Received: 4 May 2021 | Revised: 9 May 2022 | Accepted: 29 May 2022

DOI: 10.1002/smr.2481

RESEARCH ARTICLE - EMPIRICAL

Journal of Software: Evolution and Process WILEY

Can non-developers learn a simplified modeling notation quickly?

Jorge Cervantes-Ojeda | María Gómez-Fuentes | Guillermo Chacón-Acosta

Applied Mathematics and Systems, Autonomous Metropolitan University, Mexico City, Mexico

Correspondence
Jorge Cervantes-Ojeda, Applied Mathematics and Systems, Autonomous Metropolitan University, Vasco de Quiroga 4871, Mexico City, Mexico.
Email: jcervantes@cua.uam.mx

Abstract

The User Interface Transition Diagram (UITD) is a simplified modeling notation to describe the transitions between the UIs of a software system. It is designed to be used for customer-developer s improving the models. This study investigates (i) which concepts of the UITD language are easy to understand and which ones require more effort to be understood and (ii) to what extent UITD comprehensibility depends on the user's programming skill level. A questionnaire was applied to a group of 334 non-developers after they received a very brief explanation of the UITD concepts. The results were analyzed using statistical methods. The overall average score of the group was 70.42%, which is quite acceptable considering the briefness of the explanation. We observed a very low correlation between the subjects' programming skill level and their scores. This suggests that, even for people who have little or no programming skills, the entirety of the UITD symbols and concepts should quickly be understood with little further training.

KEYWORDS
communication with nontechnical stakeholders, customer–developer communication, requirements specification, user interface flow specification

1 | INTRODUCTION

In the software development industry, the success of a project depends on how much the system satisfies the needs of the users. Most empirical studies show that user participation in the requirements specification phase has a positive impact on the project.[1] However, Mendez et al[2] reported that communication flaws between developers and customers are perceived as the second most significant problem in requirements engineering and the main problem that leads to project failure. The full specification of the system/user interactions and of the interfaces that a software system will have is one of the most important aspects in the requirements specification phase. It must be clear for both customers and developers what the services that can be requested to the system are and how they will be invoked by users. From there, one can think of improvements to the features of the system, whether functional or related to usability. Usability alone is one key element that can benefit a lot from the incorporation of customer and user feedback.[3] If nontechnical stakeholders can understand a tool for modeling user–system interactions, one could obtain valuable feedback from the customer and/or user. In addition, it is desirable that the time spent training the customers/users to help them understand this modeling language is short. Thus, most of their time can be spent in the creation of and feedback about the models.

User interfaces (UI) in a software system can be modeled with different tools (not necessarily mutually exclusive). These can be either running software or graphic notations. Some of the former are software products such as InVisionApp[4] and Sketch[5] that are conceived to decide the aesthetic side of UI prototypes and to simulate transitions, but they are not designed to specify, in a precise and complete manner, all the requirements of the transition triggers between UIs. One can also use some graphic notation, either informal or formal. Informal notations include Storyboards and Navigation aps.[6] The most comprehensive are the formal graphic notations such as the Interaction Flow Modeling Language (IFML),[7] which is the standard of the Object Management Group (OMG), to fully model the abstract descriptions of the UI. Other formal graphic

notations are colored Petri nets (CPN),[8] behavior trees (BT),[9,10] state machines (SM)[11] (all of these reviewed in Section 3.2) and User Interface Transition Diagrams (UITD) (briefly described in Section 3.1).

1.1 | Motivation

Understanding a formal modeling language is usually not trivial, because most of them are designed for software developers and not for nonexperts. Van der Linden et al[12] mention that nonexperts require simplicity when working with modeling notations. It would be very useful to have a modeling language that is formal and yet easy to learn because this would facilitate communication between stakeholders with and without technical software development skills.

