programmers valued the control offered by manual service composition and the use of visual notation to represent services and activities (30% of the total themes).

The non-programmers emphasized the need for guidance and instructions during the process of composing software services, especially data flow specification which demonstrates the social learning aspect of users; e.g., "It would be good to have some sort of help guide during the data flow process," P5, non-programmer (Table 7). However, this was less of a concern for the expert sample, who focused more on user interface improvements, such as efficient navigation and the provision of immediate results (runtime effects). One non-programmer requested to simplify the terminology and stay away from technical jargon: "You should use more general words; e.g., 'function' instead of 'operator,' especially that this is for lay people," P1, non-programmer. Two non-programmers suggested a basic and advanced view for different types of users; e.g., "Perhaps it should have basic and advanced settings for different types of users and keep certain things hidden, as at the moment it puts me off," P9, non-programmer. This recommendation supports the notion of progressive learning.

**Table 7.** Strategies for improving manual service composition.

| Strategy | Non-Programmers | Programmers |
|---|---|---|
| Guide users through service composition using, e.g., wizard, tutorials | 26% | 12% |
| Include more help and instructions | 26% | 6% |
| Define complex terms | 6% | 6% |
| Provide immediate feedback | 5% | 18% |
| Improve visual appearance of the tool | 5% | 25% |
| Other disperse suggestions | 28% | 33% |

## *4.3. Perceived Satisfaction Towards Manual Service Composition*

Scales showed an excellent reliability (Cronbach alpha >0.90). Non-programmers' perceived complexity of manual service composition activities and satisfaction with the process editor differed from the programmers' in all dimensions: $t$ (22) = −3.04, $p$ = 0.006, as depicted in Figure 5. On average, the non-programmers (m = 2.42, SD = 1.16) found it less easy to define the condition than the programmers (m = 3.50, SD = 0.90). Moreover, they felt less confident (m = 3.00, SD = 1.34) using the process editor than the programmers (4.08, SD = 0.79), which may be due to their low self-efficacy. The non-programmers did not find it less difficult (m = 2.67, SD = 0.88) to understand the service composition notation than the programmers (m = 4.33, SD = 0.88). Overall, the non-programmers (m = 3.42, SD = 0.90) were less satisfied with the process editor than the programmers (m = 4.08, SD = 0.66). These satisfaction scores closely mirror the performance results and qualitative feedback, confirming hypothesis H1e.

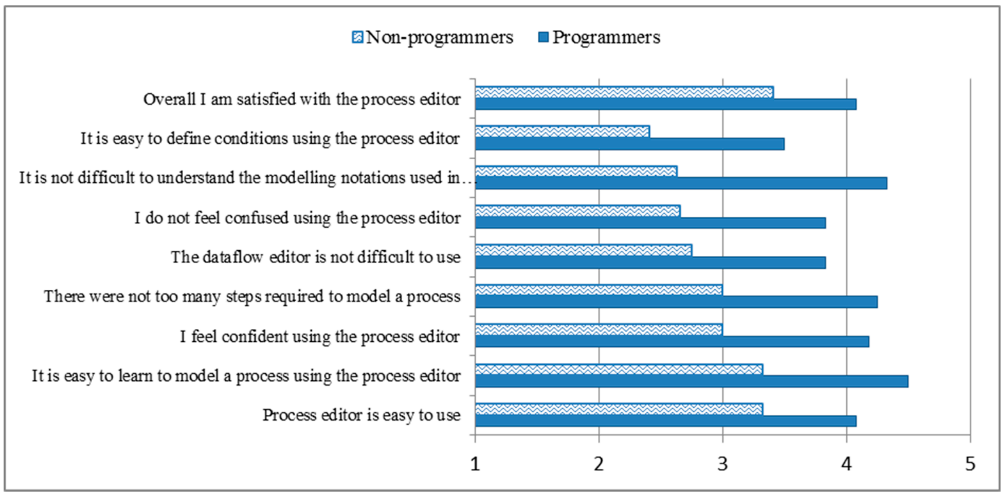


**Figure 5.** Average usability rating of manual service composition by non-programmers and programmers; all dimensions differed statistically.

## 4.4. Summary

The manual service composition hypotheses posited at the beginning of this article were reviewed in light of the evidence; see Table 8. Hypotheses H1b → H1e were fully supported, while H1a was partially supported. Thematic analysis results mirrored and affirmed the quantitative results and showed that the non-programmers' struggle to define conditions was caused by their weak conceptual understanding of the service elements, such as parameters and operators, that constitute a conditional statement, and their inability to differentiate between multiple operators and match the correct operator to the intended action. This was aggravated by the technical language used to express these composition concepts. The expert sample, however, was mainly hindered by their unfamiliarity with the syntax of the Process Editor tool to formulate conditions and dataflow connections.

**Table 8.** Review of the manual service composition hypotheses (H1a → H1e).

| Manual Composition Hypothesis | Differences between the Groups | Verdict |
|---|---|---|
| H1a Non-programmers will find it more difficult to understand service terminologies compared to programmers | -Statistical differences in satisfaction scores of composition notations (Independent samples *t*-tests)—Figure 5<br>-Percentage of conceptual problems discussing the complexity of terminologies and notations (thematic analysis)—Table 5 | Partially supported |
| H1b Non-programmers will find it more difficult to define conditional flows (IF ... ELSE) between services compared to programmers | -Statistical differences in performance results, i.e., correctness of (if ... else) specifications, task completion rates (*t*-tests)—Table 3<br>-Statistical differences in satisfaction scores of condition definition (*t*-tests)—Figure 5 | Supported |

**Table 8.** *Cont.*

| Manual Composition Hypothesis | Differences between the Groups | Verdict |
|---|---|---|
| H1c Non-programmers will find it more difficult to create dataflow connections between services compared to programmers | -Statistical differences in completion times of data binding between services (*t*-tests) —Table 3<br>-Statistical differences in satisfaction scores with respect to data connections (*t*-tests)—Figure 5 | Supported |
| H1d Non-programmers will face more challenges with manual service composition compared to programmers | -Statistical differences in respect to the manual service composition tasks (*t*-tests)—Table 3<br>-Statistical differences in respect to the number of conceptual problems (*t*-tests)—Table 4 | Supported |
| H1e Non-programmers will hold a more negative perception about manual composition compared to programmers | -Statistical differences in satisfaction scores with respect to the manual composition process (*t*-tests)—Figure 5 | Supported |

In addition to the hypotheses listed above, various interesting findings emerged with respect to manual service composition. Non-programmers were found to spend more time and needed more substantial guidance to define the parallel processes compared to programmers (*t*-tests). This was also evidenced by the percentage of conceptual problems emerging in the themes.

