Science of Computer Programming 172 (2019) 232–263

Contents lists available at ScienceDirect

Science of Computer Programming

www.elsevier.com/locate/scico

An empirical comparative evaluation of gestUI to include gesture-based interaction in user interfaces

Otto Parra [a,b,*], Sergio España [c], Jose Ignacio Panach [d], Oscar Pastor [b]

[a] Computer Science Department, Universidad de Cuenca, Ecuador
[b] PROS Research Centre, Universitat Politécnica de Valencia, Spain
[c] Department of Information and Computing Sciences, Utrecht University, the Netherlands
[d] Escola Tècnica Superior d'Enginyeria, Departament d'Informàtica, Universitat de València, Spain

ARTICLE INFO

Article history:
Received 6 March 2018
Received in revised form 30 November 2018
Accepted 3 December 2018
Available online 6 December 2018

Keywords:
Model-driven method
Human–computer interaction
Code-centric method
Gesture-based interaction
Comparative empirical evaluation

ABSTRACT

Currently there are tools that support the customisation of users' gestures. In general, the inclusion of new gestures implies writing new lines of code that strongly depend on the target platform where the system is run. In order to avoid this platform dependency, gestUI was proposed as a model-driven method that permits (i) the definition of custom touch-based gestures, and (ii) the inclusion of the gesture-based interaction in existing user interfaces on desktop computing platforms. The objective of this work is to compare gestUI (a MDD method to deal with gestures) versus a code-centric method to include gesture-based interaction in user interfaces. In order to perform the comparison, we analyse usability through effectiveness, efficiency and satisfaction. Satisfaction can be measured using the subjects' perceived ease of use, perceived usefulness and intention to use. The experiment was carried out by 21 subjects, who are computer science M.Sc. and Ph.D. students. We use a crossover design, where each subject applied both methods to perform the experiment. Subjects performed tasks related to custom gesture definition and modification of the source code of the user interface to include gesture-based interaction. The data was collected using questionnaires and analysed using non-parametric statistical tests. The results show that gestUI is more efficient and effective. Moreover, results conclude that gestUI is perceived as easier to use than the code-centric method. According to these results, gestUI is a promising method to define custom gestures and to include gesture-based interaction in existing user interfaces of desktop-computing software systems.

1. Introduction

Gesture-based interfaces are harder to implement and test than traditional interfaces (i.e. interfaces based on WIMP – Window–Icon–Mouse–Pointer) using a mouse and a pointer [1] because they require more skills and knowledge of the software engineers about programming languages and tools to write source code. There are some complications in the definition of custom gestures (in this paper, the word "gesture" is used to refer to touch-based gestures) and their inclusion in user interfaces. Gesture-based interaction is supported at the source code level (typically third-generation languages)

* Corresponding author.
E-mail addresses: otto.parra@ucuenca.edu.ec, otpargon@upv.es (O. Parra), s.espana@uu.nl (S. España), joigpana@uv.es (J.I. Panach), opastor@dsic.upv.es (O. Pastor).

https://doi.org/10.1016/j.scico.2018.12.001
0167-6423/

[2] that is, using a code-centric method where the developers write source code using a programming language in order to implement user interfaces with gesture-based interaction included. This involves a great effort with regard to coding and maintenance when multiple platforms are targeted [3], has a negative impact on reusability and portability, and it complicates the definition of new gestures [4].

We proposed gestUI, a model-driven method described in [5], to help in the definition of custom gestures and in the inclusion of gesture-based interaction in existing user interfaces for desktops. The choice of gestUI is due to the fact that gestUI is the only Model-Driven Development method that allows the end-user to personalise their own gestures. gestUI method employs a model-to-model (M2M) transformation using ATL (http://eclipse.org/atl/) and a model-to-text (M2T) transformation using Acceleo (http://www.eclipse.org/acceleo/) to obtain a user interface with gesture-based interaction included in its functionalities. Regarding the gesture recognition required in our work, there are several methods to consider, i.e.: $-family ($1 designed for unistroke gestures [6], $N designed for multistroke gestures [7], $P designed for memory reduction [8], $Q for mobile, wearable and embedded devices [9] and variants), !FTL [10], etc. In our work, we consider multi-stroke gestures and we adopt $N because this recogniser supports this type of gestures and it is easy to understand and easy to implement. According to [7], $N supports XML language to describe a gesture, therefore, gestUI supports gestures defined in XML language.

Moody [11] considers that the objective of the validation should not be to demonstrate that the method is "correct" but that the method could be adopted based on its pragmatic success which is defined as "the efficiency and effectiveness with which a method achieves its objectives". According to ISO 9241-210 [12] and ISO 25062-2006 [13], usability is defined as "the extent to which a product can be used by specified users to achieve specified goals with effectiveness, efficiency and satisfaction in a specified context of use". Additionally, ISO 25062-2006 establishes that usability evaluation involves using (1) subjects who are representative of the target population of users of the software, (2) representative tasks, and (3) measures of efficiency, effectiveness and subjective satisfaction. The ISO also defines that at least one indicator in each of these aspects should be measured to determine the level of usability achieved [2]. In order to evaluate satisfaction, we consider the Method Evaluation Model (MEM) [1,3] which considers three primary constructs: perceived ease of use – PEOU ("the degree to which a person believes that using a particular system would be free of effort"), perceived usefulness – PU ("the degree to which a person believes that using a particular system would enhance his or her job performance") and intention of use – ITU ("the extent to which a person intends to use a particular system").

The main contribution of this paper is the design and analysis of an experiment that compares two types of methods to define custom gestures and the inclusion of gesture-based interaction in user interfaces: code-centric and model-driven. With the aim of validating gestUI, we have designed a comparative empirical evaluation in which we consider both methods to define custom gestures and to include the gesture-based interaction. We evaluate efficiency, effectiveness and satisfaction (by means of PEOU, PU and ITU) when the subjects apply gestUI in comparison with a code-centric method to include gesture-based interaction in existing user interfaces. We used as indicators: (i) the time to finish the task for efficiency; (ii) the percentage of correct tasks carried out in the experiment for effectiveness, and (iii) the MEM questionnaires for PEOU, PU and ITU. Results of this evaluation help to know to which extent the use of Model-driven Development (MDD) helps in the process to define custom gestures and to include gesture-based interaction in user interfaces.

We present and discuss the results obtained in the experiment. We have found that the results obtained for efficiency and effectiveness are better when the subjects use gestUI in relation to when they employ a code-centric method. Also, the results achieved for PEOU, PU and ITU are better if the subjects use gestUI in relation to when they employ a code-centric method.

This paper is organised as follows. Section 2 presents the related work. Section 3 describes two methods for including gesture-based interaction: the code-centric method and the model-driven method – gestUI. Section 4 describes the experimental planning. Section 5 reports the results of the experiment. Section 6 includes a discussion about the results obtained in the experiment. Section 7 contains conclusions and outlines our future work.

2. Related work

In this section, we analyse the related work about comparative evaluation between methods based on a model-driven paradigm and other existing methods (e.g. traditional software development methods) to develop software. There are several works that report the comparison of experiments using MDD, some of them are described in the following paragraphs and summarised in Table 1. The following information is included in each column: author(s) of the paper, year of publication, type of study (e.g. case study, evaluation, controlled experiment), goal of the study, field of application of the experiment, variables, experimental subject, and the tool used in the experiment.

Kapteijns et al. [14] describe a case study of the development of a small middleware application in order to make a comparison between MDD implementation with regular third-generation programming. The MDD framework used, which is called XuWare, allows the generation of a "create–remove–update–delete" functionality for Web applications from Unified Modelling Language (UML) models. Results obtained show that MDD is highly applicable to small-scale development projects under conditions which can easily be satisfied.

Bunse et al. [15] describe a case study in order to compare MARMOT (based on MDD and component-based development) with RUP and Agile Development. In this evaluation, subjects developed a small control system for an exterior car mirror. The metrics employed in the evaluation are: model-size, amount of reused elements, defect density and, devel-

Table 1
Related work.

Author	Year	Type of study	Goal	Application field	Variables	Experimental subjects	Tool employed in the experiment
Kapteijns et al. [14]	2009	Case study	To compare an MDD implementation with regular third generation programming	A small middleware application	Development productivity, development time, application complexity	Novice and expert developers	XuWare
Bunse et al. [15]	2009	Case study	To compare MARMOT (based on MDD and component-based development) with RUP and Agile Development	A small control system for an exterior car mirror	Development effort, model size, defect density, amount of reused elements, model size	Graduate students	MARMOT
Kane et al. [16]	2011	User study	To compare the behaviour of two types of users (blind and sighted) when they use gestures in touch-based user interfaces	An application for a Tablet PC		Blind and sighted people	
Ricca et al. [17]	2012	Controlled experiment	To compare UniMod programming with code-centric programming	Two new versions of Svetofor and Telepay: Svetofor+ and Telepay+	Effectiveness of UniMod	Bachelor degree students	UniMod versions of Svetofor and Telepay
Martinez et al. [21]	2012	Quasi-experiment	To compare the productivity and satisfaction of junior Web developers developing the business layer of a Web 2.0 Application using three methods: code-centric, model-based or a MDE approach	Web 2.0 application	Productivity and satisfaction	Master's degree students	A set of tools to implement the business layer of a web application
Papotti et al. [18]	2013	Quantitative study	To evaluate the impact of using model-driven code generation vs. traditional development	Web application	Development time	Undergraduate students	A set of tools to implement
Martinez et al. [20]	2013	Quasi-experiment	To compare three methods: model-driven, model-based and code-centric	Web application	Perceived usefulness, perceived ease of use, compatibility, subjective norm, voluntariness	Graduate students	A set of tools to implement the business layer of a web application
Condori-Fernandez et al. [19]	2013	Empirical approach	Usability of model-driven tools		Efficiency, Effectiveness and satisfaction	PROS-UPV Researchers	INTEGRANOVA
Martinez et al. [22]	2014	Empirical study	Maintainability	Web application	Performance and satisfaction	Graduate students	OOH4RIA
Cervera et al. [23]	2015	Empirical evaluation	To evaluate usefulness and ease of use of MOSKitt4ME		Perceived usefulness, ease of use	Master's and PhD students, a post-doc, industrial software engineers	MOSKitt4ME
Panach et al. [24]	2015	Experiment	To verify some of the most cited benefits of MDD	Web application	Quality, effort, productivity and satisfaction	Final-year Master's degree students	INTEGRANOVA
Safdar et al. [25]	2015	Experiment	To compare the productivity of the software engineers when they use UML modelling tools	Modelling software processes	Learnability and memory load	Undergraduate and graduate students	IBM Rational Software Architecture, Papyrus, and MagicDraw
Neto et al.	2017	Comparative study	To compare the proposed framework with other existing frameworks	Credit Behavioural Scoring solutions	Performance (time), Effectiveness		RelAggs, CoMoVi, and CbMVV frameworks
Santos et al. [27]	2018	Empirical evaluation	To measure the effectiveness in terms of productivity	Developing signal control agents in the domain of ATSC simulations with a MDD approach	Productivity		DSL4ABMS
Hamid and Weber [28]	2018	Empirical evaluation	To measure effort and feasibility of MDE application in the metrology domain	Developing a practical application to a use case in the metrology domain with strong security requirements		Industry practitioners (Twenty people experts in the engineering secure systems)	
Oliveira et al. [29]	2018	Empirical evaluation	To compare BRCode with genMDE	Development of Enterprise Information Systems in industry	Development effort and financial gains	Software developers	An Enterprise Resource Planning (ERP) system

opment effort. Their evaluation reveals that model-driven, component-oriented development performs well and leads to maintainable systems and a higher-than-normal reuse rate.

Kane et al. [16] describe two user studies that compared how blind people and sighted people use touch screen gestures. The authors of this research describe some design considerations related with touch-based user interfaces in different types of devices and the definition of touch-based gestures by both types of users. They conducted a study in which both blind and sighted participants were asked to invent gestures that could be used to conduct standard computing tasks on a touch screen-based tablet PC. To determine if there were significant differences in how blind and sighted people performed the same gestures, the authors conducted a second study in which all participants performed the same set of standard gestures. Based on the results of these two studies, they offer preliminary advice on how to design future touch screen-based applications for both blind and sighted users.

Ricca et al. [17] describe a controlled experiment with the aim of investigating the effectiveness of Model-driven development during software maintenance and evolution activities. Subjects (Bachelor degree students) used two software systems (Svetofor and Telepay) and by means of UniMod obtained two new versions of these software systems. In this experiment, the results showed a marked reduction in time to complete the maintenance tasks, with no important impact on correctness when UniMod is used instead of conventional programming.

Papotti et al. [18] describe a quantitative study in order to evaluate the impact of using model-driven code generation vs. traditional development of software systems to implement a web application. Results show that the development time to code generation is shorter than the time required when using traditional development.

Condori-Fernandez et al. [19] describe an empirical approach for evaluating the usability of model-driven tools. They propose a framework to evaluate the usability, applying it to INTEGRANOVA, an industrial tool that implements an MDD software development method called the OO-Method. The authors report results about the usability evaluation in terms of efficiency, effectiveness and satisfaction within an experimental context.

Martinez et al. [20] describe a quasi-experiment in order to compare three methods (model-driven, model-based and code-centric) developing the business layer of a Web 2.0 application. Results show that MDD approaches are the most difficult to use but, at the same time, are considered to be the most suitable in the long term. Additionally, these authors in [21] report a quasi-experiment in order to evaluate productivity and satisfaction when a group of Master's degree students develop a Web application using three methods: code-centric, model-based (UML) and model-driven (OOH4RIA). Results show that the use of model-driven Engineering practices significantly increases both productivity and satisfaction of junior Web developers, regardless of the particular application. Other work reported by these authors [22] concerns an empirical study on the maintainability of Web applications. In this work, they compare model-driven Engineering with the code-centric method by using OOH4RIA and Visual Studio .NET, respectively. The results show that maintaining Web applications with OOH4RIA clearly improves the performance of the subjects.

Cervera et al. [23] describe an empirical evaluation using TAM and Think Aloud methods to assess usefulness and ease of use of MOSKitt4ME. The results were favourable, that is, MOSKitt4ME was highly rated in perceived usefulness and ease of use; the authors also obtained positive results with respect to the users' actual performance and the difficulty.

Panach et al. [24] describe an experiment in order to compare quality, effort, productivity and satisfaction of MDD and traditional development. Subjects (final-year Master's degree students) built two web applications from scratch. Results obtained show that for small systems, and less programming-experienced subjects, MDD does not always yield better results than a traditional method, even considering effort and productivity.

Safdar et al. [25] describe an experiment, in the context of model-driven software engineering (MDSE), with undergraduate and graduate students in order to compare the productivity of the software engineers while modelling with UML tools (IBM Rational Software Architecture, Papyrus, and MagicDraw). The authors measure the productivity in terms of modelling effort required to correctly complete a task, learnability, time and number of clicks, and memory load required for the software engineer to complete a task. Their results show that MagicDraw yields good results in terms of learnability, memory load, and completeness of tasks.

Neto et al. [26] propose a framework based on MDD to systemise the pre-processing stage in knowledge discovery projects. This stage is a complex task that demands from database designers a strong interaction with experts having a broad knowledge about the application domain. In order to validate the proposed framework, two comparative studies were conducted to show that the proposed framework delivers a performance equivalent or superior to those of existing frameworks and reduces the time of data transformation with a confidence level of 95%.

Santos et al. [27] describe their proposal composed of a modelling language and model-to-code transformations for producing runnable simulations automatically. In order to analyse the productivity of their MDD approach, they compared the amount of design and implementation artefacts produced using their approach and traditional simulation platforms. The authors report an evaluation of the effectiveness of their MDD approach consisting in an empirical study in order to measure the productivity of their MDD approach. The results show that their MDD approach is effective for reducing the effort required for developing agent-based simulations.