However, it is more likely for a software development organization to be interested in adopting a notation if there is a study providing evidence of how easily it is understood. Therefore, it is important to carry out studies that evaluate different aspects of modeling languages. For example, there is a study[13] showing that graduate students with experience in the software industry had difficulties using the IFML symbols when modeling the front-end interfaces. This study identified some IFML elements that were misused. Performing studies similar to Randerson Queiroz and Marques[13] will provide information that would eventually lead to improvements either in the design of the languages or the way these languages are taught.

The UITD[14] is a formal graphical notation specialized in the specification of users' interactions with the system and the flow of the interfaces that the system will present to them. It has been applied in refs.[15–17] The UITD is technically accurate. With it, user–system interactions can be described in a complete way. Hence, it can be used reliably to start the development of the system. It is supposed to be very simple to learn, so it should improve communication between technical and nontechnical stakeholders. Here, we designed a test to investigate this claim. It should be noted that this is not a comparative study to other modeling notations, but it is rather our contribution to learn about the understandability of the UITD concepts for a considerably large sample of people after giving them a very brief training session.

1.2 | About our study

This study aims to investigate two aspects: (i) which concepts of the UITD language are easier to understand by nontechnical stakeholders and which ones require a longer explanation time and (ii) to what extent UITD comprehensibility is conditioned to users having a certain programming skill level. It is believed that most of the customers have none or just basic programming skills. We assume that the more programming skills people have, the better they can understand a model because they have developed an abstraction ability for certain systems. For the UITD to be a good communication tool between customers and developers, its comprehensibility should not be strongly conditioned on the customer programming skills, that is, the correlation between the subjects' programming skill level and their understanding level of the UITD should be low. In this work, we show that this low correlation exists.

We modeled a simple system using a UITD and presented it to a group of 334 undergraduate students, and, after a brief (10 min) introductory explanation about the UITD features, we asked them to answer a questionnaire. When asked about their current programming skill level, most of the participants considered themselves beginners or uninitiated in programming. Our results show that, after a short explanation, people can understand most of the UITD notation and that their level of programming skill has little influence on this understanding. This suggests that the UITD notation is indeed easy to understand and that it might be worth giving it a try as a modeling tool in software development projects to communicate with nontechnical stakeholders, after more than just a brief explanation, of course.

This is the first of a series of formal studies on the understandability of the UITD by nontechnical stakeholders. We were able to include 334 engineering students who were willing to do both: receive training about the UITD and answer a questionnaire. In order to extend the validity of our results, users from other backgrounds, for example, business and media, will be considered in future work.

The rest of this work is structured as follows: Section 2 presents a review of the work that, to the best of our knowledge, has been done regarding the customer–developer communication and the comprehensibility of modeling notations. Section 3 formally defines the UITD, explains its features, provides a comparison of the UITD with other tools for modeling the behavior of UIs in a software system, and gives some examples. Section 4 describes the experimental study methodology. Section 5 shows results and answers to the research questions. Section 6 contains conclusions, and Section 7 contains discussion and future work.

2 | RELATED WORKS

Customer–developer communication has been addressed in different fields, such as information systems, human–computer interaction, and requirements engineering.[18] Regarding work in the area of requirements, Ricca et al[19] found that the screen mock-ups used to complement the

use cases facilitated the understanding of functional requirements during communication with stakeholders. One of the advantages of the UITD is to help in the understanding of the way in which the user is required to interact with the system. The fact that the UITD can easily be understood without having programming skills makes the UITD a useful tool for communicating with customers during the requirements elicitation process. Then, a combination of the UITD with the screen mock-ups (as explained in literature[14–16]) could be very useful during the requirements elicitation phase.

Ogunyemi et al[3] report that there are few works related to navigation design. The UITD is a modeling tool that is especially useful for designing navigation between a system's UIs. Here, by showing the potential usefulness of the UITD in this area, we help practitioners to motivate themselves to use this modeling tool.