## 5. Results: Assisted Service Composition

This part tested non-programmers' reactions and perceived satisfaction with assisted service composition, an AI-based composition approach, where the system hides technical details (e.g., service specifics) and takes control of various design decisions on behalf of the end-user, such as conditional operators, control flow, and dataflow connections. Hence, the complexity of service composition is reduced but at the expense of flexibility and expressiveness.

### *5.1. Performance during Assisted Composition*

Generally, both user groups showed similar task completion rates across the three tasks and spent the same time to complete task one and two (see Table 9). Nonetheless, user background had a strong effect on completion time of the service selection task (task 3)—$t$-test (22) = −2.58, $p < 0.05$—with the non-programmers' spending significantly less time to select services (m = 192.75, SD = 88.29) than programmers' (m = 297.67, SD = 109.51). This may be attributed to their less careful investigation of the services available for each activity.

**Table 9.** Task completion time and number of participants who completed assisted-composition tasks. Statistical significance is marked by **.

| Task | Non-Programmers | | Programmers | |
|---|---|---|---|---|
| | Average Task Completion Time (Seconds) | # Users Who Completed the Tasks | Average Task Completion Time (Seconds) | # Users Who Completed the Tasks |
| 1. Finding and navigating to the right service design template "Student Registration" | 16.17 (SD = 8.44) | 12 | 16.00 (SD = 9.87) | 12 |

**Table 9.** *Cont.*

| Task | Non-Programmers | | Programmers | |
|---|---|---|---|---|
| | Average Task Completion Time (Seconds) | # Users Who Completed the Tasks | Average Task Completion Time (Seconds) | # Users Who Completed the Tasks |
| 2. Removing “Police Registration” activity from design template | 47.33 (SD = 39.29) | 11 | 56.08 (SD = 58.13) | 11 |
| 3. Selecting appropriate services for activities ** | 192.75 (SD = 88.29) | 12 | 297.67 (SD = 109.51) | 11 |

## 5.2. *Assisted-Composition Problems and Overall Perception*

Notably, the non-programmers reported less usability issues (*t*-test (22) = −2.20, *p* = 0.038) and were more positive about the assisted composition than the programmers, with the statistical difference approaching significance (*t*-test (22) = 1.96, *p* = 0.063) supporting hypothesis H2a, as shown in Table 10. This contrasts the results of manual composition (Section 4), were more conceptual issues were prevalent. These results support hypothesis H2b, which stipulates a more positive view by non-programmers toward assisted composition than programmers. So, which factors induced these opposing perceptions by the two user groups?

**Table 10.** Average number of problems, positive comments, and suggestions during the assisted service composition; Cells in grey signify statistical differences.

| Category | Non-Programmers | | Programmers | |
|---|---|---|---|---|
| | M | SD | M | SD |
| Overall problems | 1.92 | 1.78 | 3.17 | 2.12 |
| -Conceptual problems | 1.08 | 1.16 | 0.92 | 1.24 |
| -Usability problems | 1.00 | 1.20 | 2.25 | 1.54 |
| Positive comments | 3.33 | 2.30 | 1.67 | 1.82 |
| Suggestions | 1.25 | 1.86 | 1.58 | 1.67 |

To answer this question, we studied the types of conceptual weaknesses and strengths of assisted composition. Table 11 stipulates that the basic capabilities of assisted-composition approach and inability to express complex application logic and behaviour instigated the negative perception, especially amongst the expert sample, who typically wanted more control to create complex application logic and behaviour. Technically-oriented terminology and the absence of runtime effects were deterrents from using assisted composition by the non-programmers. A common issue to both user groups was the inability to differentiate between the available services under each activity of the design template.

A few non-programmers highlighted that the composition process is quite simple and there is no opportunity to see why certain actions are happening, nor the possibility to program complex behaviours. For instance, users are unable to express conditional statements (e.g., “Or”) using the tool. The feedback from non-programmers 1 and 5 below suggest that not all non-programmers are happy with hiding the logic of the composition process. This is an interesting and not-entirely expected result, since the expectation is that simplicity would guarantee a favourable perception by non-programmers. This problem was of paramount importance to the programmers, constituting 64% of their comments, because assisted composition constrains them from creating complex application logic and sophisticated application behaviour (supporting Hypothesis H2a). Therefore, whilst abstracting composition approaches from technical representations may reduce cognitive complexity and learning costs for the lay people, it is not always a favourable option for end-users, including some non-programmers.

**Table 11.** Conceptual problems during assisted composition.

| Conceptual Problem | Non-Programmers | Programmers |
|---|---|---|
| Basic capabilities of composition process and implicit application logic | 38% | 64% |
| Terminology and technical jargon | 23% | 9% |
| Differentiation between services | 15% | 18% |
| Absence of runtime effects | 15% | 0% |
| Other disperse problems | 9% | 9% |

A plausible explanation might be the easy nature of the assisted-composition approach and the level of assistance provided to users by the tool, without considering the limitations and lack of freedom imposed on the expert sample by the assisted-composition approach.

**Conceptual problem one:** Basic capabilities of the assisted-composition design approach.
E.g., "I do not find the selection process intuitive; it should be a flow of processes, perhaps page by page," P5, non-programmer.
"A little difficult as some services were compatible and some not without explanation. Would like reasoning behind decisions," P1, non-programmer.
"I would prefer to see and input logic and see how things work rather than click and drag. The process is too simple and there is no opportunity to see why things are happening," P11, programmer.

The non-programmers were intimidated by the terminology and language used within the tool, as they were deemed technical and complex. However, this did not emerge as an issue for the expert sample.

**Conceptual problem two:** Terminology and technical jargon.
E.g., "The language is too complicated; for example, press execute to deploy composition," P11, non-programmer.
"How to use guide is a little technical and complicated," P6, non-programmer.

Both groups of users were undecided about differences between the candidate services under each activity of the design template, making the selection of services a challenging task. The tool did not provide any systematic criteria to demonstrate the quality of services.

**Conceptual problem three:** Differentiation between the services.
E.g., "I do not understand the difference between services. In terms of service provider it is a little unclear," P5, non-programmer.
"Again, I have two services. I don't know which one to go for," P8, programmer.
"What is this is BACS service, and what is the difference between these services," P4, programmer.

Finally, the non-programmers complained about the inability to view the outcome of their composition efforts and test the composite application. This observation is aligned with the findings of Namoun et al. [11], where end-users emphasized the need to view runtime results as the development process unfolds. Non-programmers P1 and P9 suggested that availability of runtime effects aids understanding and allows users to inspect the behaviour of the application and debug any unapparent problems.