Hamid and Weber [28] describe an approach based on metamodelling and model transformation techniques to define patterns at different abstraction levels and to generate different representations according to the target domain. They report an empirical evaluation of the proposed approach through a practical application to an industrial use case in the metrology domain with strong security requirements. The case study aimed to determine whether the pattern-based approach leads

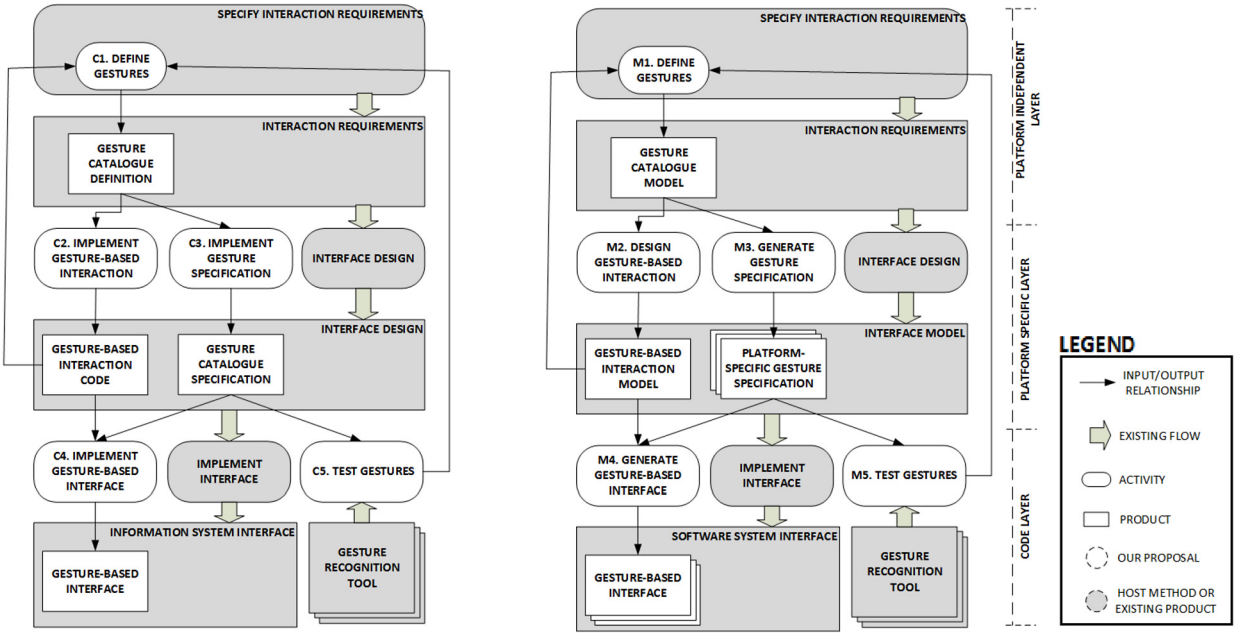

Fig. 1. Methods overview: code-centric (left) and model-driven (gestUI) method (right).

to a reduced number or to a simplification of the engineering process steps. Results reveal that domain experts perceived the approach to be extremely useful and agreed regarding the benefits of adopting the approach in a real industrial context.

Oliveira et al. [29] report a comparative empirical evaluation between BRCode (an interpretative MDE approach for fast-changing Enterprise applications) and genMDE (a generative MDE approach). They use a case study that involves the development of an Enterprise Resource Planning (ERP) system data collection based on 34 realistic scenarios in a Brazilian company. The case study was conducted to evaluate the proposed technique by investigating the impact of the proposed approach on the development effort and financial gains. The results show the strength of using BRCode to support software production companies in the business environment.

All these works describe comparative evaluations in order to check whether or not model-driven produces better results than other methods (e.g. code-centric method, method based on RUP and Agile methodology). The types of study used in these evaluations are mainly case studies, empirical evaluations and quantitative studies. As far as the authors know, there are no previous experiments that dealt with the comparison of a model-driven versus a code-centric method in the context of generating gesture-based interaction. So, this paper is a step forward in the process of covering this gap.

Apart from code-centric and gestUI, there are some commercial products to define customised gestures based on standard gestures (e.g., Touch Me Gesture Studio of Microsoft [30], ASUS Smart Gesture [31], etc.). The gestures obtained with these products could be added in a user interface as an additional task performed by developers.

3. Methods to include gesture-based interaction

Next, we describe the two methods used in the experiment to include gesture-based interaction in an existing user interface [32]: code-centric and model-driven. Both methods are shown in Fig. 1, code-centric on the left and model-driven on the right. In this figure, we use a similar representation to show the two methods, with the purpose of facilitating the understanding of the process to be followed in the comparative evaluation described in this paper.

Although the two methods are shown in a similar way, it is necessary to indicate that the code-centric method is based on tasks related to the source code, while the object-oriented method bases its actions on related activities with the treatment of models, as explained in this section.

Fig. 1 shows the user interface development life cycle for both methods. In both cases we start from existing activities and products (represented by means of colour grey) used to develop interfaces that must be enhanced to support gesture-based interaction and a set of new activities and products (represented by means of the colour white) that deal explicitly with the gesture-based interaction. In the following sections we describe proposed activities and products of each method.

3.1. Code-centric method

The **code-centric method** consists in a set of tasks [33] (e.g. conceptualisation and requirements gathering, analysis and functional description, design, coding, testing and deployment) related with the implementation of a software system using

a programming language and a tool where software engineers work entirely by editing source code. In this method, software engineers employ the integrated development environment (IDE) available in some case tool (e.g. Microsoft Visual Studio,[1] Eclipse Window Builder,[2] NetBeans,[3] etc.) which includes toolbars and wizards to construct the user interface by including forms, panels, buttons, etc. The next step in this process is the code writing to include the logic required in the software and to complete the development of the user interface.

Specifically, the work [34] describes an example of this method to develop a user interface by means of Eclipse SWT Designer (Window Builder). This toolkit does not include components to define custom gestures nor to include gesture-based interaction. SWT works under the assumption that the user interface is already implemented and the developer writes additional source code containing gesture-based interaction. The final product obtained in the process is the source code which is compiled in order to implement the user interface.

The set of activities to perform with the aim of including gesture-based interaction in an existing source code through the code-centric method (Fig. 1, left) is detailed in the following paragraphs:

1. Activity C1: this allows software engineers to define the gestures requirement specification (by means of a language to specify requirements, e.g. text) which makes up the gesture catalogue and the actions to be performed using such gesture catalogue. The product obtained in this process is a requirements document containing the specification of the interaction between gestures and actions included in a user interface.
2. Activity C2: this permits software engineers to select the user interface to include the gesture-based interaction according to the aforementioned requirements specification, then he/she analyses the source code of the selected user interface with the aim of determining the actions included in the user interface source code. The software engineer defines the gesture–action correspondence by specifying the gesture that allows the execution of an action included in the user interface.
3. Activity C3: this allows software engineers to specify, by means of XML language each gesture included in the requirements document of the gesture catalogue. This gestures specification is required in order to be supported by the gesture recogniser algorithm. In this work we use $N [7] as the gesture recogniser. The product obtained in this step is the gesture catalogue specification written in XML.
4. Activity C4: in this activity the software engineer writes the source code that implements the methods needed to execute the actions specified with the previously defined gestures, that is, the software engineer combines two products (i) gesture-based interaction source code and (ii) gesture catalogue specification in order to obtain the gesture-based user interface. The product obtained in this last step is the user interface source code including gesture-based interaction.
5. Activity C5: this permits testing gestures using existing frameworks (e.g. quill [35], iGesture [36], $N [7]). The gesture catalogue is generated according to the gesture definition of each framework, hence the users sketch gestures in each framework in order to test the definition of each gesture. It is important to indicate that quill and iGesture have been used only as target platforms to perform validity tests of the generation of the gesture catalogue, that is, to verify that the result of the M2T transformation (definition of a gesture using XML) is adequate and correct. By other side, the $N transformation is performed in order to check if the definition obtained is correct and to test the gesture in the canvas included in the website of University of Washington,[4] where is described gesture recogniser algorithm called $N.

There are activities represented in Fig. 1 (e.g. implement interfaces, interface design) whose functionality is included in the process of development of user interfaces using some tools available. These activities are not described in this paper because we consider that these activities belong to traditional development methods for obtaining user interfaces by using typical development tools.

When a software engineer employs a code-centric method to include gesture-based interaction, some of the following problems are involved [37–39]: (i) the amount of time required to implement this type of interaction. Depending on of the case tool used to obtain the source code required to implement the user interface, the amount of time could be high. In this case, software engineers have two options to obtain it: (a) writing the source code from scratch or (b) adapting existing source code; (ii) the gesture specification is not multi-platform; (iii) it is hard to reuse the source code to support gesture-based interaction in other platforms; (iv) software engineers require skills in the programming language of each platform employed in the implementation of user interfaces of information system; (v) in some cases, the Integrated Development Environment (IDE) is not available in all platforms required by users. This situation complicates the job of the software engineers because they need more training in order to use different IDEs in each platform.

[1] See https://docs.microsoft.com/en-us/windows/desktop/appuistart/implementing-a-user-interface to check the process to build a user interface using Visual Studio of Microsoft.

[2] See http://www.vogella.com/tutorials/EclipseWindowBuilder/article.html to check the process to build a user interface using Eclipse.

[3] See https://netbeans.org/kb/docs/java/quickstart-gui.html to check the process to build a user interface using Netbeans.

[4] https://depts.washington.edu/madlab/proj/dollar/ndollar.html.

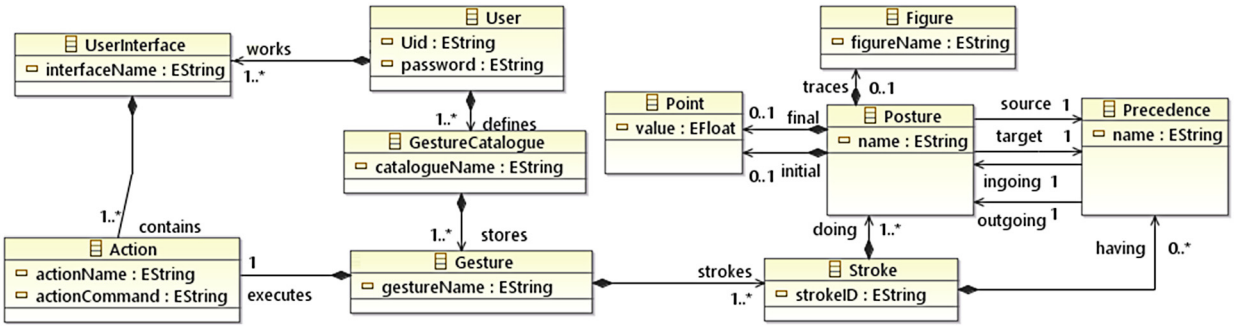

Fig. 2. gestUI metamodel (taken from [42]).

3.2. gestUI: a model-driven method

Unlike a code-centric method, a model-driven method requires the definition of models and model transformations for the generation of source code in order to obtain, in this case, a user interface. Software engineers do not require the use of an IDE to develop a user interface, making it more affordable the process of obtaining the source code of the user interface by means of automatic generation.

In order to tackle the problems described in Section 3.1, and that arise when a code-centric method is used, software engineers have another alternative to implement a software system: using a model-driven paradigm. In this case, software engineers use tools to describe the structure and behaviour of their software system using conceptual models, and source code is generated automatically through transformation rules. An example of this method is gestUI [5], which allows the inclusion of gesture-based interaction from conceptual models without writing any line of code (Fig. 1, right).

gestUI [5] is a model-driven method that permits the definition of custom gestures and the inclusion of gesture-based interaction in user interfaces for desktop-computing (Fig. 1, right) by means of models and model transformations [40]. Gesture-based interaction is based on the selection and use of functions provided in software systems by means of gestures on touch-based devices [41]. gestUI is model-driven since its main artefacts are conceptual models (Fig. 2).

We consider that a user interface is used by one or more users. Each user can define his/her own gesture catalogue containing one or more gestures; each gesture permits an action to be executed. This action is contained in the user interface. Each gesture is composed of one or more strokes defined by postures, and described by means of coordinates (X, Y). The sequence of strokes in the gesture is specified by means of an order of precedence. Each posture in a gesture is related to a figure (line, circle, etc.) with an orientation (up, down, left, right), and is qualified by a state (initial, executing, final). Single-stroke gestures and multi-stroke gestures are defined by the number of strokes in the gesture.

gestUI is composed of three layers according to the model-driven method: a platform-independent layer, a platform-specific layer and source code, as shown in Fig. 1, right. Also, this method is iterative and user-centric: (i) it is iterative because if the users are not satisfied with the definition of gestures, they can repeat the process (i.e. gestUI provides two loopbacks in order to repeat the process to define gestures), and (ii) it is user-centric because the users are the main actors in the process of defining custom gestures and in the inclusion of gesture-based interaction in user interfaces.

During the process of code generation, an option is added to the information system which allows redefining any gesture that is difficult for the user to trace or remember during the runtime of the information system [42,43]. In this context, tailoring mechanisms are provided by gestUI to define custom gestures for each user and to modify this definition during the execution stage. In this context, each user decides the gesture-based interaction elements in the beginning of the software development life cycle, which will be applied in the user interface in the execution stage. The software engineer then includes this specification in the user interface and finally, when the user interface is ready, the user performs actions using the previously defined interaction elements. However, if the user wants to change the initial specification of a gesture-based interaction then it is necessary to include the tools required to consider the modification of the gesture catalogue specification with the aim of improving the interaction process.

gestUI is designed to be inserted in any existing user interface development method. Within this context, Fig. 1, right, shows the activities and products (grey background) of an existing method to build interfaces that aim to be enhanced with gestUI and activities and products specific of gestUI (white background). For example, if a software engineer uses Eclipse to generate a user interface during the stage of implementation, it is possible to take the source code of this user interface in order to apply gestUI to include gesture-based interaction and to obtain a user interface supporting gestures.

In this work, in order to tackle the problems indicated in Section 3.1, we use gestUI to obtain user interfaces with gesture-based interaction included. Additionally, MDD helps to improve the quality of the product (the user interface) in terms of productivity, portability, interoperability, reusability. Table 2 shows the relationship between the aforementioned problems and such factors.

Fig. 3 shows three systems involved in the process described in this paper using gestUI: the information system with interfaces where we aim to include gesture-based interaction, the gestUI tool to include the gesture-based interaction, and a framework to test the gestures defined using gestUI (i.e. quill, iGesture, $N).

Table 2
Relationship between detected problems and factors of the quality of the product improved with MDD.

Problems described in Section 3.1	Factors of the quality of the product improved by using MDD [33]
Problem (iv): Software engineers require skills in the programming language of each platform.	Productivity
Problems (ii): The gesture specification is not multi-platform.	Portability
Problem (v): IDE is not available in all platforms required by users.	Interoperability
Problem (iii): It is hard to reuse the source code to support gesture-based interaction in other platforms.	Reusability
Problem (i): The amount of time required to implement this type of interaction depending of the method to obtain the source code.	Source code automatic generation

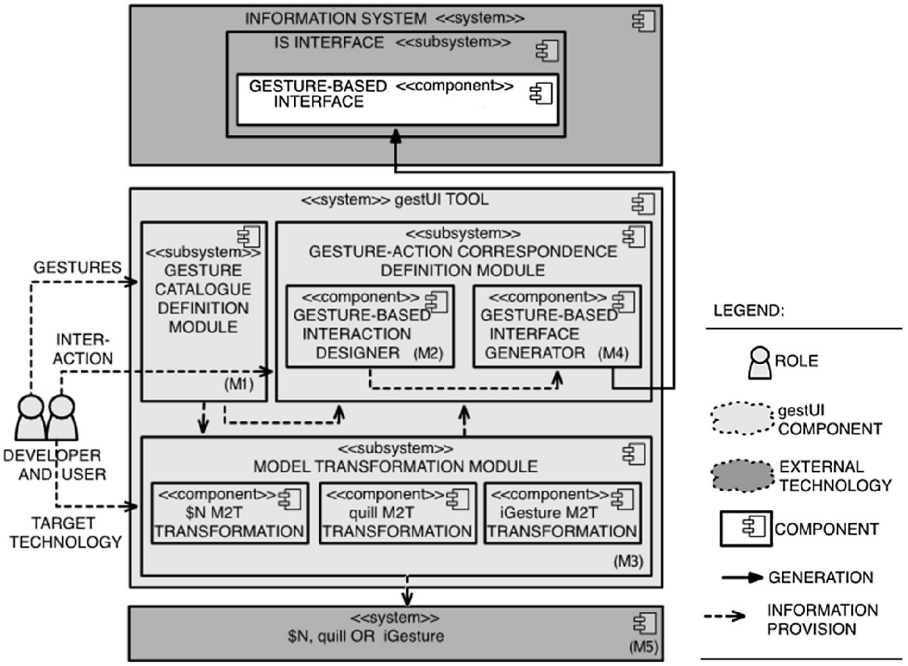

Fig. 3. gestUI tool support.