High-quality models are decisive when building software. The quality of models depends on different aspects.[20] Comprehensibility is a key quality of modeling languages because if a diagram is not assimilated correctly by the receiver, then communication cannot be properly established.[21] Research on model comprehension has been done in different areas. Regarding business process modeling, Recker et al[22] identified user characteristics that influence the comprehension of conceptual models of business domains, and Polančič and Gregor[23] investigated the perception of the modelers and the modeling times when using different modeling tools. Mendling et al[24] present a broad review on model comprehension studies in this area. Regarding other areas, recent work on model comprehensibility has been done in testing[25,26]; Liebel and Tichy[27] performed a study comparing the comprehensibility of functional requirements modeled in different graphical modeling languages. Scanniello et al[28] reported an empirical study on how models help in the understanding of source code. In other less recent studies, Cruz-Lemus et al[29] investigate whether the use of composite states may improve the understandability of UML state charts or not, and they highlight the importance of the understandability of the models. The main focus of all these studies is the perception of modelers and not that of nonexperts. The understanding of nonexperts is the main aspect to consider when establishing communication with customers.

Gómez and Cervantes[14] present results of a questionnaire applied to 46 people, in which the UITD effectiveness seems to be favorable, but it still needs to be extended to a more formal study. In this work, the number of respondents is much higher, and we carried out a formal statistical analysis according to the software engineering experimentation procedure given in Wohlin et al,[30] which gives our results more validity than those in Gómez and Cervantes.[14]

Bridging the gap between developers and customers during software development processes is a topic of interest.[31] Because the UITD is intended to improve communication between stakeholders in a software development project, it should help to reduce the gap between them. As far as we know, there is no recent study that evaluates how well nontechnical stakeholders can understand diagrams written in a notation that models the transitions between UIs.

3 | THE UITD

The UITD graphical notation was designed to simplify the specification and design of system–user interactions without losing the technical detail that is necessary to develop the system. This simplification should allow customers and users (most of them non-developers) to participate in the specification of such interactions. In turn, this would imply that the requirements specification has a better degree of validity because if customers can understand the model of the system that they will use, they can say whether or not the modeled system is the required one.

3.1 | Definition

A UITD is a directed graph with nodes representing UIs and edges representing transitions between UIs. Each of the UIs has a name and a unique number that allows easy identification during the design and subsequent stages of the project. Transitions have an origin and a destination UI and a label. Labels in transitions are composed of exactly one mandatory user action on the origin UI, and, whenever two transitions with the same origin UI are triggered by the same user action, the label also contains additional information about the conditions that need to be met in order to trigger the transition. Labels do not have a specific format to include those conditions, but instead, they have a free format. This is done on purpose so that nontechnical stakeholders do not need to think about *how* to write them, but only about *what* to write. Developers should easily figure out what the action is and what the condition is in each case, and any ambiguity should also easily be clarified. In Gómez and Cervantes,[14] it is demonstrated how the UITD complies with visual notation design rules proposed by Moody.[32]

Figure 1 gives an example of a UITD that describes a portion of a system. From interface #1, there are two possible transition triggers: (i) Search by ID that sets interface #2 (Enter item ID), and (ii) Add new item that sets interface #5. Interfaces #2 and #5 also have their respective transitions, and each of these leads to its corresponding destination interface. In interface #2 when the user enters an item_id and clicks view, two transitions can be triggered: *Enter existing ID and click view* or *Enter non-existing ID and click view*. So, it should be understood that the transition to be triggered depends on the condition: existing ID or non-existing ID.