**Conceptual problem four:** Absence of runtime effects.
E.g., "I cannot see the results. If this was real, I would just go online and find out and use it," P1, non-programmer.
"It was easy, but I wish I could see the end result, so I could understand what I have done," P9, non-programmer.

In respect to usability problems, non-programmers had difficulty manipulating the "Police Registration" activity, as it could only be removed using a context menu, which was not obvious (23% of themes). They were also unsure as to why some services were greyed out upon selecting a specific service (17% of themes). However, this confusion diminished as soon as the experimenter explained that the greying-out feature highlights incompatibility of services. Other comments focused on general issues with the user interface.

Despite the above challenges, the non-programmers praised the tool as being easy to follow; e.g., "It is easy to follow the logical steps," P4, non-programmer. They also said it was easy to operate more than the expert sample; e.g., "The process is straightforward," P7, non-programmer, confirming hypothesis H2a. Moreover, they appreciated the clickable nature of services (i.e., direct manipulation) and the ability to remove activities (Table 12).

**Table 12.** Frequency of positive aspects of assisted-composition.

| Positive Aspect | Non-Programmers | Programmers |
|---|---|---|
| Ease of use and intuitiveness of composition process | 28% | 15% |
| Direct manipulation of services and activities | 22% | 25% |
| Use of colours to specify compatibility of services | 15% | 5% |
| Other disperse positive features | 35% | 55% |

The most remarkable strategies suggested by the non-programmers emphasized the need for a wizard guide and tool tips to guide novice users through the selection process of services and activities, and to supply services with further details (e.g., service properties and provider information) to empower them to differentiate between the quality of services during service selection (Table 13). Interestingly, two programmers demanded the inclusion of flow and process diagrams to show details and flow connections between services, confirming the first conceptual problem; e.g., "Something like a flow diagram to show details of services and links between them might be useful," P2, programmer.

**Table 13.** Strategies for improving assisted-composition.

| Strategy | Non-Programmers | Programmers |
|---|---|---|
| Include a wizard, tool tips, and help sections | 33% | 20% |
| Add more details and information about services and activities | 27% | 20% |
| Include options for quickly removing activities | 13% | 15% |
| Other disperse suggestions | 27% | 45% |

## *5.3. Perceived Satisfaction Towards Assisted Composition*

Satisfaction scores of the assisted-composition approach by both user groups were similar (*t*-test not significant), mirroring the performance results (as in Figure 6). The non-programmers rated the assisted-composition tool as easy to use (m = 4.08, SD = 0.77) and easy to learn (m = 4.26, SD = 0.81). They also agreed that the tool is not difficult to navigate (m = 3.83, SD = 1.09), expressed willingness to use the tool in the future (m = 3.83, SD = 1.00), and were generally satisfied with the tool (m = 3.79, SD = 0.72), as shown in Figure 6. The summary ratings do not show significant differences between programmers and non-programmers; thus, failing to support H2a in contrary to the indicators of conceptual issues above. This is an interesting result, which demonstrates the utility of the assisted composition for both groups, despite programmers finding it a bit restrictive and not fully transparent.

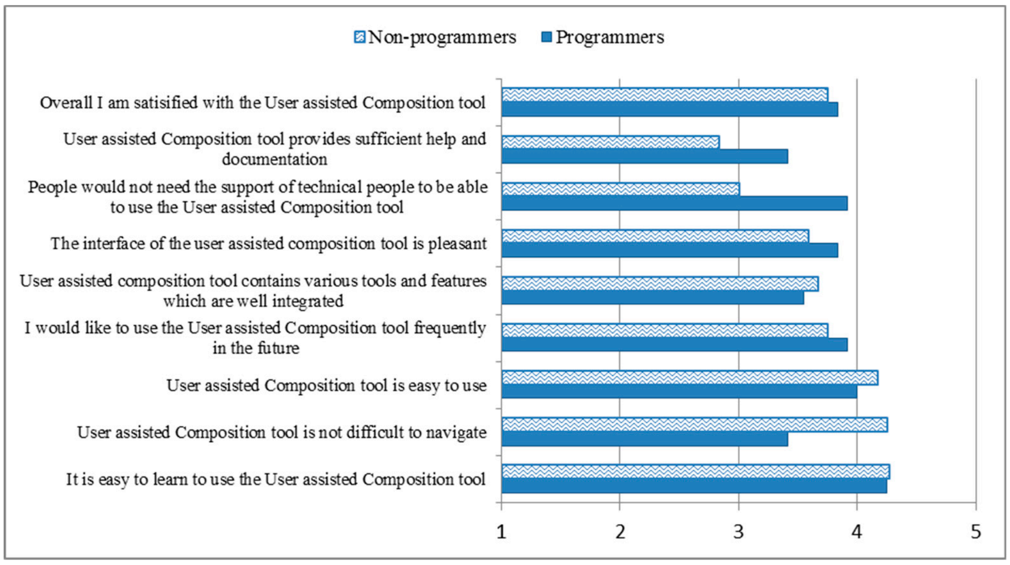


**Figure 6.** Average usability rating of assisted service composition by non-programmers and programmers; no statistical differences were detected.

## *5.4. Summary*

The assisted service composition hypotheses posited at the beginning of this article were reviewed in light of the evidences; see Table 14. Statistical analysis showed that the non-programmers prefer assisted composition over manual composition, which is an expected result (i.e., supporting H2b). Moreover, thematic analysis of user comments showed a more positive perception of assisted composition by the non-programmers than the programmers. However, the other results (i.e., satisfaction scores) were generally similar across the two groups; thus, partially supporting H2a.

**Table 14.** Review of the assisted-composition hypotheses (H2a and H2b).

| Assisted Composition Hypothesis | Differences between the Groups | Verdict |
|---|---|---|
| H2a Non-programmers will hold a more positive perception about assisted composition compared to programmers. | -Statistical differences in number of positive feedback of the assisted-composition approach (*t*-tests)—Table 10<br>-Statistical differences in number of perceived usability problems of the assisted-composition approach (*t*-tests)—Table 10<br>-Percentage of conceptual problems (thematic analysis)—Table 11 | Partially supported |
| H2b Non-programmers will favour assisted composition than manual composition compared to programmers. | -Statistical differences in satisfaction scores between assisted composition and manual composition (*t*-tests) | Supported |

In addition to the hypotheses listed above, various interesting findings emerged with respect to assisted service composition. Non-programmers were found to spend less time selecting the services of the design template compared to programmers (*t*-tests), which signifies a high reliance and trust in the recommendations of the tool by the non-programmers. Moreover, programmers complained about the simplicity and inability to express complex logic by the assisted composition, as shown by the high percentage of conceptual problems concerning this issue.