Regarding the gestUI system, we have developed a tool support [5] using the Java programming language and the Eclipse Modelling Framework to implement it. This tool is composed of three subsystems, as shown in Fig. 3: Gesture Catalogue Definition Module, Gesture–Action Correspondence Definition Module and Model Transformation Module. Next we describe these subsystems.

1. Subsystem "Gesture Catalogue Definition Module". This contains the M1 activity (Fig. 1, right):
   (i) **M1 Activity:** The subject draws custom gestures using a finger (or a pen/stylus) on a touch-based screen. Each gesture is stored in a repository. Then, in order to define the platform-independent gesture catalogue, the subject chooses one or more gestures from the repository and then they are inserted in the gesture catalogue model. This catalogue model (compliant with the metamodel described in [5]) is the input for the "Model Transformation Module" and the "Gesture–Action Correspondence Definition Module" subsystems.
2. Subsystem "Gesture–Action Correspondence Definition Module". This contains two components:
   (A) Component: "Gesture-Based Interaction Designer". The inputs for this component are: the gesture catalogue model (from M1) and the user interface to include gesture-based interaction. This subsystem contains the M2 activity (Fig. 1, right):
   (ii) **M2 Activity:** This defines the gesture–action correspondence through the following process: (1) selecting a user interface source code with the aim of analysing it and finding the actions included in it by applying a parsing process. The parsing process permits the discovery of a set of actions by means of checking the source code to search strings (or substrings) containing keywords (e.g. in the Java programming language: JButton, JPanel) [44]. Some complications can occur in the parsing process, especially depending on the programming language in which the user interface is written and the components that have been included in that interface. Therefore, the rules that are included in the parser must be very accurate in order to be able to exhaustively analyse the source code of a user interface and, in this way, find the actions included in the interface as a previous step to the inclusion of gesture-based interaction.

The process of defining gesture–action correspondence takes as input two arguments: (i) the previously defined gesture catalogue model with the aim of assigning each gesture with an action; (ii) the source code of a user interface to search keywords related with actions contained in the structure of source code that is based on a programming language such as Java (e.g. JButton to define a button, JPanel to define a panel). As a result of this process we obtain a set of actions included in the user interface. Therefore, if any action is found, a one-to-one relationship is defined between this action and a gesture.

(B) Component: "Gesture-Based Interface Generator". The inputs for this component are: gesture–action correspondence and the user interface source code. The output of this component is the new version of the user interface source code. It contains the M4 activity (Fig. 1, right):

(iii) **M4 Activity:** This executes a code generation process in order to obtain the new version of the user interface source code containing gesture-based interaction. By using an automatic process, we insert each gesture–action correspondence in the corresponding component of the user interface. This process is iterative while any action is found in the source code of the user interface. Finally, we apply a code generation obtaining the user interface with gesture-based interaction included.

3. Subsystem "Model Transformation Module". The inputs for this subsystem are: gesture catalogue model and the target platform to perform the model transformations. This subsystem contains the M3 activity (Fig. 1, right):

(iv) **M3 Activity:** This includes the transformation rules and the scripts written in ATL and Acceleo to apply M2M and M2T transformations, respectively. This activity requires two inputs: the gesture catalogue definition model and the target technology. Firstly, a M2M transformation is performed to obtain the gesture catalogue model according to the specification of the gestures to be used in the gesture recogniser algorithm. In this case, we consider as the target platform the $N gesture recogniser and we obtain the platform-specific gesture catalogue specification. Secondly, an M2T transformation is performed to obtain the gesture catalogue described in XML in order to include it in the user interface. Thirdly, a M2T transformation is performed to obtain the user interface source code including gesture-based interaction. Additionally, in order to test these transformations, an additional M2T transformation is performed to obtain a gesture catalogue to be included in two frameworks to test gestures: (i) quill [35] using GDT 2.0 to describe the gesture catalogue and (ii) iGesture [36] using XML to describe the gesture catalogue. Regarding the source code generated by gestUI, it can be in any programming language used to implement user interfaces (e.g. Visual Basic, C#, Java). In this case, we need to specify in the transformation rules the keywords of the programming language used to implement the user interface. In this work, we use Java as programming language to generate source code of a user interface.

Additional information about models and model transformations included in gestUI can be found in [40,42].

3.3. Differences between code-centric method and gestUI (model-driven method)

There are some differences to consider between both methods in our work:

a. In the code-centric method the developer needs to specify the custom gesture by means of XML. The custom gesture definition using gestUI is by means of a canvas that helps the developers to sketch the gesture. A module in gestUI captures each point of the drawing done with the user's finger.
b. The code-centric method is based on the writing of source code to include gesture-based interaction by means of an IDE. In our experiment, the developer employs Java as programming language to include gesture-based interaction in a user interface. gestUI does not require an IDE to write source code. gestUI is based on the specification of information to execute model transformations in order to obtain the source code in Java, which includes gesture-based interaction in a user interface. The target programming language (i.e. Java) can be chosen and it depends on the target system where the code will be inserted.
c. Using the code centric method, the developer needs to deal with several factors: knowledge of the programming language, knowledge of the target platform, experience using the IDE to write the code, etc. Using gestUI, the developer does not need to know the characteristics of each programming language or platform. Developers only need to know how to deal with conceptual models, relegating coding particularities to model to code transformations.

In summary, the differences are directly related with the factors to improve the quality of the software product discussed in Table 2, Section 2.

4. Experimental planning

This section describes the design of the experiment according to the guidelines of Wohlin et al. [45].

4.1. Goal

According to the Goal/Question/Metric template suggested by Moody [11], the research goal is:

Analyse *the outcome of a code-centric and a model-driven method for including gesture-based interaction into user interfaces,*
For the purpose of *carrying out a comparative evaluation*
With respect to *the usability of the model-driven method*
From the viewpoint of *researchers*
In the context of *researchers and practitioners interested in gesture-based interaction*

4.2. Research questions and hypothesis formulation

The goal of our study is to compare the usability of a method to deal with gesture-based interfaces through code-centric versus model-driven. Since usability is an abstract concept, we need to operationalise it through more measurable concepts. According to ISO 25062-2006 [13], usability can be measured through effectiveness, efficiency and satisfaction. Following the works of Moody [11], satisfaction can be measured using perceived usefulness, perceived ease of use and intention to use.

We consider two scenarios in the experiment, the first one is related to the inclusion of gesture-based interaction (the subject follows a set of tasks specified in the experiment to include gesture-based interaction in a user interface) and the second scenario is related to the definition of custom touch gesture (the subject employs a finger or a pen/stylus to sketch a gesture on a touch-based surface).

Regarding the type of gestures used in this experiment, even though gestUI supports multi-stroke gestures, we decided to use only single-stroke gestures because our goal in the study is based on CRUD operations (which are simple gestures).[5] The experiment is based on CRUD operation since they are the most frequently used in information systems [46,47]. This decision could result in a simplification of the experimental tasks but it does not reduce the validity of the experiment. We simplified the tasks in order to minimise the threat Boring, this way we can recruit more subjects. Usually, it is difficult to recruit subjects for long experiments and the percentage of abandonments is high in them. Note that we are comparing gestUI versus code-centric to solve the same problem. So, even using a simple experimental problem we are not benefiting any treatment, conditions are the same. The study of how the complexity of the problems can affect the results is out of scope of our experiment.

Therefore, in the evaluation of efficiency and effectiveness we consider research questions (RQ1, RQ2, RQ3 and RQ4) to measure usability within each scenario, since we are interested in evaluating the subjects when they are including gesture-based interaction in the user interface and when they are defining gestures. However, for the evaluation of satisfaction (PEOU, PU and ITU) we consider research questions (RQ5, RQ6 and RQ7) without differentiating between scenarios, since we are interested in the global value of the method (code-centric and gestUI) for usability.

Considering this perspective, the research questions and the null hypothesis (named as H_{0i}, with $i = [1..5]$ and corresponding with each research question) proposed for the experiment are:

RQ1: *Regarding the inclusion of gesture-based interaction in user interfaces, is there any difference between the effectiveness of the code-centric method and gestUI?* The null hypothesis tested to address this research questions is: H_{01}: *There is no difference between the effectiveness of gestUI and the code-centric method in the inclusion of gesture-based interaction in user interfaces.*

RQ2: *Concerning the definition of custom touch gestures, is there any difference between the effectiveness of the code-centric method and gestUI?* The null hypothesis tested to address this research questions is: H_{02}: *There is no difference between the effectiveness of gestUI and the code-centric method to specify custom gestures.*

RQ3: *Regarding the inclusion of gesture-based interaction in user interfaces, is there any significant difference between the efficiency of the code-centric method and gestUI?* The null hypothesis tested to address this research question is: H_{03}: *There is no difference between the efficiency of gestUI and the code-centric method in the inclusion of gesture-based interaction in user interfaces.*

RQ4: *Concerning the definition of custom touch gestures, is there any difference between the efficiency of the code-centric method and gestUI?* The null hypothesis tested to address this research question is: H_{04}: *When the subjects define gestures, efficiency is the same independently of the method used.*

RQ5: *How do subjects perceive the usefulness of gestUI in relation to the code-centric method?* The null hypothesis tested to address this research question is: H_{05}: *gestUI is perceived as easier to use than the code-centric method.*

RQ6: *How do subjects perceive the ease of use of gestUI in relation to the code-centric method?* The null hypothesis tested to address this research question is: H_{06}: *gestUI is perceived as more useful than the code-centric method.*

RQ7: *What is the intention to use of gestUI related to the code-centric method?* The null hypothesis tested to address this research question is: H_{07}: *gestUI has the same intention to use as the code-centric method.*

[5] Prior to the experiment, the gestures that represented the CRUD operations were defined by the authors of this work. However, a module was included (equivalent to an option in the menu of the information system used in the experiment) that allowed the redefinition of the gestures used in the experiment. This fact means that the gestures can be defined/redefined by the users of the information system. The main idea of this module is that if the gesture defined by the software engineers/designers are complicated to remember or difficult to trace, users can define their own gestures to perform the actions in the information system.

Table 3
Factor and treatments of the experiment.

Factor	Treatment		Description
	ID	Name	
Method to use	I	Code-centric method	Subjects manually write the source code to define custom gestures and to include gesture-based interaction in a user interface.
	II	gestUI	Subjects employ gestUI with the aim of defining custom gestures and including gesture-based interaction in a user interface.

Table 4
Response variables to evaluate effectiveness and efficiency of gestUI.

Response variables	Metrics	Definition	Research question
INCLUSION OF GESTURE-BASED INTERACTION			
Effectiveness in the inclusion of gesture-based interaction	Percentage of correct tasks carried out in the inclusion of gesture-based interaction (PTCCI).	This is the relationship between: the number of tasks correctly completed and the total number of tasks during the inclusion of gesture-based interaction in the user interface.	RQ1
Efficiency in the inclusion of gesture-based interaction	Time to finish the task during the inclusion of gesture-based interaction in the user interface (TFTI).	This is the number of minutes spent on each task. This is reported by the subjects during the inclusion of gesture-based interaction in the user interface.	RQ3
CUSTOM GESTURE DEFINITION			
Effectiveness in the custom gesture definition	Percentage of correct tasks carried out in the custom gesture definition (PTCCG).	This is the relationship between: the number of tasks carried out correctly and the total number of tasks during the definition of custom gestures.	RQ2
Efficiency in the custom gesture definition	Time to finish the task during the custom gesture definition (TFTG).	This is the number of minutes spent on the experimental task. This is reported by the subjects during the definition of custom gestures.	RQ4

4.3. Factor and treatments

Each software development characteristic to be studied that affects the response variable is called a factor [48] (a.k.a. “independent variable”). In this case, the factor detected in the experiment is the method to use and it has two treatments: the code-centric method and the model-driven method. Table 3 includes the description of the factor and its two treatments.

Eclipse Framework is used as a tool to operationalise the code-centric method. This tool is used to implement the source code in Java that represents a user interface. gestUI operationalises the model-driven method. gestUI is used to include gesture-based interaction in a user interface through conceptual models (without writing any lines of code) [49].

4.4. Response variables and metrics

Response variables are the effects studied in the experiment caused by the manipulation of factors. In this experiment, we evaluate gestUI with regard to: effectiveness, efficiency and satisfaction.

4.4.1. Response variables for effectiveness and efficiency

In this experiment, we are interested in the evaluation of the subjects when they define custom gestures using a finger (or a pen/stylus) on a touch-based surface, and we also are interested in the evaluation of the subjects using gestUI to include gesture-based interaction. Therefore, we need metrics to evaluate efficiency and effectiveness for each scenario.

In this experiment, in order to answer the research questions (RQ1, RQ2, RQ3 and RQ4), we define a metric per research question with the aim of evaluating the effectiveness and efficiency of gestUI when the subjects work in two scenarios: (i) they include gesture-based interaction in a user interface and (ii) they define custom gestures during the experiment. Table 4 shows the response variables classified per scenario and research question. The columns of Table 4 describe the response variables, their metrics, definition and the research question that they aim to answer.

In this work, the term “correct task” means that the user has performed the task of defining a gesture (or the inclusion of gesture-based interaction in the user interface) without errors.

4.4.2. Response variables for satisfaction

In this experiment, in order to answer research questions RQ5, RQ6 and RQ7, we define a metric for each one with the aim of measuring satisfaction through PEOU, PU and ITU. We use a 5-point Likert scale in order to measure ITU, PEOU and PU. In this case we are not distinguishing between defining custom gestures and including gesture-based interaction in a user interface during the experiment, rather we are measuring satisfaction of the whole process. Table 5 describes response variables, their metrics, definition and the research questions that we aim to answer.

Table 5
Responses variables to measure satisfaction of use gestUI.[a]

Response variable	Metrics	Definition	Research question
Satisfaction	Perceived ease of use (PEOU)	This is the arithmetic mean of the Likert scale values of MEM questionnaire items related with perceived ease of use.	RQ5
	Perceived usefulness (PU)	This is the arithmetic mean of the Likert scale values of MEM questionnaire items related with perceived usefulness.	RQ6
	Intention to use (ITU)	This is the arithmetic mean of the Likert scale values of MEM questionnaire items related with intention to use.	RQ7

[a] We are aware that Likert scales are qualitative data but some studies propose converting them to quantitative to work with statistical tests [60].

Table 6
Summary of RQ's, hypotheses, response variables and metrics.

Response variables	Metric	RQ	Hypotheses
Effectiveness in the inclusion of gesture-based interaction	PTCCI	RQ1	H_{01}
Effectiveness in the custom gesture definition	PTCCG	RQ2	H_{02}
Efficiency in the inclusion of gesture-based interaction	TFTI	RQ3	H_{03}
Efficiency in the custom gesture definition	TFTG	RQ4	H_{04}
Perceived ease of use	PEOU	RQ5	H_{05}
Perceived usefulness	PU	RQ6	H_{06}
Intention to use	ITU	RQ7	H_{07}

Table 6 shows a summary of the research questions, hypotheses, response variables and metrics used to test these hypotheses.

4.5. Experimental subjects

The experiment was conducted in the context of the Universitat Politècnica de València (Spain). We had 21 subjects (15 males and 6 females) who are six M.Sc. and fifteen Ph.D. students in Computer Science. The experiment is not part of a course and the students are encouraged to participate on a voluntary basis.

The background and experience of the subjects are found through a demographic questionnaire handed out at the first session of the experiment. This instrument consists of 15 questions on a 5-point Likert scale. According to the questions included in the demographic questionnaire, the results are:

- Mainly, the subjects are between 25–29 (33%) and 30–34 years (24%).
- Regarding the computing platform, two of the most used are: Microsoft Windows (52% of the subjects) and MacOS (33%). Linux is used by 15% of the subjects.
- All subjects (100%) indicated that they had taken a Java programming course. 62% of the participants had taken a model-driven development (MDD) course and 52% of the subjects had taken a human–computer interaction (HCI) course.
- Regarding the software development experience using Eclipse IDE and Java, 43% of the subjects reported that they have "Average" self-rated programming expertise on a 5-point Likert scale. In this scale, the number 3 means "Intermediate" and the number 5 means "Expert".
- Furthermore, the subjects reported their experience in model-driven development. The "Average" self-rated model-driven development expertise was 33% on a 5-point Likert scale. In this scale, the number 3 means "Intermediate" and the number 5 means "Expert". Also, in this field, 29% have a "Poor" level and 14% have a "Very Poor" level.
- Regarding experience using gestures on a device/computer, 71% of the subjects occasionally use gestures in their daily activities. Additionally, 43% of the subjects would like to define custom gestures to use them in their daily activities.

Table 7 summarises the information about the subjects extracted from the demographic questionnaire. We conclude that subjects have some experience in the context of software development related with this experiment, but they do not have experience in the definition of custom gestures and the inclusion of gesture-based interaction in user interfaces.

4.6. Experiment design

In this experiment, we use a *crossover design* [45] (a.k.a. a paired comparison design). This is a type of design where each subject applies both methods, that is, the subjects use one method (the code-centric method) and then they use a second method (gestUI, a model-driven method) or vice versa. The order of use of each method depends on which group the subject was assigned to at the beginning of the experiment in such a way that each treatment is balanced among all the subjects. This design has the advantages that we are using the largest sample size to analyse the data, hence we avoid the learning effect and the problem is not confounded with the treatments.

Table 7
Summary of demographic questionnaire.

	Value	%
Average age		
20–24 years	3	14
25–29 years	7	33
30–34 years	5	24
35–39 years	4	19
>40 years	2	10
Gender		
Male	15	71
Female	6	29
Computing platform		
Microsoft Windows	11	52
MacOS	7	33
Linux	3	15
Courses taken		
Java	21	100
HCI	11	52
MDD	13	62
Software development experience		
Average experience	9	43
Poor experience	7	33
Very Poor experience	5	24
Experience using gestures	15	71
No experience using gesture	6	29
Model-driven development experience		
Average experience	7	33
Poor experience	6	29
Very Poor experience	3	14

Table 8
Crossover design.

ID	Treatment	Subjects	
I	Code-centric method	G1	G2
II	Model-driven method (gestUI)	G2	G1

Table 9
Operators and average time on KLM.

Operator	Description	Average time	Observations
M	Mental Operation	1.2 s	mentally prepare
H	Home	0.4 s	Home in on keyboard or mouse (change of device)
P	Point	1.1 s	point with mouse
K	Keystroke	0.28 s	keystroke or mouse button press
R(t)	System responsive	t s	Waiting for the system to become responsive (t)

With the aim of comparing both methods against each other, each subject uses both methods (treatments) on the same object; to minimise the effect of the order in which subjects apply the methods, we balanced the treatment applied in the first term. As Table 8 shows, the experiment is carried out with the subjects separated into two groups (G1 and G2). Each group is composed of subjects that are assigned according to a random value obtained by means of a random numbers calculator available on the Internet (https://www.random.org/). Therefore, the 21 subjects were randomly split into two groups following a process known as counterbalancing: (a) 11 subjects first apply gestUI and then the code-centric method, whilst (b) the other 10 subjects start with the code-centric method and then apply gestUI.

The expected time to fulfil the tasks defined in each treatment was around two hours. This value was estimated based on two factors: (i) a previous pilot test and (ii) using the KLM method (Keystroke Level Method) [50,51]. KLM is a model for predicting the time that an expert user needs to perform a given task on a given computer system. KLM is based on counting keystrokes and other low-level operations, including the user's mental preparations and the system's responses [51]. Using this model, we estimate the time required to input the lines of code required in the code-centric method considering the operators and their average time proposed in [52] and shown in Table 9.

Table 10
Estimating time for the experiment.

Treatment	Previous pilot test	By using KLM
Code centric method	1 h 08 min.	0 h 57 min.
Model-driven method	0 h 24 min.	0 h 21 min.
Total time	**1 h 32 min.**	**1 h 18 min.**

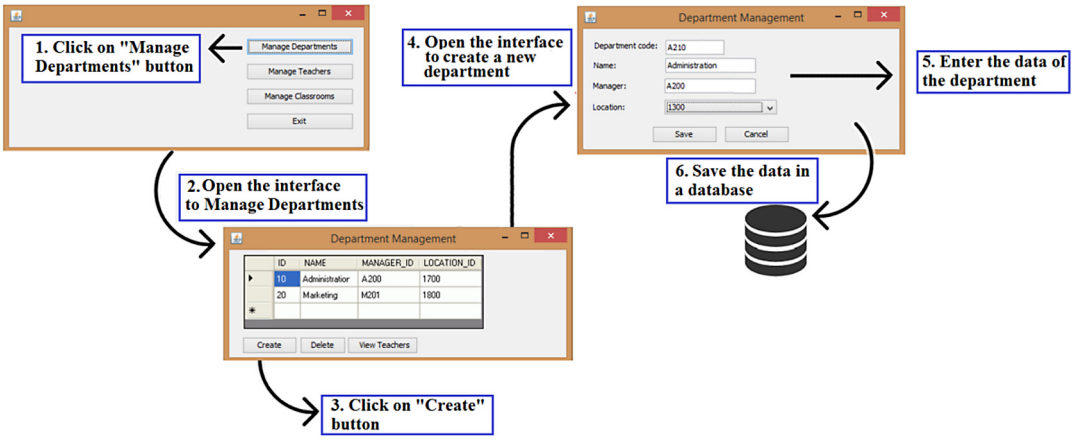

Fig. 4. Software system supporting traditional interaction.

The values of K operator is defined according to type of user: expert typist, average skilled typist, average non-secretarial typist, worst typist [50]. In this experiment, we consider the average time of a non-secretarial typist because the subjects participating in the experiment had to type in the source code that was included in the respective instrument of each method, as is explained in the following pages. R(t) operator (t indicates the time in seconds that the user has to wait) defines the time when the computer is busy doing some processing, and the user must wait before they can interact with the system. The estimated values of time to perform the experiment are shown in Table 10.

4.7. Experimental objects

The object used in the experimental investigation is a requirements specification created for this purpose. It contains the description of a problem related with the definition of custom gestures and the inclusion of gesture-based interaction in user interfaces of a software system supporting traditional interaction using a mouse and keyboard. Fig. 4 shows this software system containing a main user interface to manage information of departments, teachers and classrooms in a university by means of CRUD operations. Each option opens a new interface to specify information required by the university.

Even though gestUI has been used to include gestural interaction in another type of software (see [43,42]), we decided to perform the experiment with a simple information system since the subjects are familiar with the use of GUI interfaces with windows and buttons to perform actions. That is, an information system to focus the analysis on answering the research questions and avoiding functional complications.

Using traditional interaction, when the subjects click on the 'Manage Departments' button a new interface is opened, which contains the information of each previously defined department in a grid included in the user interface. Next, clicking on the 'Create' button, a new interface is opened to enter information concerning a new department. Finally, when the information is complete, the 'Save' button saves the information in a database.

The user must perform the same CRUD operations but using custom gestures, that is, by means of gesture-based interaction. In this case, our work is related with the use of indirect symbolic gestures for triggering a command. The gesture is related with a previously defined action containing the command to execute. Implicitly, such action contains parameters to include in the command to execute. In general, the gesture can be single-stroke or multi-stroke, but in this experiment we use single-stroke gestures because we use gestures to specify CRUD operations by means of simple drawings (letters C, R, U and D).

From the point of view of achieving the objective of the experiment reported in this article, that is, to compare the actions carried out with the code-centric method and with gestUI, in our opinion, the use of single-stroke gestures in the experiment does not have serious consequences. Basically, the imposed restrictions in the type of gestures depend on where the gestures are applied to decide the use of single-stroke or multi-stroke gestures. In this case, single-stroke gestures are used because they are suitable and enough for plotting the letters that represent the gestures of the CRUD operations. In another case, a multi-stroke gesture could be a better solution for the purpose of drawing a gesture that allows executing an action already established in a software (for example, using multi-stroke gestures in a CASE tool to draw primitives from a

Table 11
Instruments defined for the experiment.

Instrument	Description
Demographic Questionnaire	Questionnaire to assess the subjects' knowledge and experience of the technologies and concepts used in the experiment. This document includes questions containing Likert-scale values ranging from 1 (strongly disagree) to 7 (strongly agree).
Task Description Document for the code-centric method	Document that describes the tasks to be performed in the experiment using the code-centric method and containing empty spaces to be filled in by the subjects with the start and end times of each step of the experiment. This document contains guidelines to guide the subject throughout the experiment and the source code to be included in the user interface.
Task Description Document for the model-driven method (gestUI)	Document that describes the tasks to be performed in the experiment using the model-driven method and containing empty spaces to be filled in by the subjects with start and end times of each step of the experiment. This document contains guidelines to guide the subject throughout the experiment.
Post-test Questionnaire for the code-centric method	Questionnaire with 16 questions containing Likert-scale values ranging from 1 (strongly disagree) to 7 (strongly agree) to evaluate satisfaction of the whole process when the subjects use the code-centric method to define custom gestures and to include gesture-based interaction.
Post-test Questionnaire for the model-driven method (gestUI)	Questionnaire with 16 questions containing Likert-scale values ranging from 1 (strongly disagree) to 7 (strongly agree) to evaluate satisfaction of the whole process when the subjects use the model-driven method (gestUI) to define custom gestures and to include gesture-based interaction.

diagram, or a multi-stroke gesture to execute a complex action that includes the command to execute and some parameter or option of the command).

Therefore, if gesture-based interaction is included in user interfaces, the subjects can sketch gestures on the touch-based display of the computer in order to execute some actions (in this case, the CRUD operations). One gesture can contain the definition of one or more actions, but the gesture–action correspondence must be unique per interface. Gestures are defined during the specification of the gesture-based interaction in each user interface. In this case, the 'D' gesture contains two actions (each one in a different interface): (i) it can be used to open the user interface to manage departments, and (ii) it can be used to delete one previously selected record in the database.

After the user draws the gesture, $N is used for recognition. This process is online and, if the gesture is recognised, the previously specified action is triggered. This specification was made at the time the gesture–action correspondence was defined (see Section 3.2, in the description of the Subsystem "Gesture–Action Correspondence Definition Module"). In this case, an action is related with a command containing the instruction to execute each CRUD operation, for example, the gesture "S" executes the command to save the information included in the fields of the user interface in the database of the information system.

Even though the problem is small, it contains the necessary elements to validate the method: (i) a gesture catalogue definition containing the aforementioned six gestures, and (ii) the process to include the gesture-based interaction in the existing user interface source code. The inclusion of a greater number of user interfaces or gestures in the catalogue during the experiment would mean repetitive work for the subjects.

4.8. Instrumentation

All the material required to support the experiment was developed beforehand, including the preparation of the experimental object, instruments and task description documents for data collection used during the execution of the experiment. The instruments used in the experiment are described in Table 11.

4.9. Experiment procedure

This section describes the procedure used to conduct the experiment. Prior to the experiment session, a pilot test was run with one subject who finished the Master's degree in Software Engineering in the Universitat Politècnica de València. This pilot study helped us to improve the understandability of some instruments.

In this experiment, we consider a user interface of the existing software system mentioned in Section 4.7. In this user interface, users perform CRUD operations to manage information by means of a traditional interaction with a mouse and a keyboard. We are interested in including gesture-based interaction in the user interfaces of a software system. So, the experiment addresses a real problem, i.e. the definition of custom gestures and the inclusion of gesture-based interaction in an existing user interface to perform the aforementioned operations.

Prior to the experiment and considering that gestUI is a model-driven method, the definition of the gesture catalogue starts with the platform independent definition (that is, the PIM of the gestures), then through the use of model transformations the specific model of gestures is obtained (that is, the PSM of the gestures). The detail of this process is not included in this article, but it can be reviewed in [5]. On the other hand, we define the gesture catalogue that the subjects require to apply both treatments in the experiment in coordination with some representative users of this type of information systems. The gesture catalogue (see Table 12) consists of four gestures to execute each CRUD operations action and one additional gesture to save the information in the database ('S' gesture). Observe that the 'D' gesture has two actions to

Table 12
Gesture catalogue defined in the experiment.

Action	Gesture	Description	User interface
Open the "Managing Department" user interface		The user sketches this gesture to open the user interface to manage departments in the university.	Main user interface
Create a new department		The user sketches this gesture to open the user interface to create a new department.	Managing departments
Read a department record		The user sketches this gesture to open the user interface to read the previously selected record of a department.	Managing departments
Update the information of the existing department		The user sketches this gesture to open the user interface to update the previously selected record of a department.	Managing departments
Delete a record of a department		The user sketches this gesture to open the user interface to delete the previously selected record of a department.	Managing departments
Save the information of a department		The user sketches this gesture to save the information of a department in the database.	Department Information

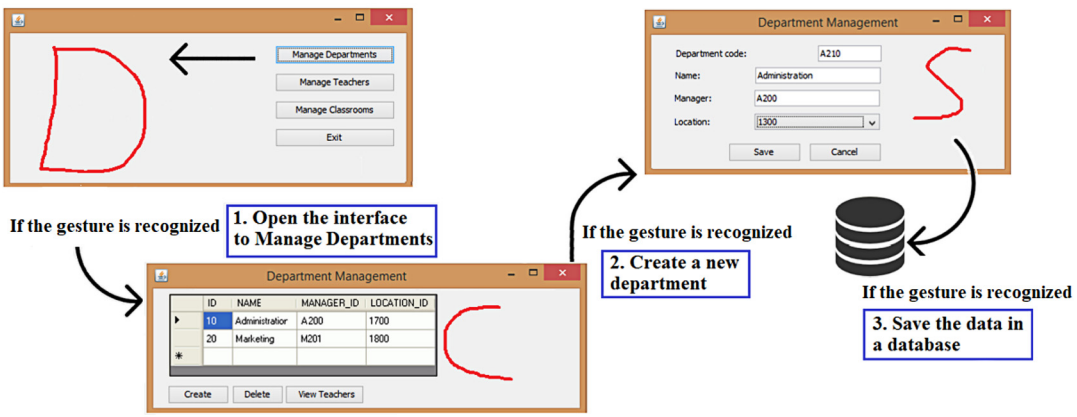

Fig. 5. Software system supporting gesture-based interaction.

execute depending on the user interface where the gesture is sketched by the user. This gesture catalogue is included in the Task Description Document of each treatment.

Hence, the user interface must contain the definition of gestures to perform CRUD operations. For instance, Fig. 5 shows three gestures defined in the user interface: (i) 'D', to open the user interface to manage departments; (ii) 'C', to create a new department, by opening the user interface to enter the information of a new department; (iii) 'S', to save the information in the database.

In this experiment, each user interface of the information system used in the experiment has an additional area where the gestures are drawn exclusively there, without worrying about "invading" the rest of the user interfaces on the screen of the computer. This decision was taken since the gesture might start on some control of other user interface and an unexpected action might occur before finishing the gesture.

We consider two versions of the "Task Description Document", as explained in Table 11. We use a sub-index 'c' when naming the task ID to express the treatment "Code-centric method" and we use a sub-index 'g' to express this treatment gestUI when naming the task ID. The subjects apply both treatments designed in the experiment with the aim of managing the input of gestures sketched by the users to execute actions in the software system. Task Description Documents were delivered to the subjects before starting the experiment.

Fig. 5 shows the execution of the actions related to the set of gestures defined to perform the experiment described in this paper. The following events occur when the user sketches the gestures:

- The user sketches a gesture ("D") in order to open a user interface to manage departments.
- If the gesture is recognised, the "Manage Departments" user interface is opened. Next, the user sketches the gesture ("C") with the aim of opening a user interface to create departments.
- If the gesture is recognised, the "Create a department" user interface is opened. When the user finishes to enter the information of the department, he/she sketches other gesture ("S") in order to save the information in a database.
- Finally, if the gesture is recognised, the information is saved in a database.

Table 13
An excerpt of the Task Description Document containing the sequence of steps for custom gesture definition using the code-centric method.

No.	Task ID	Task description	Observations
1	$TG1_C$	Definition of gesture "C"	The subject sketches the "C" gesture using a finger or a pen/stylus
2	$TG2_C$	Definition of gesture "R"	The subject sketches the "R" gesture using a finger or a pen/stylus
3	$TG3_C$	Definition of gesture "U"	The subject sketches the "U" gesture using a finger or a pen/stylus
4	$TG4_C$	Definition of gesture "D"	The subject sketches the "D" gesture using a finger or a pen/stylus
5	$TG5_C$	Definition of gesture "S"	The subject sketches the "S" gesture using a finger or a pen/stylus
6	$TG6_C$	Save gesture catalogue	The subject saves the gesture catalogue

The steps in the procedure of the experiment are:

Step 1: The goal of the experiment was introduced to the subjects and guidelines on how to conduct the process were given to them.