## 6. Discussion and Practical Implications for Service Composition

This research has investigated the service composition issues and mental models of non-programmers and compared them to those of an expert sample of programmers, with the aim of identifying gaps and flaws in their decision-making strategies because of their lack of technical knowledge and programming experience. This approach has been adopted for decades and has been shown to assist in creating design support tools [70]. Our study examined two service composition approaches; namely, manual service composition and assisted composition (driven by artificial intelligence techniques). These tasks constitute the major activities of service-development lifecycle and enjoy variable levels of support from our tools. Manual service composition is a chart flow-like composition approach and allows end-users to build service compositions by wiring diverse visual notations, which represent services, and specifying control flow and data flow between the services. This approach enjoys very little support from the process editor, as the end-user is responsible for defining the compositional logic and execution behaviour. Assisted composition, however, is a design template-based composition approach and allows end-users to build service composition by direct manipulation (e.g., point and click) of services for each activity within the template. This activity enjoys a high level of support from the tool where service details are abstracted and artificial intelligence techniques are used to make complex design decisions (e.g., dataflow) on behalf of the user [21].

Overall, the results of our study provide substantial evidence of the differences between non-programmers' and programmers' composition performance, which is in stark contrast to the claims of recent studies [18,23]. Our study advises that non-programmers prefer the automated support offered by the assisted-composition approach (i.e., H2b supported), while programmers prefer the control and expressive power offered by the manual composition approach. We now revisit our research questions and recommend relevant guidelines to support non-programmers' mental models and decision-making strategies in respect to service composition.

**RQ1:** *What type of design challenges are faced by non-programmers when composing services manually using a workflow-based composition approach that offers minimal tool support?*

Manual service composition revealed the most differences between the mental models and decision-making strategies of non-programmers and programmers when they were tasked with implementing a student registration software service. The activity of "setting a condition" was the most challenging task non-programmers encountered owing to their inability to understand the meaning and functionality of conditional operators (i.e., H1b supported), in addition to the technical language in which parameters were written. This was evident from the results, as all non-programmers who completed this task required substantial assistance from the experimenter. The qualitative results also confirmed their struggle with the technical specifics and terminologies (i.e., H1a partially-supported) involved in selecting and populating conditional operators.

The task of 'creating dataflow connections' triggered the highest number of mistakes by both user groups; however, the non-programmers took longer to complete it, as it involved understanding of both service outputs and service inputs and making the correct associations between them (i.e. H1c supported). This result agrees with various findings, such as [11,71]. Such understanding and associations are beyond ordinary people. However, recent research showed that introducing programming constructs, such as data driven loops and single assignments, had a positive influence on the programmatic understandability and learning of end-users [72].

A new observation in our study showed that the task of 'adding a parallel activity' proved to be less problematic for the non-programmers than the condition definition and data flow connection tasks. However, the dilemma for the non-programmers revolved around understanding and using the appropriate notations, e.g., split notation and merge, to specify parallel execution of events (i.e., H1a partially-supported). Indeed, they took longer to complete this task than the programmers, which may be attributed to their unfamiliarity with programming languages and visual notations.

Interestingly, manual service composition took longer and resulted in more conceptual problems than usability problems when compared to assisted composition (i.e., H1d supported), thus leading to a negative view towards manual service composition by non-programmers. This demonstrates the complex nature of manual service composition and the need for strong development and analytical skills to undertake these tasks. Despite its perceived complexity, a few non-programmers liked the control and freedom offered by manual service composition.

The low performance scores, coupled with user feedback, demonstrate that non-programmers are unable to comprehend the concepts of condition definition and data flow connection, primarily due to their unfamiliarity with such tasks that demand vast technical knowledge and due to the weak support provided by the tool. Satisfaction scores show that non-programmers perceive manual service composition as less easy to learn and use, the visual notation as less easy to understand, and feel confused during the composition process (i.e., H1e supported). This may be attributed to their unfamiliarity with programming languages and composition notations. These observations suggest a strong link between a user's self-efficacy, technical background and skills, and ability to learn and perform development tasks.

Our design recommendations to lower the barriers to manual service composition by non-programmers are as below.

- **DR1:** "*Employ user language and familiar terminology and avoid technical and low-level language*." Non-programmers have a deficient syntactic knowledge, and using specialized terms aggravates the perceived complexity of service composition and lowers self-esteem. This guideline agrees with the previous recommendations [11,73].
- **DR2:** "*Implement a robust and rich manipulation mechanism by which non-programmers can perform differing manual service composition tasks and operations directly on services*." For non-programmers who are unwilling to learn computational concepts and languages, using approaches such as "user interface service composition" [41,42], the "what you see is what you get" approach [5], or "composition at the presentation layer" [74] might be viable solutions to reducing the cognitive challenges and effort. Demonstrating composition approaches through a dedicated software tutor has also been shown to be an effective method [75].
- **DR3:** "*Abstract composition activities that require analytical skills from technical details and employ more natural metaphors to achieve complex operations, such as conditional statements, dataflow, and parallel events*." Manual service composition tasks that require a strong understanding of service composition concepts and programming constructs (e.g., if-else statements and loops) are beyond the capabilities of non-programmers and impose high cognitive loads. An alternative strategy may be to use natural language [76,77] and programming by example [75] to express complex concepts. Trigger-action programming (e.g., IFTTT) is another approach that has demonstrated promising results, although further research is required [65,78,79]. Recently, the 5W model showed promising results for the expression of causes and effects (i.e., conditions) [74]. Programming constructs may also be visualized, and this has been showed to enhance a learner's understandability [72].
- **DR4:** "*Enable non-programmers to observe and inspect the behaviour of their compositions on the fly and introduce fault/bug localization features*." It is nearly impossible to create perfect compositions on the first attempt, as evident in the condition definition and dataflow specification tasks; therefore, service-development environments should host features to locate and fix bugs within the composition models. This guideline agrees with previous research [80]. This way, users can check the correctness of their requirements and composite service [81].

- **DR5:** "*Provide composition walkthroughs and examples of varying complexity to raise self-efficacy and confidence of non-programmers progressively*." This design guideline enables a "gentle slope" of cognitive challenge and tailorability.

**RQ2:** *What are the attitudes of non-programmers when a software tool is "taking over" their design by abstracting technical details and advising them about consequences of their choices?*

The qualitative results showed two opposing views towards the assisted service composition. The first view, by non-programmers, appreciates the ease with which services can be composed through simple clicks on activities and their services (i.e., H2a partially-supported). Assisted composition simplifies service composition through activity-based templates, offering direct manipulation of services without the need to write programming code. These features support the limited syntactic and semantic knowledge of non-programmers, as they can work directly with the services that interest them without dwelling in computational abstracts. Interestingly, the help section of the tool was described as technically-oriented and difficult to understand, and non-programmers expressed doubts over their capability to perform composition without assistance. This can be attributed to their low self-efficacy and confidence [5,58]. The second view, by programmers, criticizes the assisted composition for its inflexibility and restrictiveness, and questions whether it is sufficient to express complex scenarios. For instance, this approach to composition did not allow the formation of conditions, limiting the richness of the composite services produced. Programmers were more inquisitive than non-programmers' as they requested to learn the mechanics behind assisted composition and why certain things happen. This was evidenced in the time they spent arranging activities and assigning services to template activities, which was statistically longer than that of non-programmers'. Justifiably, programmers were negative towards the assisted-composition approach due to its low level of expressiveness and power, which did not allow them to achieve advanced application behaviour (i.e., H2a partially-supported). Service composition approaches that rely solely on direct manipulation often lose their power of expressiveness and flexibility. In conclusion, such kinds of approaches may be more suitable for users whose primary function is not programming.

The design recommendations to lower the barriers to assisted service composition by non-programmers are as below.

- **DR1:** "*Use activity or goal-oriented templates to facilitate and support service composition tasks*." Non-programmers are capable of undertaking development tasks that make use of natural languages [73]. Activity-based templates identify the goal of composition and break this down into simple actions that achieve that goal (e.g. task analysis). This process supports the strategic knowledge of non-programmers who are naturally aware of domain concepts but not acquainted with the underlying programming and service composition concepts.
- **DR2:** "*Use direct manipulation to enable non-programmers to directly use services and their constituents (e.g., input and outputs)*." Direct manipulation or direct selection of services in our context offers cognitive benefits to non-programmers, for it lowers the need to learn abstract programming constructs and eliminates the need to write programming languages.
- **DR3:** "*Provide two design views: a simple view, hiding complexities, and an advanced view for those willing to delve into and tweak the tool's decision-making*." Some non-programmers were dissatisfied with the oversimplification and hiding of all aspects of composition. This guideline ensures flexibility and support for a wide range of users.

In summary, during manual service composition, the non-programmers exhibited a weak understanding of service parameters and conditional operators, and demonstrated a weak ability to map service outputs to inputs. In contrast, the programmers were able to complete the design solutions with less assistance, leading to a more satisfying development experience. This is not to imply that non-programmers' mental models need to be fixed but rather accommodated. However, during assisted service composition, the non-programmers enjoyed the ease by which they were able to assemble the services, contrary to the programmers, who were dissatisfied with the lack of freedom, expression, and limitations imposed by the approach.

## 7. Limitations of the Study

Admittedly, the study herein has various limitations worth qualifying; firstly, it included a small sample of students purposefully-selected to fit the goals of the research, which may not be representative of the population; thus, restricting the generality of the findings. Therefore, future studies need to increase the sample size and include end-users who are not students but rather practitioners and domain experts. Secondly, the training undertaken may have not been sufficient to familiarize the non-programmers with service-composition activities and the underlying concepts. Thirdly, the service composition tools used did not exhibit runtime aspects which may have influenced user perception and judgment of composition results and experience; indeed, such negative influence has been reported in other works [11]. Thirdly, our user study explored the differences between the non-programmers and programmers using only one composition scenario; thus, further scenarios are needed assert the purported findings. Finally, our testing was limited to modelling processes using work-flow based and AI-based service composition approaches. Other service-development activities, such as service annotation and monitoring were not investigated in this study.

It is part of our research agenda to conduct a set of controlled-experiments focusing on the link between composition style, such as a front-end approach and workflow-based approach, and service-development performance. In these controlled-experiments, we will utilize fully-functional development tools, and increase the sample size by involving more participants from different backgrounds, considering different sub-groups that form technical users to allow for a more comprehensive training to take place, in order to draw solid, generalizable conclusions from the research findings.

## 8. Conclusions

This article presents the results of a confirmatory user study, which investigated non-programmers' design challenges when developing service compositions, conceptual difficulties experienced, and overall satisfaction with service composition, and compared those to a baseline sample of programmers. A total of 24 participants, including 12 non-programmers and 12 programmers, performed two types of service development activities (i.e., manual service composition and assisted service composition) to achieve a composite service using the SOA4All studio. Manual service composition revealed the greatest differences between the two user groups, where the non-programmers struggled to express conditional statements, add parallel events to the composition, and specify data-flow associations. Notably, this had a negative impact on their overall perception of ease of process modelling, confidence, and overall service-composition experience. The non-programmers stressed the need to stay away from technical jargon and to further simplify the service composition tasks. Assisted composition, however, was liked by the non-programmers, owing that to its simplicity and ease of use, but was criticized for its lack of expressiveness and inflexibly by the programmers.

**Author Contributions:** Conceptualization, A.N.; data curation, A.N. and A.O.; formal analysis, A.N. and A.O.; funding acquisition, N.M.; methodology, A.N. and N.M.; visualization, A.N.; writing—original draft, A.N.; writing—review and editing, A.N. and N.M.

**Funding:** This research was funded by the European Commission, under grant number 215219.

**Conflicts of Interest:** The authors declare no conflict of interest.

## Appendix A

Please rate each of the following software experience questions on a 1–5 Likert scale ranging from '1 = Extremely poor' to '5 = Excellent'.

| | **Extremely Poor (1)** | **Below Average (2)** | **Average (3)** | **Above Average (4)** | **Excellent (5)** |
|---|---|---|---|---|---|
| My experience in software development using programming languages (e.g., Java, C++) is | ❍ | ❍ | ❍ | ❍ | ❍ |
| My experience in using software development environments (e.g., Eclipse, NetBeans, Microsoft Visual Studio) is | ❍ | ❍ | ❍ | ❍ | ❍ |
| My experience in software design using design notations (e.g., UML) is | ❍ | ❍ | ❍ | ❍ | ❍ |
| My experience in using semantic technologies (ontologies, annotations, reasoning) is | ❍ | ❍ | ❍ | ❍ | ❍ |
| My experience in Business Process Modelling (BPM) is | ❍ | ❍ | ❍ | ❍ | ❍ |
| My experience in web service composition is | ❍ | ❍ | ❍ | ❍ | ❍ |
| My experience in process life-cycle management (deployment, monitoring, launching) is | ❍ | ❍ | ❍ | ❍ | ❍ |

- Rate how interested are you in developing software applications

○ Not at All Interested (1)
○ Not Very Interested (2)
○ Neutral (3)
○ Somewhat Interested (4)
○ Very Interested (5)

- Rate how likely are you to develop software applications in the future

○ Very Unlikely (1)
○ Somewhat Unlikely (2)
○ Neutral (3)
○ Somewhat Likely (4)
○ Very Likely (5)

- What are your favourite software/service-development languages or platforms?