Step 2: Each subject filled in a Demographic Questionnaire before starting the experiment where the subjects were asked about age, gender, courses taken, experience in software development, experience in model-driven development, and experience using gestures (Table 7). Results of this questionnaire are described in Section 4.5.

Step 3: The subjects did the experiment divided into two groups (G1 and G2) following the instructions given in the Task Description Document of each method. In this experiment, for each method, we separately evaluate two processes: (i) custom gesture definition and (ii) inclusion of gesture-based interaction, since we are interested in evaluating effectiveness and efficiency of the subjects when they specify gestures on a touch-based device and when they include gesture-based interaction. The evaluation of effectiveness and efficiency, taking in account PTCCG, PTCCI, TFTI, and TFTG (see Section 4.4) is performed based on the information registered in the Task Description Document.

During the inclusion of gesture-based interaction in the user interface we use the next process: When the subject finished the previously defined tasks to include the gesture-based interaction (Table 14 includes an excerpt of the corresponding Task Description Document), we analyse the questionnaire filled by the subject regarding performed tasks: number of tasks correctly completed and the number of incomplete tasks. Then, we obtain the percentage of tasks correctly completed in relation with the total number of tasks. Similar process is followed to calculate PTCCG.

Next, we evaluate each method (code-centric and gestUI) in a global way with regard to PEOU, PU and ITU. The sequence of steps for each group is the following.

– G1 group. G1 subjects applied the code-centric method to complete Treatment I.

Treatment I (*code-centric method*). In this case, the subjects received the Task Description Document containing instructions to apply the code-centric method with the aim of adding new source code to **define custom gestures**. Following the instructions included in the Task Description Document, the subjects perform a sequence of steps (see Table 13 that contains an excerpt of the Task Description Document) to define the catalogue of gestures described in Table 12. The definition of a gesture using the code-centric method consists of the creation of an XML file whose structure, in this case, is based on the gesture specification according to $N gesture recogniser [7].

Even though the subjects received an Eclipse project containing existing source code of the user interface, they must write new lines of source code included in the Task Description Document using the editor of the Eclipse IDE in the existing source code in order to add functionalities related to gesture-based interaction. The source code included in the instrument called "Task Description Document" does not represent the only solution code for the purpose of including interaction based on gestures in the user interface, however, it is a solution that allows to conduct the experiment.

The decision to include source code was taken with the aim of reducing the duration time of the experiment considering that if the subjects had written all the source code to define gestures and to include gesture-based interaction in the existing user interface from scratch, probably they would have required a greater number of hours (or maybe days). We think that this decision could have some influence on the result of the experiment depending on the subject's experience in software development and the time required to write source code in the Eclipse Framework IDE.

An excerpt of the sequence of steps to perform in the experiment to **include gesture-based interaction** using the code-centric method is included in Table 14.

Tasks $TI1_C$, $TI2_C$ and $TI3_C$ allow the adaptation of the source code of $N gesture recogniser in the source code of the user interface with the aim of adding a gesture recogniser in the software system to recognise the gestures sketched by the users. $TI4_C$ includes a panel in the user interface where the gestures are sketched by using a finger or pen/stylus. $TI5_C$ and $TI6_C$ permit the inclusion of listeners to sense the finger that is sketching a gesture. These listeners capture the information produced on the user interface when a gesture is sketched. $TI7_C$ and $TI8_C$ manage the process to draw the gesture on the user interface. $TI9_C$ implements a method to define the gesture–action correspondence. In this case, the subject needs to execute a process to search actions included in the source code. We use a user interface where the actions are related with buttons definition (e.g. 'Manage Departments', 'Create', 'Save'). Subjects define the action–gesture relationship using the specification of gestures described in Table 12.

Table 14
An excerpt of the Task Description Document containing the sequence of steps for gesture-based interaction inclusion using the code-centric method.

Task ID	Task description
$TI1_C$	To include $N as gesture recogniser in the software system.
$TI2_C$	To implement methods and attributes required to use $N as gesture recognition.
$TI3_C$	To implement the method to read gestures sketched by the user.
$TI4_C$	To add a new panel in the user interface to draw gestures.
$TI5_C$	To write a method to implement a listener sensing the finger (or pen/stylus) that is drawing a gesture.
$TI6_C$	To write a method to implement a listener sensing that the gesture definition is complete.
$TI7_C$	To implement a method to manage graphics in Java.
$TI8_C$	To implement a method to paint a gesture on the user interface.
$TI9_C$	To implement a method containing the gesture–action correspondence.
$TI10_C$	To compile the new version of the source code and to run the software system.

Table 15
An excerpt of the Task Description Document for custom gesture definition using gestUI.

Task ID	Task description
$TG1_G$	Definition of gesture “C”
$TG2_G$	Definition of gesture “R”
$TG3_G$	Definition of gesture “U”
$TG4_G$	Definition of gesture “D”
$TG5_G$	Definition of gesture “S”
$TG6_G$	Executing model-transformation to obtain a platform-independent gesture catalogue

Table 16
Gesture–action correspondence step-by-step definition.

No.	Description	Explanation
1	It selects a gesture from the gesture catalogue	This contains the gesture selected by the subject.
2	It selects an action from the list of actions included in the user interface	This contains the actions selected by the subject.
3	It contains the gesture–action correspondence definition	The subject confirms the gesture–action correspondence.
4	It generates the new version of the source code of the user interface	This contains the process to generate the source code of the user interface containing gesture-based interaction.

As a final result, the subjects obtain a new version of source code containing gesture-based interaction in the user interface in order to execute actions indicated in the requirements specification using gestures. Then, in $TI10_C$, the subjects must compile the source code of the software system in Eclipse IDE, and then they can execute the software system in order to test the gestures defined in the process to execute the previously specified actions in the experiment.

– G2 group. G2 subjects employed gestUI to complete Treatment II.

Treatment II (*gestUI*). In this procedure, we consider the same user interfaces of the software system shown in Fig. 4. G2 subjects received the Task Description Document containing instructions to apply gestUI to define custom gestures and to include gesture-based interaction in the user interface. This treatment consists of the definition of the gesture catalogue, and the specification of data to apply model transformations in order to generate the source code of the user interface containing the gesture-based interaction.

Firstly, the subjects define the gesture catalogue by means of a pen/stylus or a finger on a touch-based surface. These gestures are stored in a repository, as described in Section 3.2, and then the platform-independent gesture catalogue (gesture-catalogue model) is obtained. The tasks to perform this step are included in Table 15, which shows an excerpt of the Task Description Document for this treatment.

Secondly, with the aim of obtaining the platform-specific gesture specification, subjects apply a model-to-model transformation that requires as input the gesture catalogue model.

Thirdly, the subject selects the user interface and the platform-specific gesture specification to design the gesture-based interaction by defining the gesture–action correspondence. This correspondence is defined with the aim of assigning each gesture to an action. Fig. 6 shows the interface of the tool that contains the process to define this correspondence consisting of steps 1 to 4 shown in red.

Finally, gestUI generates the code with a new version of the user interfaces including gesture-based interaction. Then, the subjects use Eclipse IDE to compile the source code of the software system and afterwards they test the gestures defined in the process.

Table 16 contains the description of the steps shown in Fig. 6.

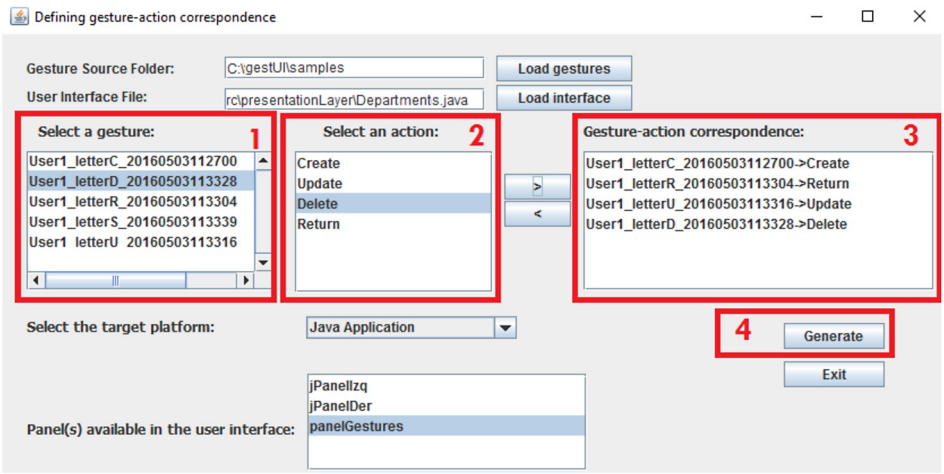

Fig. 6. Gesture–action correspondence definition using tool support. (For interpretation of the colours in the figure(s), the reader is referred to the web version of this article.)

At the end of this process, the result is the generated source code of the user interface of the software system supporting gesture-based interaction to execute actions, according to the definition of gesture–action correspondence. Fig. 5 shows the same software system described in Fig. 4 but supporting gesture-based interaction.

Step 4. Subjects filled in the corresponding Post-Test Questionnaire according to the treatment employed in the experiment.

According to Table 8, in Section 4.6, after the G1 subjects employed the code-centric method they must employ gestUI to complete Treatment II, repeating steps 1 to 3 again. In similar way, after the G2 subjects employed the gestUI method they must employ the code-centric method to complete Treatment I.

The data to evaluate PEOU, PU and ITU in this experiment were obtained from the post-task and post-test questionnaires. After the data was gathered, they were checked for correctness and the subjects were consulted when necessary. The data obtained of the aforementioned questionnaires filled in by the subjects are used to measure the response variables defined in Section 4.4.

4.10. Threats of validity

In this section we discuss the most important threats to the validity of this evaluation. We have classified the threats according to Wohlin et al. [45], each of which is discussed below.

Internal validity: The main threats to the internal validity of the experiment are:

(i) *Subject's experience in defining gesture-based interaction*: this threat was resolved since none of the subjects had any experience in tasks related to the topic of custom gesture definition included in the experiment, according to the pre-test questionnaire. So, the subjects' experience in both treatments is the same.

(ii) *Subject's experience in software development*: there are some factors that can influence the experiment:
   a. Some of the subjects could have more experience than others in the development of software. Although we used the pre-test questionnaire in order to find out their experience in this field, this threat could not be resolved since we designed the groups in a random way. This threat could affect the evaluation of the effectiveness and the efficiency because the time required to perform the experiment depends on the experience level of the subjects.
   b. In some cases, subjects without an adequate level of experience in managing source code could produce syntax errors in the source code when inserting the additional source code. This threat could be resolved, since the subjects received adequate information and printed source code without errors included in the Task Description Document with the aim of obtaining a new version of the existing source code of the user interface.

(iii) *Information exchange among subjects*: this threat was resolved since the experiment was developed in one session, and it was difficult for the subjects to exchange information with each other.

(iv) *Learning effect*: this threat could not be resolved in both treatments (described in Section 4.7.1 and 4.7.2, respectively) since the process to define custom gestures is identical to the five gestures included in the experiment. Therefore, the definition of the first gesture required more time and effort compared to the following gestures. This threat could affect the evaluation of efficiency and effectiveness because the time needed to perform the experiment depends on the experience level of the subjects.

(v) *Fatigue*: The experiment may suffer this threat since the gestures used with gestUI and with code-centric are the same, which may be boring for the subjects.

External validity: The main threats to the external validity of the experiment are:

i. *Duration of the experiment*: there are some factors that can influence the duration of the experiment.
   a. Since the duration of the experiment was limited to 2 hours, only one interface, six actions (CRUD operations + save the information + open the interface to manage departments) and five gestures were selected. However, repetitive tasks could permit a reduction of time since the subject already knows the process to perform. This threat could not be resolved since these tasks, even though repetitive, were necessary to build the system.
   b. Subjects require time to analyse the structure and the logic of the existing source code before the inclusion of the additional source code. This threat could be resolved by including adequate instructions in the Task Description Document in order to perform the experiment.
   c. If any subject requires the maximum amount of time to perform the experiment, which is 2 hours (according to what is specified in Section 4.6), the information is considered not valid to process because this situation can represent some of the following situations: (i) the subject writes source code slowly using the keyboard and mouse, (ii) a subject does not have the same experience in the use of software tools for software development in relation to other subjects and he/she requires more time to complete the experiment probably performing additional tasks (e.g. checking if the source code was completely transcribed from the Task Description Document to the Eclipse project, checking for syntax errors in the source code).
   d. Total time required to perform the experiment depends on of the typing speed and the experience of the subject in managing source code. This threat could not be resolved in Treatment I (it contains more lines of code to write than Treatment II) since we do not check each subject's typing ability on the computer.
   e. Time required to check whether the inclusion of the gesture-based interaction was successful varies depending on the experience of the subjects. This threat could be resolved since the subjects answered a question in the pre-test questionnaire about experience in the use of an IDE to develop software in a positive way (43% have an "average" self-rated expertise and 38% have an "experienced" self-rated experience).

ii. *Representativeness of the results*: despite the fact that the experiment was performed in an academic context, the results could be representative with regard to novice evaluators with no experience in evaluations related with the gesture interaction definition and inclusion. With respect to the use of students as experimental subjects, several authors suggest that the results can be generalised to industrial practitioners [53,54].

Construct validity: The main threat to the construct validity of the experiment is:

(i) *Type of measurements to consider in the experiment*: measurement that are commonly employed in this type of experiment were used in the quantitative analysis. The reliability of the questionnaire was tested by applying the Cronbach test, the obtained value is higher than the acceptable minimum (0.70).

Conclusion validity: The main threats to the conclusion validity of the experiment are:

(i) *Validity of the statistical tests applied*: this was resolved by applying Wilcoxon Signed-rank test, one of the most common tests used in the empirical software engineering field. According to Wohlin et al. [45] if we have a sample whose size is less than 30 and we have a factor with two treatments, we can use non-parametric statistical tests such as the Wilcoxon Signed-rank test. In Section 4.11 the non-parametric tests used in this experiment are detailed.
(ii) *Low statistical power:* this happens when the sample size is not large enough. The power of any statistical test is defined as the probability of rejecting a false null hypothesis. According to G*Power [55] the sample size needed for an effect size of 0.8 is 20 subjects, which is the number of subjects we have. So, this threat has been minimised.

4.11. Data analysis

The calculated values are checked to see the p-value (significance level). An important issue is the choice of significance level which specifies the probability of the result being representative. Generally speaking, the practice dictates rejecting the null hypothesis when the significance level is less than or equal to 0.05 [48].

The first step is to analyse the reliability of the data obtained in the experiment: we start by calculating the *Cronbach coefficient* (alpha). In this case, the result obtained is 0.736. According to [56] if the Cronbach coefficient is greater or equal to 0.7 then the reliability of the data is assumed.

Boone et al. [57] recommend some data analysis procedures for Likert scale data: (a) for central tendency: *mean*, (b) for variability: *standard deviation*, (c) for associations: *Pearson's r*, and (d) other statistics using: *ANOVA*, *t-test*, *regression*. According to Juristo et al. [48], if we have a sample whose size is less than 30 and it follows a normal distribution, then we employ t-distribution (Student's), but if the sample does not follow a normal distribution then we can apply the Wilcoxon Signed-rank test in order to analyse the data obtained in the experiment. A normality test using the Shapiro–Wilk test is required in order to verify if the data is normally distributed. We use this test as our numerical means of assessing normality because it is more appropriate for small sample sizes ($<$50 samples). Then, using Shapiro–Wilk we obtained the result that the data is not normally distributed. In this case, we cannot apply the t-distribution test because this test requires normally distributed data. So, we apply the Wilcoxon Signed-rank test.

Table 17
Non-parametric Levene's test for the variables in the experiment.

Variable	*F*	df1	df2	Sig.
PEOUg	0.353	1	19	0.560
PEOUc	0.004	1	19	0.948
Pug	0.042	1	19	0.840
PUc	0.754	1	19	0.396
ITUg	0.147	1	19	0.706
ITUc	0.416	1	19	0.527

Table 18
Descriptive statistics for metrics.