1.
2.
3.

- Gender:

○ Male (1)
○ Female (2)

- Specify your current job/course of studies: . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .

- Indicate the highest level of education that you have completed

○ High school (1)
○ Undergraduate (2)
○ Masters (3)
○ PhD (4)
○ Diploma/Certificate (5)
○ Other (6) ____________________

## References

1. Erl, T. *SOA Principles of Service Design*; Prentice Hall Press: Upper Saddle River, NJ, USA, 2007.
2. Van der Aalst, W.M.P.; Dumas, M.; ter Hofstede, A.H.M. Web service composition languages: Old wine in new bottles? In Proceedings of the 29th Euromicro Conference, Belek-Antalya, Turkey, 1–6 September 2003; pp. 298–305.
3. Scaffidi, C.; Shaw, M.; Myers, B. Estimating the numbers of end users and end user programmers. In Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '05), Dallas, TX, USA, 20–24 September 2005; pp. 207–214.
4. Mehandjiev, N.; Sutcliffe, A.; Lee, D. Organisational view of end-user development. In *End User Development, Human-Computer Interaction Series*; Lieberman, H., Paterno, F., Wulf, V., Eds.; Springer: Berlin, Germany, 2006; Volume 9, p. 492.
5. Ko, A.J.; Abraham, R.; Beckwith, L.; Blackwell, A.; Burnett, M.; Erwig, M.; Scaffidi, C.; Lawrance, J.; Lieberman, H.; Myers, B.; et al. The state of the art in end-user software engineering. *ACM Comput. Surv.* **2011**, *43*, 21. [CrossRef]
6. Norman, D.A. Some observations on mental models. In *Mental Models*; Psychology Press: Hove, UK, 2014; pp. 15–22.
7. Ganesarajah, D.; Lupu, E. Workflow-based composition of web-services: A business model or a programming paradigm? In Proceedings of the Sixth International Enterprise Distributed Object Computing Conference (EDOC'02), Lausanne, Switzerland, 20 September 2002; pp. 273–284.
8. Rao, J.; Su, X. A survey of automated web service composition methods. In *International Workshop on Semantic Web Services and Web Process Composition*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 43–54.
9. Boshernitsan, M.; Graham, S.L.; Hearst, M.A. Aligning development tools with the way programmers think about code changes. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, 28 April–3 May 2007; pp. 567–576.
10. Namoun, A.; Nestler, T.; de Angeli, A. Conceptual and usability issues in the composable web of software services. In Proceedings of the International Conference on Web Engineering, Vienna, Austria, 5–9 July 2010; pp. 396–407.
11. Namoun, A.; Nestler, T.; de Angeli, A. Service composition for non-programmers: Prospects, problems, and design recommendations. In Proceedings of the 2010 Eighth IEEE European Conference on Web Services (ECOWS '10), Washington, DC, USA, 1–3 December 2010; pp. 123–130.
12. Weber, I.; Paik, H.Y.; Benatallah, B. Form-based web service composition for domain experts. *ACM Trans. Web* **2013**, *8*, 2–40. [CrossRef]
13. Radeck, C.; Meißner, K. Assisted end user development for non-programmers: Awareness, exploration and explanation of composite web application functionality. In Proceedings of the International Conference on Web Information Systems and Technologies, Porto, Portugal, 25–27 April 2017; pp. 249–275.
14. Santos, M.; Villela, M.L.B. Characterizing end-user development solutions: A systematic literature review. In Proceedings of the International Conference on Human-Computer Interaction, Orlando, FL, USA, 26–31 July 2019; pp. 194–209.
15. Corno, F.; Russis, L.D.; Roffarello, A.M. RecRules: Recommending IF-THEN rules for end-user development. *ACM Trans. Intell. Syst. Technol.* **2019**, *10*, 58. [CrossRef]
16. Mehandjiev, N.; Namoune, A.; Wajid, U.; Macaulay, L.; Sutcliffe, A. End user service composition: Perceptions and requirements. In Proceedings of the 2010 Eighth IEEE European Conference on Web Services (ECOWS '10), Ayia Napa, Cyprus, 1–3 December 2010; pp. 139–146.
17. de Angeli, A.; Battocchi, A.; Chowdhury, S.R.; Rodriguez, C.; Daniel, F.; Casati, F. End-user requirements for wisdom-aware EUD. In *End User Development*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 245–250.
18. Cappiello, C.; Matera, M.; Picozzi, M. A UI-centric approach for the end-user development of multidevice mashups. *ACM Trans. Web* **2015**, *9*, 11. [CrossRef]
19. Zhao, L.; Loucopoulos, P.; Kavakli, E.; Letsholo, K.J. User studies on end-user service composition: A literature review and a design framework. *ACM Trans. Web* **2019**, *13*, 15. [CrossRef]
20. Daniel, F.; Matera, M. *Mashups: Concepts, Models and Architectures*; Springer: Berlin/Heidelberg, Germany, 2014.