	N	Min.	Max.	Mean	Std. Dev.
PTCCIg	21	50	100	82.1429	17.9284
PTCCIc	21	50	100	77.3810	15.6220
PTCCGg	21	75	100	91.6667	12.0762
PTCCGc	21	25	100	71.4286	19.8206
TFTIg	21	9.00	33.00	19.7143	7.0224
TFTIc	21	18.00	49.00	28.3810	7.8834
TFTGg	21	12.75	66.75	31.8929	16.8301
TFTGc	21	60.50	346.25	154.6786	66.5967
PEOUg	21	1	5	3.2857	0.2154
PEOUc	21	1	5	3.3280	0.5073
PUg	21	1	5	3.8176	0.3451
PUc	21	1	5	3.2786	0.5762
ITUg	21	2	5	3.7381	0.7179
ITUc	21	1	4	2.9286	0.6761
Valid *N*	21				

The next step is verifying whether the data satisfy the sphericity condition and whether they are homogeneous:

- In order to check the *sphericity condition, Mauchly's test* can be used. However, in this work, there are only two levels of repeated measures (with the gestUI method and with a code-centric method), which precludes a sphericity violation and the test is unnecessary.
- *Non-parametric Levene's test* is used to test if the samples have homogeneity in their variances. In the result of this test we can observe in column "Sig." in Table 17, that the non-parametric Levene's test for homogeneity of variances provides a p-value > 0.05, allowing us to assume that the data have homogeneity in their variances.

In the section 5, we report the quantitative results of the experiment based on the statistical analysis of the data using (i) descriptive statistics (mainly arithmetic mean), (ii) box-and-whisker plot, (iii) Spearman's Rho correlation coefficient to study the correlation between both treatments, and (iv) the Wilcoxon Signed-rank test with the aim of addressing the research questions. The results of applying Wilcoxon Signed-rank test are described grouped by variables (PTCCI, PTCCG, TFTI, TFTG, PU, PEOU and ITU).

Additionally, at the end of the Section 5, we include the results of the effect size calculation in order to check the meaningfulness of the results and allow comparison between studies.

A significance level of 0.05 was established to statistically test the obtained results with subjects in the experiment. The analysis has been performed using the SPSS v.23 statistical tool.

5. Results

In this section, the subscript 'g' located at the end of each variable means "using the gestUI method", and the subscript 'c' means "using the code-centric method". Next, we analyse the results for each research question.

5.1. RQ1: effectiveness in the inclusion of gesture-based interaction

We consider two treatments to analyse PTCCI in the inclusion of gesture-based interaction: PTCCIg and PTCCIc.

According to Table 18, the mean of PTCCIg (82.14%) is greater than the mean of PTCCIc (77.38%), that is, the subjects achieved a greater percentage of correctly carried out tasks using gestUI than when they employed the code-centric method.

Fig. 7 presents the box-and-whisker plot containing the distribution of the PTCCI variable per method. The medians of PTCCIg and PTCCIc are similar, but the third quartile is better for PTCCIg, since the percentage of correctly carried out tasks achieved by the subjects using gestUI is greater than the percentage achieved when the subjects use the code-centric method. This means that gestUI is slightly more effective than the code-centric method when the subjects include gesture-based interaction in user interfaces.

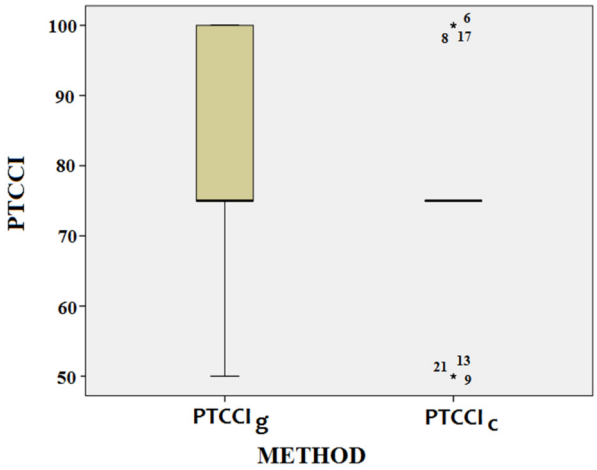

Fig. 7. Box-and-whisker plot of PTCCI.

Table 19
Wilcoxon Signed-rank test for PTCCI.

		N	Mean rank	Sum of ranks
PTCCIc–PTCCIg	Negative ranks	6[a]	4.50	27.00
	Positive ranks	2[b]	4.50	9.00
	Ties	13[c]		
	Total	21		

[a] PTCCIc < PTCCIg.
[b] PTCCIc > PTCCIg.
[c] PTCCIc = PTCCIg.

Using Spearman's Rho correlation coefficient, we obtained a positive correlation (0.638). So, we can conclude that PTCCIg and PTCCIc are strongly correlated, that is, when the percentage of correctly carried out tasks using gestUI increases, the percentage using the code-centric method also increases.

In order to check whether the observed differences were significant we ran the Wilcoxon Signed-rank test. We obtained the results shown in Table 19. They show that two subjects (2/21) have obtained a greater number of correctly carried out tasks using the code-centric method compared to gestUI to include gesture-based interaction in the experiment. Six subjects (6/21) have obtained a greater number of correctly carried out tasks using gestUI compared to the code-centric method. However, thirteen subjects (13/21) have obtained the same number of correctly carried out tasks for both methods.

The results obtained with this test are: 2-tailed p-value $= 0.157 > 0.05$ and $Z = -1.414$, therefore, according to this result, we cannot reject the null hypothesis and can conclude that "*There is no difference between the effectiveness of the gestUI and the code-centric methods in the inclusion of gesture-based interaction in user interfaces*".

5.2. RQ2: effectiveness in the definition of custom gestures

We consider two treatments to analyse PTCCG in the custom gesture definition: PTCCGg and PTCCGc.

According to Table 18, the mean of PTCCGc (71.43%) is less than the mean of PTCCg (91.67%), that is, the subjects achieved a relatively greater percentage of correctly carried out tasks using gestUI than when they employed the code-centric method.

Fig. 8 presents the box-and-whisker plot containing the distribution of the PTCCG variable per method. The median, the first quartile and the third quartile are better for PTCCGg, since it achieved a greater percentage of correctly carried out tasks. This means that gestUI was more effective than the code-centric method when the subjects define custom gestures.

Using Spearman's Rho correlation coefficient, we obtained a positive correlation (0.456). Then, we can conclude that PTCCGg and PTCCGc have a moderate correlation, that is, when the percentage of correctly carried out tasks with PTCCGg increases, there is a moderate increment in the percentage of PTCCGc.

In order to check whether the observed differences were significant we ran the Wilcoxon Signed-rank test. We obtained the results shown in Table 20.

It shows that fourteen subjects (14/21) have obtained more correctly carried out tasks using gestUI compared to using the code-centric method, zero (0/21) subjects have obtained more correctly carried out tasks using the code-centric method than using gestUI, and there are seven (7/21) subjects that have obtained the same percentage using both methods.

The results obtained with this test are: 2-tailed p-value $= 0.000 < 0.05$ and $Z = -3.556$, therefore, according to this result, we reject the null hypothesis and can conclude that "*gestUI is more effective than the code-centric method in the definition of custom gestures*".

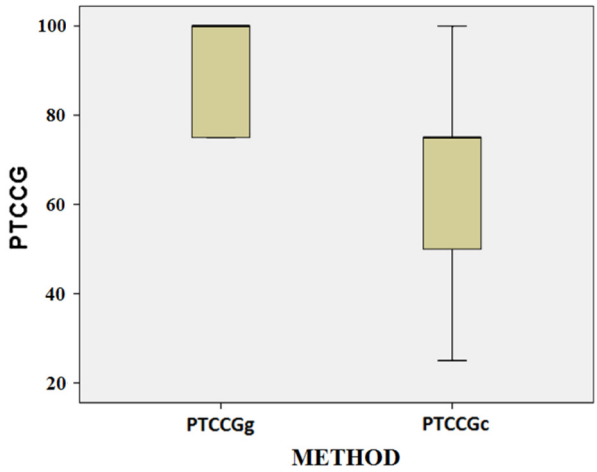

Fig. 8. Box-plot-whisker of PTCCG.

Table 20
Wilcoxon signed-rank test for PTCCG.

		N	Mean rank	Sum of ranks
PTCCGc–PTCCGg	Negative ranks	14[a]	7.50	105.00
	Positive ranks	0[b]	.00	.00
	Ties	7[c]		
	Total	21		

[a] PTCCGc < PTCCGg.
[b] PTCCGc > PTCCGg.
[c] PTCCGc = PTCCGg.

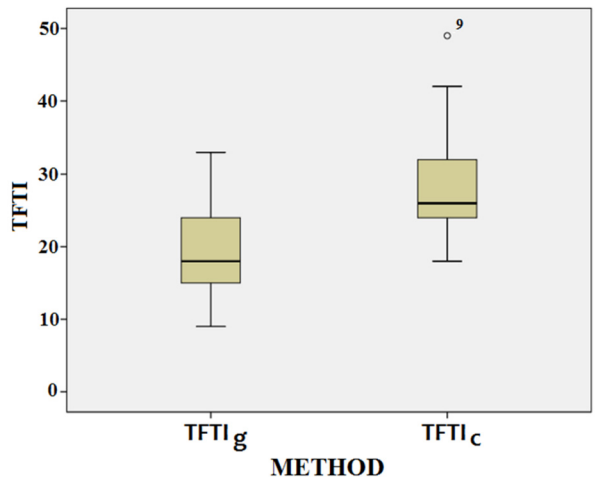

Fig. 9. Box-plot for TFTI.

5.3. RQ3: efficiency in the inclusion of gesture-based interaction

We consider two treatments to analyse TFTI in the inclusion of gesture-based interaction: TFTIg and TFTIc.

According to Table 18, the mean of TFTIc (28.38) is greater than that of TFTIg (19.71), that is, the time required to include gesture-based interaction in the experiment using the code-centric method is greater than the time needed to perform this task using gestUI.

Fig. 9 presents the box-and-whisker plot containing the distribution of the TFTI variable per method. The medians, first quartile and third quartile are better for TFTIg, since the time needed to conduct the experiment is less when the subjects use gestUI rather than when the subjects use the code-centric method. This means that the time to finish the task with gestUI is better than with code-centric.

Table 21
Wilcoxon Signed-rank test for TFTI.

		N	Mean rank	Sum of ranks
TFTIc–TFTIg	Negative ranks	3[a]	7.17	21.50
	Positive ranks	18[b]	11.64	209.50
	Ties	0[c]		
	Total	21		

[a] TFTIc < TFTIg.
[b] TFTIc > TFTIg.
[c] TFTIc = TFTIg.

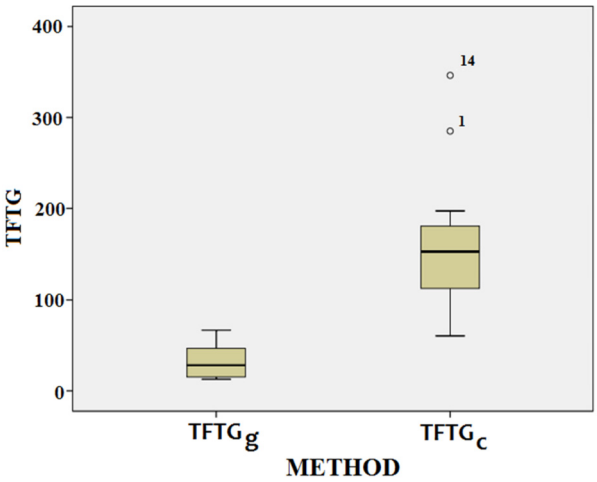

Fig. 10. Box-plot of TFTG.

Using Spearman's Rho correlation coefficient, we obtained a positive correlation (0.210). Then, we can conclude that TFTIg and TFTIc have a weak correlation, that is, between TFTIg and TFTIc there is not a significant relationship (Sig. (2-tailed) > 0.05) in the process of including gesture-based interaction.

In order to check whether the observed differences were significant we ran the Wilcoxon Signed-rank test. We obtained the results shown in Table 21. They show that eighteen subjects (18/21) have employed more time using the code-centric method compared to gestUI to include gesture-based interaction in the experiment. Three subjects (3/21) have employed less time using the code-centric method than gestUI to include gesture-based interaction in the experiment.

The values obtained with this test are: 2-tailed p-value $= 0.001 < 0.05$ and $Z = -3.269$, therefore, according to this result, we reject the null hypothesis and we can conclude that "*gestUI is more efficient than the code-centric method in the inclusion of gesture-based interaction in user interfaces*".

5.4. RQ4: efficiency in the definition of custom gestures

We consider two treatments to analyse TFTG in the definition of custom gestures: TFTGg and TFTGc.

According to Table 18, the mean of TFTGc (154.67) is greater than the mean of TFTGg (31.89), which means that the time required to define custom gestures in the experiment using the code-centric method is greater than the time to do this task using gestUI.

Fig. 10 presents the box-and-whisker plot containing the distribution of the TFTG variable per method. The median, first quartile and third quartile are better for TFTGg, since TFTGg needs less time to complete the task. This means that gestUI was more efficient than code-centric method regarding the time required by the subject to define custom gestures during the experiment.

Using Spearman's Rho correlation coefficient, we obtained a positive correlation (0.216). Then, we can conclude that TFTGg and TFTGc have a weak correlation, that is, when the time required to define custom gestures using code-centric method increases, the time using gestUI method also has a weak increment.

In order to check whether the observed differences were significant, we run Wilcoxon Signed-rank test. We obtain the results shown in Table 22.

It shows that twenty-one subjects (21/21) have employed more time using the code-centric method than gestUI to define custom gestures in the experiment.

Table 22
Wilcoxon Signed-rank test for TFTG.

		N	Mean rank	Sum of ranks
TFTGc–TFTGg	Negative ranks	0[a]	.00	.00
	Positive ranks	21[b]	11.00	231.00
	Ties	0[c]		
	Total	21		

[a] TFTGc < TFTGg.
[b] TFTGc > TFTGg.
[c] TFTGc = TFTGg.

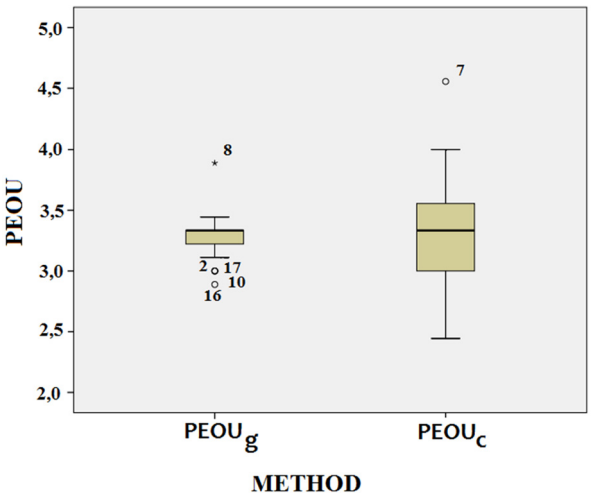

Fig. 11. Box-plot for PEOU.

The values obtained with this test are: 2-tailed p-value $= 0.000 < 0.05$ and $Z = -4.015$, therefore, according to this result, we reject the null hypothesis and we can conclude that "*When the subjects define gestures, gestUI is more efficient than the code-centric method*".

5.5. RQ5: perceived ease of use

We consider two treatments to analyse PEOU: PEOUg and PEOUc.

Table 18 presents the results obtained through questions related to PEOU within Post-task and Post-test questionnaires. In this case, the mean is above 3.0 in both cases. There is a difference of 0.042 between the mean of PEOUc and the mean of PEOUg, that is, the PEOU of gestUI is relatively greater than the PEOU of the code-centric method.

Fig. 11 shows the box-and-whisker plot containing the distribution of the PEOU variable per method. The medians of both treatments are the same. The first quartile is slightly better for gestUI and the third quartile is slightly better for the code-centric method. This means that there are no differences between both treatments.

Using Spearman's Rho correlation coefficient, we obtain a positive correlation (0.408). So, we can conclude PEOUg and PEOUc have a moderate correlation, that is, when the perceived ease of use with gestUI increases, PEOU using the code-centric method also increases.

In order to check whether the observed differences were significant, we ran the Wilcoxon Signed-rank obtaining the results shown in Table 23. They show that eight subjects (8/21) perceive that gestUI is easier to use than the code-centric method, eight subjects (8/21) perceive that the code-centric method is easier to use than gestUI and, five (5/21) perceive that both methods are easy to use.

The values obtained with this test are: 2-tailed p-value $= 0.917 > 0.05$ and $Z = -0.104$, therefore, according to this result, we cannot reject the null hypothesis and we can conclude that "*gestUI is perceived as easier to use than the code-centric method*".

5.6. RQ6: perceived usefulness

We consider two treatments to analyse perceived usefulness: PUg and PUc.

Table 18 presents the results obtained through questions related to PU in Post-task and Post-test questionnaires. In this case, the mean of PUc is less than PUg, that is, perceived usefulness of gestUI (mean = 3.82) is greater than the perceived usefulness of the code-centric method (mean = 3.28).

Table 23
Wilcoxon Signed-rank test for PEOU.

		N	Mean rank	Sum of ranks
PEOUc–PEOUg	Negative ranks	8[a]	8.25	66.00
	Positive ranks	8[b]	8.75	70.00
	Ties	5[c]		
	Total	21		

[a] PEOUc < PEOUg.
[b] PEOUc > PEOUg.
[c] PEOUc = PEOUg.

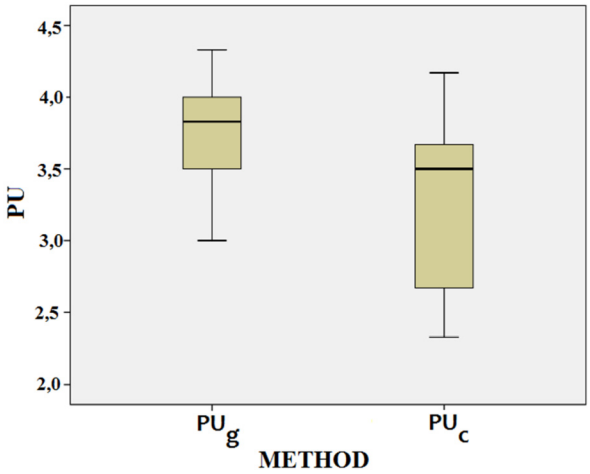

Fig. 12. Box-plot of PU.

Table 24
Wilcoxon Signed-rank test for PU.

		N	Mean rank	Sum of ranks
PUc–PUg	Negative ranks	15[a]	10.63	159.50
	Positive ranks	3[b]	3.83	11.50
	Ties	3[c]		
	Total	21		

[a] PUc < PUg.
[b] PUc > PUg.
[c] PUc = PUg.

Fig. 12 presents the box-and-whisker plot containing the distribution of the PU variable per method. The median, first quartile and third quartile of PUg is better than PUc. This means that the subjects perceived gestUI to be more useful than the code-centric method.

Using Spearman's Rho correlation coefficient, we obtain a positive correlation (0.310). So, we can conclude that PUg and PUc have a weak correlation, that is, when the perceived usefulness of the code-centric method increases, the perceived usefulness using the gestUI method also increases.

In order to check whether the observed differences were significant, we ran the Wilcoxon Signed-rank obtaining the results shown in Table 24. This test shows that fifteen subjects (15/21) perceive gestUI to be more useful than the code-centric method in the experiment. Three subjects (3/21) perceive the code-centric method to be more useful than gestUI, and three (3/21) consider that both methods have the same level of perceived usefulness in the experiment.

The values obtained with this test are: 2-tailed p-value $= 0.001 < 0.05$ and $Z = -3.239$, therefore, according to this result, we reject the null hypothesis and we can conclude that "*gestUI is perceived as more useful than the code-centric method*".

5.7. RQ7: intention to use

We consider two treatments to analyse ITU: ITUg and ITUc.

Table 18 presents the results obtained through questions related to ITU in Post-test and Post-task questionnaires. In this case, the mean of ITUg (3.74) is above 3.0 while the mean of ITUc (2.93) is below to 3.0.

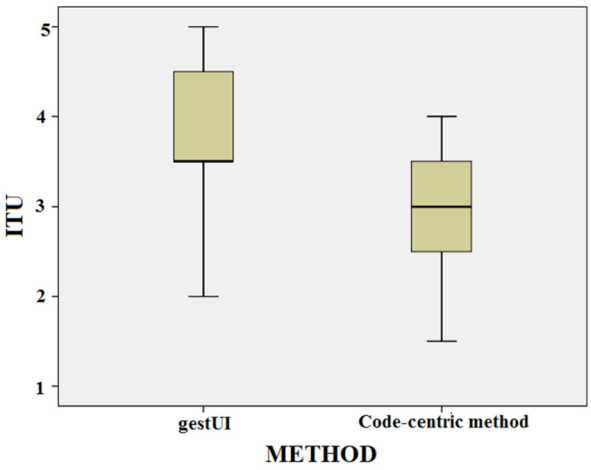

Fig. 13. Box-plot of ITU.

Table 25
Wilcoxon Signed-rank test for ITU.

		N	Mean rank	Sum of ranks
ITUc–ITUg	Negative ranks	13[a]	8.65	112.50
	Positive ranks	2[b]	3.75	7.50
	Ties	6[c]		
	Total	21		

[a] ITUc < ITUg.
[b] ITUc > ITUg.
[c] ITUc = ITUg.

Table 26
Summary of the results obtained in the experiment.

Variable	Null hypothesis status	Conclusion
PTCCI	Not rejected	There is no significant difference between the effectiveness of gestUI and the code-centric method in the inclusion of gesture-based interaction in user interfaces.
PTCCG	Rejected	There is a significant difference between the effectiveness of gestUI and the code-centric method in the specification of custom gestures. Results obtained are better when the subjects use gestUI rather than the code-centric method, that is, PTCCGg is greater than PTCCGc.
TFTI	Rejected	There is a significant difference between the efficiency of gestUI and the code-centric method in the inclusion of gesture-based interaction in user interfaces. The results obtained are less when the subjects use gestUI rather than when they use the code-centric method.
TFTG	Rejected	When the subjects define gestures, gestUI is more efficient than the code-centric method.
PEOU	Not rejected	gestUI is perceived as easier to use than the code-centric method.
PU	Rejected	gestUI is perceived as more useful than the code-centric method.
ITU	Rejected	gestUI has an intention to use greater than the code-centric method

Fig. 13 presents the box-and-whisker plot containing the distribution of the ITU variable per method. The median, the first and third quartile are better for ITUg. This means that gestUI has a greater intention to use than the code-centric method when the subjects use it to define custom gestures and to include gesture-based interaction.

Using Spearman's Rho correlation coefficient, we obtain a positive correlation (0.080). So, we can conclude that ITUg and ITUc have a very weak correlation, that is, when the intention to use of gestUI (ITUg) increases, the intention to use of the code-centric method (ITUc) increases very little compared with ITUg.

In order to check whether the observed differences were significant, we ran the Wilcoxon Signed-rank obtaining the results included in Table 25. They show that gestUI has greater intention to use than the code-centric method (13/21 subjects), the code-centric method has two (2/21) subjects with intention to use, and six (6/21) subjects have an intention to use for both methods.

The values obtained with this test are: 2-tailed p-value $= 0.003 < 0.05$ and $Z = -3.005$, therefore, according to this result, we reject the null hypothesis, and we can conclude that "*gestUI has an intention to use greater than the code-centric method*".

In summary, the result of each hypothesis is shown in Table 26.

Table 27
Effect size of the metrics.

Response variable	Metric	Mean	St. Dev.	Cohen's *d*	Equivalence
Effectiveness in the inclusion of gesture-based interaction.	PTCCI				
	PTCCIg	82.1429	17.9284	0.2832	Medium
	PTCCIc	77.3810	15.6220		
Effectiveness in the custom gesture definition.	PTCCG				
	PTCCGg	91.667	12.076	1.233	Large
	PTCCGc	71.428	19.821		
Efficiency in the inclusion of gesture-based interaction.	TFTI				
	TFTIg	19.714	7.022	1.161	Large
	TFTIc	28.381	7.883		
Efficiency in the custom gesture definition.	TFTG				
	TFTGg	31.893	154.678	2.5279	Large
	TFTGc	16.8301	66.5967		
Satisfaction	PU				
	PUg	3.8176	0.3451	1.1349	Large
	PUc	3.2786	0.5762		
	PEOU				
	PEOUg	3.2857	0.2154	0.1085	Small
	PEOUc	3.3280	0.5073		
	ITU				
	ITUg	3.7381	0.7179	1.1609	Large
	ITUc	2.9286	0.6761		

5.8. Effect-size calculation

According to Kotrlik [58], effect size measures focus on the meaningfulness of the results and allow comparison between studies, furthering the ability of researchers to judge the practical significance of results presented. We use means and standard deviations of the metrics defined in this experiment to calculate Cohen's d and effect-size correlation r. The calculation was performed using the effect size calculator provided by the University of Colorado (Colorado Springs), available at http://www.uccs.edu/~lbecker/.

Based on the work of Lakens [59], we can see that the effect size is "Large" if $d > 0.8$, "Medium" if $d <= 0.5$ and $d > 0.2$, and "Small" if $d < 0.2$. In Table 27, we present the results of the effect size calculation of the metrics included in this experiment and this shows the equivalences applied to the results obtained.

According to this classification, the results obtained for effect size show that:

(i) In the case of PTCCG, TFTI, TFTG, PU and ITU, the effect size calculated through Cohen's d is greater than 0.8, which means that it is classified as "Large". So, there is a significant difference in the application of each method in this experiment related to: effectiveness in the definition of custom gestures (PTCCG), efficiency in the inclusion of gesture-based interaction (TFTI), efficiency in the definition of custom gesture (TFTG), perceived usefulness (PU) and intention to use (ITU).

(ii) In the case of PTCCI, the effect size calculated through Cohen's d is equal to 0.2832 ($d > 0.2$), which is classified as "Medium". So, the difference in the application of each method to include gesture-based interaction in a user interface considering the effectiveness in the inclusion of gesture-based interaction, is not important.

(iii) In the case of PEOU, the effect size calculated through Cohen's d is less than 0.2 ($d = 0.1085$), which is classified as "Small". So, there is a minimum difference in the application of each method in this experiment related to the perceived ease of use (PEOU).

In the next section, we analyse the results obtained in this experiment.

6. Discussion

In this section, we discuss the results of the experiment described in Section 5 in order to draw some conclusions regarding the comparison of gestUI (a model-driven method) and the code-centric method (traditional software development). In order to validate gestUI, three aspects are considered in this experiment: effectiveness (using PTCCI, PTCCG), efficiency (using TFTI and TFTG) and satisfaction (using PU, PEOU and ITU). The discussion about the results obtained in the experiment is performed according to the aforementioned research questions.

6.1. Effectiveness

RQ1: Effectiveness in the inclusion of gesture-based interaction

Regarding PTCCI metric, which is related with RQ1, the results show that there is no significant difference between the results obtained when the subjects applied gestUI and when the subjects applied the code-centric method to include gesture-based interaction in an existing user interface. We consider that the small difference obtained (approximately 4%) by applying both methods to calculate PTCCI is because (i) the subjects used existing source code (included in the Task Description Document) instead of writing the source code from scratch as is done in a typical development process [39]. This context helped to obtain better results with the code-centric method and the difference was less than expected; (ii) the subjects were not familiar with the process defined in gestUI to apply a model-driven method (i.e. by using model transformations to include gesture-based interaction); (iii) the subjects did not have experience in the inclusion of gesture-based interaction and when they applied gestUI, the process was not very intuitive to follow. However, the process implemented in gestUI is like a typical wizard included in some available applications in any operating system. Therefore, if the user is not familiar with the process of gestUI, this feature (wizard) helped to the users to perform the inclusion of gesture-based interaction with minor number of problems regarding to code-centric method.

RQ2: Effectiveness in the definition of custom gestures

About PTCCG metric, which is related with RQ2, values obtained show that gestUI is significantly more effective than the code-centric method in the definition of custom gestures. The percentage obtained with gestUI is greater than the percentage obtained with the code-centric method. In this case, the difference between the percentage of task correctly carried out in the custom gestures definition using gestUI or using the code-centric method is almost 20%. This difference is due to subjects using gestUI having a more intuitive process to follow to define gestures and to obtain a XML file containing the description of the gesture. In gestUI we added a canvas to draw gestures, in a similar way as $N. Therefore, the subjects can perform this task easier than with other method. By other side, using the code-centric method, the process of defining gestures is more complex because it includes additional tasks (e.g. analyse the shape of the gesture, draw it and define it using XML, among others) requiring more effort. In both methods, the subjects started defining a gesture whose definition process was new for them, but in the other gestures, the process was similar which meant that they required less effort to define the rest of gestures in the same method.

6.2. Efficiency

RQ3: Efficiency in the inclusion of gesture-based interaction

Concerning the TFTI metric, related with RQ3, values obtained show that gestUI is significantly more efficient than the code-centric method in the inclusion of gesture-based interaction in user interfaces. When the subjects did the experiment using gestUI, they required less time than when they used the code-centric method. The difference of time between both methods is moderate (8.67 min.) in the inclusion of gesture-based interaction in user interfaces, this could be related to the ability to type the source code in a correct way, probably because the subjects had experienced developing software (according to the demographic questionnaire, the average self-rated programming expertise was 43%). Also, they required less time to type source code since they had experience using the integrated development environment used in the experiment (according to the demographic questionnaire 38% had an "experienced" level and 43% had a "medium experienced" level with Eclipse Framework). Probably, the aforementioned wizard available in gestUI to perform the process of inclusion of gesture-based interaction helped to the subject to be more efficient completing the process specified in the Task Description Document. By other side, the null experience of the subjects in the use of gestUI involved a higher time in the use of gestUI. Other results of the use of gestUI is that we identified difficulty to apply the model transformations included in gestUI.

RQ4: Efficiency in the definition of custom gestures

Regarding TFTG metric, related with RQ4, obtained results show that the time required to define custom gestures using gestUI is less than the time required using the code-centric method. The difference of the time required to define custom gestures, by means of each method, is high (122.7857) since some subjects had some problems with the definition of gestures using XML language as they were not familiar with the syntax of XML. Another aspect that could have increased the time required with the code-centric method is related to syntax errors generated during the process of gesture definition. If the subjects run the experiment first with gestUI and then with the code-centric method, they require a longer time than those subjects that run the experiment first with the code-centric method and then with gestUI. In this case, there were some problems when the subjects employed $N to recognise some gestures sketched by them. This could have had some influence in the duration of the process of custom gesture definition.

In summary, regarding effectiveness and efficiency, we can say that:

- The result obtained in the experiment permit one to say, in general, that the effectiveness and efficiency of gestUI are greater than those of the code-centric method.
- Considering the metrics PTCCG, TFTG and TFTI, the results obtained with Cohen's d value ($d > 0.8$, i.e. "Large") suggest a high practical significance for the results obtained. Also, Cohen's d value ($d = 0.2832$ for PTCCI) suggested a moderate practical significance for the results obtained.

- Concerning the values of TFTG and TFTI obtained in the experiment, we think that if the subjects had written the source code from scratch, the difference in time would have been greater. In general, the overall results lead us to interpret that gestUI has achieved better effectiveness and efficiency for the subjects in almost all the analysed statistics in comparison with the code-centric method.
- Finally, considering effect size, we can conclude that in comparison, effectiveness and efficiency of gestUI are better than those obtained with the code-centric method in the custom gesture definition.

6.3. Satisfaction

RQ5: Perceived ease of use

With respect to PEOU, related with RQ5, obtained results show that the difference between PEOUg (3.286) and PEOUc (3.328) is minimal (0.0423). So, we can say that the subjects perceive that both methods are easy to use. However, in the case of the code-centric method, this result could be influenced by the inclusion of source code in the Task Description Document as was explained in Section 4.7.1. This decision was taken with the aim of reducing the complexity of the code-centric method and the time required to do the experiment. Other factor that may affect the result can be the experience of the subjects in the use of the IDE (Eclipse) used in the experiment to write the source code required in the code-centric method. The subjects perceived as ease of use the code-centric method because they are familiar with the process of writing code in an IDE to obtain a solution of software.

RQ6: Perceived usefulness

Regarding PU, which is related with RQ6, obtained results show that there is difference (0.539) between the values of PUg (3.8176) and PUc (3.2786). So, we can say that the subjects perceive gestUI to be more useful than the code-centric method. The subjects perceive the usefulness of gestUI by noting that if gestUI is easy to use they may find gestUI more useful, and hence, have some motivation to use it. Specifically, the subjects perceive the usefulness of gestUI when they use it to automatically obtain source code to include gesture-based interaction in a user interface based on a specification of gestures and actions to define the gesture-based interaction.

RQ7: Intention to use

About ITU, related with RQ7, obtained results show that there is a difference (0.8095) between the values of ITUg (3.7381) and ITUc (2.9286). So, we can say that the subjects have an intention to use gestUI greater than the code-centric method. This conclusion is based on the fact that the subjects considered gestUI as easy to use and useful compared to the code-centric method.