21. Mehandjiev, N.; Namoun, A.; Lécué, F.; Wajid, U.; Kleanthous, G. End users developing mashups. In *Web Services Foundations*; Springer: New York, NY, USA, 2014; pp. 709–736.
22. Minhas, S. A Framework for Constructing End User Oriented Service Mashups. Ph.D. Thesis, The University of Manchester United Kingdom, Manchester, UK, 2017.
23. Cheng, B.; Zhai, Z.; Zhao, S.; Chen, J. LSMP: A lightweight service mashup platform for ordinary users. *IEEE Commun. Mag.* **2017**, *55*, 116–123. [CrossRef]
24. Lieberman, H.; Paternò, F.; Klann, M.; Wulf, V. End-USER DEVELOPMENT: An emerging paradigm. In *End User Development*; Lieberman, H., Paterno, F., Wulf, V., Eds.; Springer: Berlin, Germany, 2006; pp. 1–8.
25. Nardi, B.A. *A Small Matter of Programming: Perspectives on End User Computing*; MIT Press: Cambridge, MA, USA, 1993.
26. Lemos, A.L.; Daniel, F.; Benatallah, B. Web service composition: A survey of techniques and tools. *ACM Comput. Surv.* **2016**, *48*, 33. [CrossRef]
27. Ko, A.J.; Myers, B.A.; Aung, H.H. Six learning barriers in end-user programming systems. In Proceedings of the 2004 IEEE Symposium on Visual Languages-Human Centric Computing, Rome, Italy, 26–29 September 2004; pp. 199–206.
28. Abraham, R.; Burnett, M.M.; Erwig, M. Spreadsheet programming. In *Wiley Encyclopedia of Computer Science and Engineering*; John Wiley: Hoboken, NJ, USA, 2009; pp. 2804–2810.
29. Kongdenfha, W.; Benatallah, B.; Vayssière, J.; Saint-Paul, R.; Casati, F. Rapid development of spreadsheet-based web mashups. In Proceedings of the 18th International Conference on World Wide Web (WWW '09), Madrid, Spain, 20–24 April 2009; pp. 851–860.
30. Paternò, F.; Santoro, C. A design space for end user development in the time of the internet of things. In *New Perspectives in End-User Development*; Paternò, F., Santoro, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2017; pp. 43–59.
31. Sutcliffe, A. Evaluating the costs and benefits of end-user development. In *ACM SIGSOFT Software Engineering Notes*; ACM: New York, NY, USA, 2005; pp. 1–4.
32. McGill, T.; Klisc, C. End user perceptions of the benefits and risks of end user web development. *J. Organ. End User Comput.* **2006**, *18*, 22–42. [CrossRef]
33. Schulte, S.; Repp, N.; Eckert, J.; Berbner, R.; von Blanckenburg, K.; Schaarschmidt, R.; Steinmetz, R. Potential risks and benefits of Service-oriented Collaboration—Basic considerations and results from an empirical study. In Proceedings of the Second IEEE International Conference on Digital Ecosystems and Technologies (DEST 2008), Phitsanulok, Thailand, 26–29 February 2008; pp. 155–160.
34. Namoun, A.; Wajid, U.; Mehandjiev, N. A comparative study: Service-based application development by ordinary end users and IT professionals. In *Towards a Service-Based Internet*; Springer: Berlin/Heidelberg, Germany, 2010; pp. 163–174.
35. Burnett, M.; Fleming, S.D.; Shamsi, I.; Venolia, G.; Rajaram, V.; Farooq, U.; Grigoreanu, V.; Czerwinski, M. Gender differences and programming environments: Across programming populations. In Proceedings of the 2010 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement (ESEM '10), Bolzano-Bozen, Italy, 16–17 September 2010; p. 28.
36. DuSDar, S.; Schreiner, W. A survey on web services composition. *Int. J. Web Grid Serv.* **2005**, *1*, 1–30. [CrossRef]
37. Sheng, Q.Z.; Qiao, X.; Vasilakos, A.V.; Szabo, C.; Bourne, S.; Xu, X. Web services composition: A decade's overview. *Inf. Sci.* **2014**, *280*, 218–238. [CrossRef]
38. Martinez, A.; Patino-Martinez, M.; Jimenez-Peris, R.; Perez-Sorrosal, F. ZenFlow: A visual web service composition tool for BPEL4WS. In Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '05), Dallas, TX, USA, 20–24 September 2005; pp. 181–188.
39. Hosking, L.; Li, J.; Grundy, J. Visual modelling of complex business processes with trees, overlays and distortion-based displays. In Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '07), Coeur d'Alene, ID, USA, 23–27 September 2007; pp. 137–144.
40. Kelleher, C.; Pausch, R. Lowering the barriers to programming: A taxonomy of programming environments and languages for novice programmers. *ACM Comput. Surv.* **2005**, *37*, 83–137. [CrossRef]
41. Nestler, T.; Namoun, A.; Schill, A. End-user development of service-based interactive web applications at the presentation layer. In Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '11), Pisa, Italy, 13–16 June 2011; pp. 197–206.

42. Daniel, F.; Yu, J.; Benatallah, B.; Casati, F.; Matera, M.; Saint-Paul, R. Understanding UI integration: A survey of problems, technologies, and opportunities. *IEEE Internet Comput.* **2007**, *11*, 59–66. [CrossRef]
43. Yu, J.; Benatallah, B.; Casati, F.; Daniel, F. Understanding mashup development. *IEEE Internet Comput.* **2008**, *12*, 44–52. [CrossRef]
44. Hoyer, V.; Fischer, M. Market overview of enterprise mashup tools. In *Service-Oriented Computing*; Springer: Berlin/Heidelberg, Germany, 2008; pp. 708–721.
45. Jones, M.C.; Churchill, E.F. Conversations in developer communities: A preliminary analysis of the yahoo! Pipes community. In Proceedings of the fourth international conference on Communities and technologies (C&T '09), University Park, PA, USA, 25–27 June 2009; pp. 195–204.
46. Kuttal, S.K.; Sarma, A.; Swearngin, A.; Rothermel, G. Versioning for mashups—An exploratory study. In *International Symposium on End User Development*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 25–41.
47. Wong, J.; Hong, J.I. Making mashups with marmite: Towards end-user programming for the web. In Proceedings of the SIGCHI conference on Human factors in computing systems (CHI '07), San Jose, CA, USA, 26 April–3 May 2007; pp. 1435–1444.
48. Cao, J.; Riche, Y.; Wiedenbeck, S.; Burnett, M.; Grigoreanu, V. End-user mashup programming: Through the design lens. In Proceedings of the 28th International Conference on Human Factors in Computing Systems (CHI '10), Atlanta, GA, USA, 10–15 April 2010; pp. 1009–1018.
49. Zang, N.; Rosson, M.B.; Nasser, V. Mashups: Who? what? why? In Proceedings of the CHI '08 Extended Abstracts on Human Factors in Computing Systems (CHI EA '08), Florence, Italy, 5–10 April 2008; pp. 3171–3176.
50. Zang, N.; Rosson, M.B. What's in a mashup? And why? Studying the perceptions of web-active end users. In Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing VLHCC '08, Herrsching am Ammersee, Germany, 15–19 September 2008; pp. 31–38.
51. Al Sarraj, W.; de Troyer, O. Web mashup makers for casual users: A user experiment. In Proceedings of the 12th International Conference on Information Integration and Web-based Applications & Services (iiWAS '10), Paris, France, 8–10 November 2010; pp. 239–246.
52. Daniel, F.; Imran, M.; Kling, F.; Soi, S.; Casati, F.; Marchese, M. Developing domain-specific mashup tools for end users. In Proceedings of the 21st International Conference on World Wide Web, Lyon, France, 16–20 April 2012; pp. 491–492.
53. Radeck, C.; Blichmann, G.; Meißner, K. CapView–functionality-aware visual mashup development for non-programmers. In *Web Engineering*; Springer: Berlin/Heidelberg, Germany, 2013; pp. 140–155.
54. Aghaee, S.; Pautasso, C.; de Angeli, A. Natural end-user development of web mashups. In Proceedings of the 2013 IEEE Symposium on Visual Languages and Human Centric Computing (VL/HCC), San Jose, CA, USA, 15–19 September 2013; pp. 111–118.
55. Aghaee, S.; Pautasso, C. End-user development of mashups with naturalmash. *J. Vis. Lang. Comput.* **2014**, *25*, 414–432. [CrossRef]
56. Aghaee, S.; Pautasso, C. EnglishMash: Usability design for a natural mashup composition environment. In *Web Engineerin*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 109–120.
57. Fisher, A.; Margolis, J. Unlocking the clubhouse: The Carnegie Mellon experience. *ACM SIGCSE Bull.* **2002**, *34*, 79–83. [CrossRef]
58. Ramalingam, V.; LaBelle, D.; Wiedenbeck, S. Self-efficacy and mental models in learning to program. In *ACM SIGCSE Bulletin*; ACM: New York, NY, USA, 2004; Volume 36, pp. 171–175.
59. Loksa, D.; Ko, A.J.; Jernigan, W.; Oleson, A.; Mendez, C.J.; Burnett, M.M. Programming, problem solving, and self-awareness: Effects of explicit guidance. In Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems, San Jose, CA, USA, 7–12 May 2016; pp. 1449–1461.
60. De Raadt, M.; Watson, R.; Toleman, M. Chick sexing and novice programmers: Explicit instruction of problem solving strategies. In Proceedings of the 8th Australasian Conference on Computing Education, Hobart, Australia, 16–19 January 2006; pp. 55–62.
61. Jamoussi, Y. Towards an approach to guide end-user in interactive web services composition. In Proceedings of the International MultiConference of Engineers and Computer Scientists, Hong Kong, China, 18–20 March 2015.