In general, the results of our work indicate that gestUI is accepted by the subjects since the results obtained for effectiveness, efficiency and satisfaction with gestUI are better that the results obtained with the code-centric method. With these results we could say that gestUI is a hopeful approach and justifies further investigation.

7. Conclusions and future work

This paper compares a model-driven method (gestUI) versus a traditional software development method (the code-centric method) in terms of (i) effectiveness in the custom gesture definition, (ii) effectiveness in the inclusion of gesture-based interaction, (iii) efficiency in the custom gesture definition, (iv) efficiency in the inclusion of gesture-based interaction, and satisfaction (PEOU, PU and ITU) through an experimental investigation. Results show that, in general, gestUI has a greater effectiveness, efficiency and satisfaction level than the code-centric method, and gestUI was also perceived by the subjects as easier to use than the code-centric method. It is important to highlight that these differences between gesUI and a code-centric method arise even when we work with simple experimental problems, as we use in our experiment.

Some aspects that must be contextualised according to the type of experiment are:

(i) The sample size is small, twenty-one (21) subjects.
(ii) The subjects were M.Sc. and Ph.D. students and they do not have enough experience in the topics included in the experiment: tasks related with the custom gesture definition and the inclusion of gesture-based interaction.
(iii) The subjects have experience in software development using the Java programming language, which could have influenced the results obtained with the code-centric method.
(iv) We consider that the decision to include source code in the Task Description Document to reduce the time for the code-centric method has reduced the differences in terms of efficiency between treatments, since subjects only had to transcribe the source code specified in the document.

Gesture definition is interesting for the subjects since they can specify their own gestures with the aim of executing actions in a user interface. In this context, each subject defined four gestures in order to use them in the user interface doing CRUD operations in a database. The subjects could define their own gestures according to their preferences.

Tailoring mechanism included in gestUI is very interesting because it permits that the users redefine some gesture hard to remember or draw. If the users experience problems when using gestures, they can solve this situation by themselves using this mechanism to redefine custom gestures without the support of a software engineer.

gestUI and its features give to the users the potential needed to define custom gestures and to include gesture-based interaction in user interfaces whose source code is available. It is sufficient with apply gestUI in order to incorporate gestures in a user interface. gestUI helps to improve the level of desirability of the software system as the user employs custom gestures that he/she has defined.

Even though the experimental results are good for the usefulness of gestUI, we are aware that more experimentation is needed to confirm these results. Existing results must be interpreted within the context of this experiment. In general, the subjects considered gestUI a good solution since they defined custom gestures and they included the gestures in the user interface in a short time compared to the time required when they used the code-centric method.

As future work, we plan to perform more replications of the experiment in order to minimise the influence of the threats to validity identified. Additionally, we consider some aspects that could be included in future work such as: (i) studying how to adapt gestUI to an existing model-driven framework to implement user interfaces because its architecture means it can be adapted to an existing framework based on the model-driven paradigm; (ii) the evaluation of gestUI with end-users who do not play the role of developers, since gestUI aims to be easy to use and any user could define gestures and include them in a user interface; (iii) different replications will allow us to build a family of experiments where data could be aggregated through meta-analysis or pooling data. This way we can improve the statistical power of the analysis; (iv) to include additional platforms (e.g. mobile platform) as a target to produce gesture-based user interfaces; currently we support desktop-computing; (v) to include additional programming languages (e.g. Visual Studio .NET with C#) as target language to generate source code, currently we support Java, in order to give support for gesture-based interaction to other types of software systems.

Further details about this validation can be found at https://gestui.wordpress.com/evaluation.

Acknowledgements

This work has been supported by Department of Computer Science of the Universidad de Cuenca and SENESCYT of Ecuador, and received financial support from the Generalitat Valenciana under “Project IDEO (PROMETEOII/2014/039)” and the Spanish Ministry of Science and Innovation through the “DataMe Project (TIN2016-80811-P)”.

References

[1] M. Hesenius, T. Griebe, S. Gries, V. Gruhn, Automating UI tests for mobile applications with formal gesture descriptions, in: Proc. of 16th Conf. on Human–Computer Interaction with Mobile Devices Services, 2014, pp. 213–222.
[2] S.H. Khandkar, S.M. Sohan, J. Sillito, F. Maurer, Tool support for testing complex multi-touch gestures, in: ACM International Conference on Interactive Tabletops and Surfaces, ITS'10, NY, USA, 2010.
[3] D. Schmidt, Guest editor's introduction: model-driven engineering, IEEE Comput. 39 (2) (2006) 25–31.
[4] H. Lü, Y. Li, Gesture coder: a tool for programming multi-touch gestures by demonstration, in: Proceedings of the 2012 ACM on Human Factors in Computing Systems, 2012, pp. 2875–2884.
[5] O. Parra, S. España, O. Pastor, A model-driven method and a tool for developing gesture-based information system interfaces, in: Proceedings of the CAiSE 2015 Forum at the 27th International Conference on Advanced Information Systems Engineering, CAiSE 2015, Stockholm, Sweden, 2015.
[6] J. Wobbrock, A. Wilson, Y. Li, Gestures without libraries, toolkits or training: a $1 recognizer for user interface prototypes, in: Proceedings of ACM Symposium on User Interface Software and Technology, UIST 2007, Newport, Rhode Island, USA, 2007.
[7] L. Anthony, J.O. Wobbrock, A lightweight multistroke recognizer for user interface prototypes, in: Proc. of Graphics Interface, 2010, pp. 245–252.
[8] R. Vatavu, L. Anthony, J. Wobbrock, Gestures as point clouds: a $P recognizer for user interface prototypes, in: Proceedings of the 14th ACM International Conference on Multimodal Interaction, ICMI'12, Santa Monica, California, USA, 2012.
[9] R.-D. Vatavu, L. Anthony, J. Wobbrock, $Q: a super-quick, articulation-invariant stroke-gesture recognizer for low-resource devices, in: Proceedings of the 20th International Conference on Human–Computer Interaction with Mobile Devices and Services, MobileHCI '18, Barcelona, Spain, 2018, 23, 12 pages.
[10] J. Vanderdonckt, P. Roselli, J.L. Perez-Medina, !FTL, an articulation-invariant stroke gesture recognizer with controllable position, scale, and rotation invariances, in: Proceedings of the 20th ACM International Conference on Multimodal Interaction, ICMI'18, Boulder, CO, USA, 2018.
[11] D. Moody, The method evaluation model: a theoretical model for validating information systems design methods, in: ECIS 2003 Proceedings, Naples, Italy, 2003.
[12] ISO/IEC, Ergonomics of Human–System Interaction, ISO, 2010. [Online]. Available: https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-1:v1:en. (Accessed 28 September 2015).
[13] ISO/IEC/JTC 1/SC 7, ISO/IEC 25062:2006, Software engineering – Software product Quality Requirements and Evaluation (SQuaRE) – Common Industry Format (CIF) for usability test reports, ISO, Geneva, 2006.
[14] T. Kapteijns, S. Jansen, S. Brinkkemper, H. Houët, R. Barendse, A comparative case study of model driven development vs traditional development: the tortoise or the hare, in: 4th European Workshop on “From Code Centric to Model Centric Software Engineering: Practices, Implications and ROI”, Enschede, The Netherlands, 2009.
[15] C. Bunse, H. Gross, C. Peper, Embedded system construction – evaluation of model-driven and component-based development approaches, in: M.R. Chaudron (Ed.), Models in Software Engineering: Workshops and Symposia at MODELS 2008, Springer, Heidelberg, 2009, pp. 66–77.
[16] S. Kane, J. Wobbrock, R. Ladner, Usable gestures for blind people: understanding preference and performance, in: CHI 2011: Session: Gestures, Vancouver, Canada, 2011.
[17] F. Ricca, M. Torchiano, M. Leotta, A. Tiso, G. Guerrini, G. Reggio, On the Impact of State-based Model-Driven Development on Maintainability: A Family of Experiments using UniMod,", Journal of Empirical Software Engineering (EMSE) 23 (3) (2018) 1743–1790, https://doi.org/10.1007/s10664-017-9563-8.
[18] P.E. Papotti, A.F. do Prado, W. Lopes de Souza, C.E. Cirilo, L. Ferreira Pires, A quantitative analysis of model-driven code generation through software experimentation, in: Proceedings of 25th International Conference, CAiSE 2013, in: LNCS, vol. 7908, 2013, pp. 321–337.
[19] N. Condori-Fernandez, J.I. Panach, A.I. Baars, T. Vos, O. Pastor, An empirical approach for evaluating the usability of model-driven tools, Sci. Comput. Program. 78 (2013) 2245–2258.

[20] Y. Martinez, C. Cachero, S. Melia, MDD vs traditional software development: a practitioner subjective perspective, Inf. Softw. Technol. 55 (2) (2013) 189–200.
[21] Y. Martinez, C. Cachero, S. Meliá, Evaluating the impact of a model-driven web engineering approach on the productivity and the satisfaction of software development teams, in: Proceedings of 12th International Conference Web Engineering, ICWE 2012, in: LNCS, vol. 7387, 2012, pp. 223–237.
[22] Y. Martinez, C. Cachero, S. Meliá, Empirical study on the maintainability of Web applications: model-driven engineering vs code-centric, Empir. Softw. Eng. 19 (2014) 1887–1920.
[23] M. Cervera, M. Albert, V. Torres, V. Pelechano, On the usefulness and ease of use of a model-driven method engineering approach, Inf. Softw. Technol. 50 (2015) 36–50.
[24] J.I. Panach, S. España, O. Dieste, O. Pastor, N. Juristo, In search of evidence for model-driven development claims: an experiment on quality, effort, productivity and satisfaction, Inf. Softw. Eng. 62 (C) (2015) 164–186.
[25] S. Safdar, M. Iqbal, M. Khan, Empirical evaluation of UML modeling tools—a controlled experiment, in: Modelling Foundations and Applications, ECMFA 2015, L'Aquila, Italy, in: Lecture Notes in Computer Science, vol. 9153, 2015.
[26] R. Neto, P. Adeodato, A. Salgado, A framework for data transformation in Credit Behavioral Scoring applications based on Model Driven Development, Expert Syst. Appl. 72 (2017) 293–305.
[27] F. Santos, I. Nunes, A. Bazzan, Model-driven agent-based simulation development: a modeling language and empirical evaluation in the adaptive traffic signal control domain, Simul. Model. Pract. Theory 83 (2018) 162–187.
[28] B. Hamid, D. Weber, Engineering secure systems: models, patterns, and empirical validation, Comput. Secur. 77 (2018) 315–348.
[29] A. Oliveira, V. Bischoff, L. Gonçales, K. Farias, M. Segalotto, BRCode: an interpretive model-driven engineering approach for enterprise applications, Comput. Ind. 96 (2018) 86–97.
[30] Microsoft, TouchMe Gesture Studio, Appsolutely Apps, 2015. [Online]. Available: https://www.microsoft.com/en-us/p/touchme-gesture-studio/9wzdncrdg0l8. (Accessed 2 August 2018).
[31] ASUS, Smart Gesture – Introduction of ASUS Smart Gesture Software, 2018. [Online]. Available: https://www.asus.com/support/FAQ/1009613/. (Accessed 2 August 2018).
[32] A. Forward, T. Lethbridge, Problems and opportunities for model-centric versus code-centric software development: a survey of software professionals, in: Proceedings of the 2008 International Workshop on Models in Software Engineering, MiSE'08, Leipzig, Germany, 2008.
[33] A. Kleppe, J. Warmer, W. Bast, MDA Explained: The Model Driven Architecture: Practice and Promise, Addison–Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.
[34] L. Vogel, Eclipse Rich Client Platform. The Complete Guide to Eclipse Application Development, Lars Vogel, 2015.
[35] A.C. Long, J.A. Landay, L.A. Rowe, quill: A Gesture Design Tool for Pen-Based User Interfaces, 2009.
[36] B. Signer, U. Kurmann, M. Norrie, IGesture: a general gesture recognition framework, in: 9th Conf. on Document Analysis and Recognition, Brazil, 2007.
[37] A. Milicevic, D. Jackson, M. Gligoric, D. Marinov, Model-based, event-driven programming paradigm for interactive web applications, in: OnWard! 2013, Indiana, USA, 2013.
[38] S. Sim, R. Gallardo-Valencia, Introduction: remixing snippets and reusing components, in: Finding Source Code on the Web for Remix and Reuse, Springer Science+Business, New York, 2013, p. 348.
[39] J. Farrell, An Object-Oriented Approach to Programming Logic and Design, Course Technology, Boston, 2013.
[40] O. Parra, S. España, O. Pastor, gestUI: a model-driven method and a tool for including gesture-based interaction in user interfaces, Complex Syst. Inf. Model. Q. 6 (2016) 73–92.
[41] J. Kolb, B. Rudner, M. Reichert, Gesture-based process modeling using multi-touch devices, Int. J. Inf. Syst. Model. Des. 4 (4) (2013) 48–69.
[42] O. Parra, S. España, O. Pastor, Tailoring user interfaces to include gesture-based interaction with gestUI, in: Conceptual Modeling, ER 2016, Gifu, Japan, in: Lecture Notes in Computer Science, 2016.
[43] O. Parra, S. España, J. Panach, O. Pastor, Extending and validating gestUI using Technical Action Research, in: 11th International Conference on Research Challenges in Information Science (RCIS), Brighton, UK, 2017.
[44] K. Krugler, Krugle code search architecture, in: Finding Source Code on the Web for Remix and Reuse, Springer Science+Business, New York, USA, 2013, p. 348.
[45] C. Wohlin, P. Runeson, M. Höst, M. Ohlsson, B. Regnell, A. Wesslèn, Experimentation in Software Engineering, Springer, Berlin, 2012.
[46] C. Truica, F. Radulescu, A. Boicea, I. Bucur, Performance evaluation for CRUD operations in asynchronously replicated document oriented database, in: 2015 20th International Conference on Control Systems and Computer Science, Bucharest, Romania, 2015.
[47] F. Li, W. Wang, J. Qu, H. Han, The design of adaptive user interface based on the grey relational grade, J. Phys. Conf. Ser. 1060 (1) (2018) 1–10.
[48] N. Juristo, A. Moreno, Basics of Software Engineering Experimentation, Springer US, 2001.
[49] O. Parra, S. España, O. Pastor, Including multi-stroke gesture-based interaction in user interfaces using a model-driven method, in: Proceedings of the XVI International Conference on Human Computer Interaction, INTERACCION '15, Vilanova i la Geltrú, Barcelona, 2015.
[50] D. Kieras, Using the Keystroke-Level Model to Estimate Execution Times, University of Michigan, Michigan, USA, 2001.
[51] S. Card, A. Newell, T. Moran, The Psychology of Human–Computer Interaction, L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1983.
[52] J.H. Kim, R.C. Miller, 6.813/6.831 User Interface Design, MIT, 2009. [Online]. Available: http://courses.csail.mit.edu/6.831/2009/handouts/ac18-predictive-evaluation/klm.shtml. (Accessed 28 October 2015).
[53] P. Runeson, Using students as experiment subjects – an analysis on graduate and freshmen student data, in: Proceedings 7th International Conference on Empirical Assessment Evaluation in Software Engineering, 2003, pp. 95–102.
[54] M. Svahnberg, A. Aurum, C. Wohlin, Using students as subjects – an empirical evaluation, in: Proceedings of the Second ACM–IEEE International Symposium on Empirical Software Engineering and Measurement, Kaiserslautern, Germany, 2008.
[55] F. Faul, E. Erdfelder, A.G. Lang, A. Buchner, G*Power3: a flexible statistical power analysis program for the social, behavioural, and biomedical sciences, Behav. Res. Methods 39 (2007) 175–191.
[56] K. Maxwell, Applied Statistics for Software Managers, Prentice–Hall, 2011.
[57] H. Boone, D. Boone, Analyzing Likert data, J. Ext. Sharing Knowledge, Enriching Extension 50 (2) (2012).
[58] J. Kotrlik, The incorporation of effect size in information technology, learning, and performance research, Inf. Technol. Learn. Perform. J. 21 (1) (2003) 1–7.
[59] D. Lakens, Calculating and reporting effect sizes to facilitate cumulative science: a practical primer for t-tests and ANOVAs, Front. Psychol. 4 (2013) 1–12, Article 863.
[60] S. Jamieson, Likert scales: how to (ab)use them, Med. Educ. 38 (2004) 1217–1218.