62. Hang, F.; Zhao, L. Supporting end-user service composition: A systematic review of current activities and tools. In Proceedings of the 2015 IEEE International Conference on Web Services, New York, NY, USA, 27 June–2 July 2015; pp. 479–486.
63. Barricelli, B.R.; Cassano, F.; Fogli, D.; Piccinno, A. End-user development, end-user programming and end-user software engineering: A systematic mapping study. *J. Syst. Softw.* **2019**, *149*, 101–137. [CrossRef]
64. Akiki, P.A.; Bandara, A.K.; Yu, Y. Visual simple transformations: Empowering end-users to wire internet of things objects. *ACM Trans. Comput. Hum. Interact.* **2017**, *24*, 10. [CrossRef]
65. Rahmati, A.; Fernandes, E.; Jung, J.; Prakash, A. IFTTT vs. Zapier: A Comparative Study of Trigger-Action Programming Frameworks. *arXiv* **2017**, arXiv:1709.02788.
66. Coolican, H. *Research Methods and Statistics in Psychology*, 6th ed.; Psychology Press: New York, NY, USA, 2014.
67. Lewis, C.H. *Using the Thinking-Aloud Method in Cognitive Interface Design*; Research Report RC9265; IBM TJ Watson Research Center: Yorktown, NY, USA, 1982.
68. Lacy, S.; Riffe, D. Sampling error and selecting intercoder reliability samples for nominal content categories: Sins of omission and commission in mass communication quantitative research. *J. Mass Commun. Q.* **1996**, *73*, 969–973.
69. Tinsley, H.E.A.; Weiss, D.J. Interrater reliability and agreement. In *Handbook of Applied Multivariate Statistics and Mathematical Modeling*; Tinsley, H.E.A., Brown, S.D., Eds.; Academic Press: San Diego, CA, USA, 2000; pp. 95–124.
70. Visser, W.; Hoc, J.M. Expert software design strategies. In *Psychology of Programming*; Academic Press: Cambridge, MA, USA, 1991; pp. 235–249.
71. Desolda, G.; Ardito, C.; Costabile, M.F.; Matera, M. End-user composition of interactive applications through actionable UI components. *J. Vis. Lang. Comput.* **2017**, *42*, 46–59. [CrossRef]
72. Maćkowiak, M.; Nawrocki, J.; Ochodek, M. On some end-user programming constructs and their understandability. *J. Syst. Softw.* **2018**, *142*, 206–222. [CrossRef]
73. Good, J.; Howland, K. Programming language, natural language? Supporting the diverse computational activities of novice programmers. *J. Vis. Lang. Comput.* **2017**, *39*, 78–92. [CrossRef]
74. Desolda, G.; Ardito, C.; Matera, M. Empowering end users to customize their smart environments: Model, composition paradigms, and domain-specific tools. *ACM Trans. Comput. Hum. Interact.* **2017**, *24*, 12. [CrossRef]
75. Koedinger, K.R.; Aleven, V.; Heffernan, N.; McLaren, B.; Hockenberry, M. Opening the door to non-programmers: Authoring intelligent tutor behavior by demonstration. In *Intelligent Tutoring Systems*; Springer: Berlin/Heidelberg, Germany, 2004; pp. 162–174.
76. Pulido-Prieto, O.; Juárez-Martínez, U. A model for naturalistic programming with implementation. *Appl. Sci.* **2019**, *9*, 3936. [CrossRef]
77. Lieberman, H.; Liu, H. Feasibility studies for programming in natural language. In *End User Development*; Springer: Berlin, Germany; Dordrecht, The Netherlands, 2006; pp. 459–473.
78. Ur, B.; McManus, E.; Ho, M.P.Y.; Littman, M.L. Practical trigger-action programming in the smart home. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Toronto, ON, Canada, 26 April–1 May 2014; pp. 803–812.
79. Ur, B.; Ho, M.P.Y.; Brawner, S.; Lee, J.; Mennicken, S.; Picard, N.; Schulze, D.; Littman, M.L. Trigger-action programming in the wild: An analysis of 200,000 IFTTT recipes. In Proceedings of the 2016 Chi Conference On Human Factors in Computing Systems, San Jose, CA, USA, 7–12 May 2016; pp. 3227–3231.
80. Gross, P.; Kelleher, C. Non-programmers identifying functionality in unfamiliar code: Strategies and barriers. *J. Vis. Lang. Comput.* **2010**, *21*, 263–276. [CrossRef]
81. Chen, M.; Tan, T.H.; Sun, J.; Liu, Y.; Dong, J.S. Veriws: A tool for verification of combined functional and non-functional requirements of web service composition. In Proceedings of the 36th International Conference on Software Engineering, Hyderabad, India, 31 May–7 June 2014; pp. 564–567.