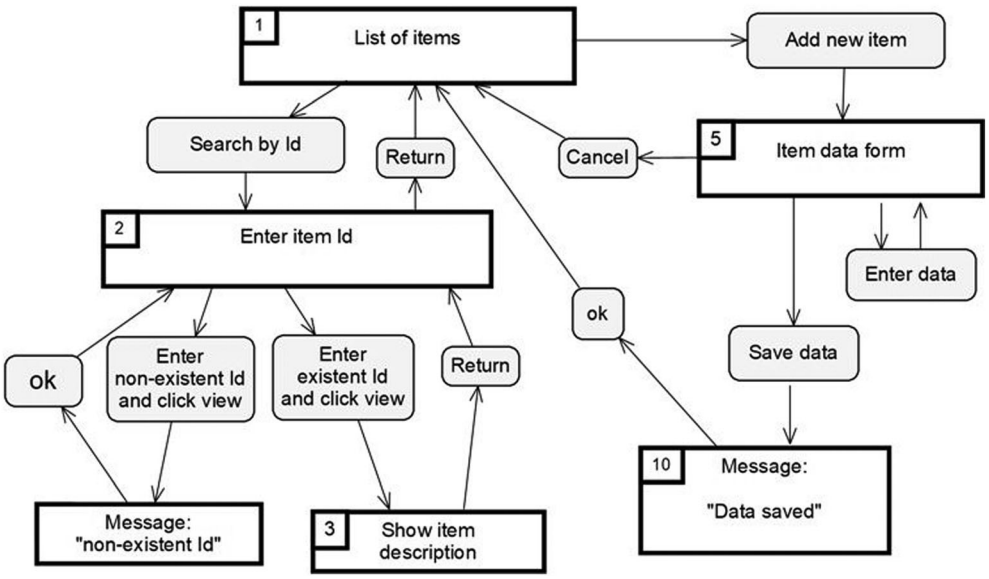


FIGURE 1 Example of a UITD describing a portion of a system

The UITD figures in this work were created with Pacestar UML Diagrammer (www.pacestar.com/uml/), which is not free. Free tools like the UITD Editor[33] (http://148.206.168.145/EditorUITDEnglish/examples/indexF.html) and Lucidchart (www.lucidchart.com) can also be used to draw a UITD.

3.2 | The UITD and other graphical notations

Storyboards are considered an informal graphic notation that is used by customers and software developers to understand the workflows in the system. They are very simple. They consist of multiple simplified screens, which only contain the basic elements and links needed for the interaction. However, they do not describe the full UI interaction set or the full detail of each UI because they only show the steps of a single use case. In contrast, the UITD includes the full UI interactions set and leaves the design of UI details to be described separately. In this way, all the information can be properly documented.

Similarly, the *Navigation Map* [6] models the transitions between interaction contexts (a screen, dialogue box, or window) also associated with a single use case. Unlike the Navigation Map, the UITD does not include UI details, thus trying to keep the reader focused on UI flows. This is because, during the requirements elicitation phase, the most important aspect is the identification and validation of all user actions for each UI. The essential difference is that navigation maps do not have a methodology to label each transition as in the UITD ignoring its dependency on system states, which is a key element in user–system interaction design.

Transitions between UIs can be modeled with general-purpose formal notations like CPN[8] and BT,[9,10] which can graphically represent any kind of behavior that is required for a system. In UML SM,[11], when an event occurs, the SM responds by performing actions, such as changing a variable, performing I/O, invoking a function, generating another event instance, or changing to another state. So, SM can also be used to model the UI flow. SM are not limited to model transitions between UIs; they aim to model all kinds of events that are directly mapped to a code that must be executed in response. IFML aims to be used by system architects, software engineers, and software developers in order to express "the content, user interaction and control behavior of the front-end of software applications."[7] It has been mainly used in the context of model-driven development.[13]

These aforementioned formal notations were not specifically designed to be used by nontechnical stakeholders, who are normally not familiar with the details included in them. Some of the symbols correspond to system tasks that are needed to provide a service, but users do not need to perform any action regarding them, and so, they are not interested in checking its validity when the interactions between system and user are being specified. Internal system tasks do not need to be explicitly specified at the customer–developer communication stage. If customers are interested in those internal system tasks, they can check them in the documentation from further development stages. It is expected that customers are mainly interested in *what* tasks the system can do and not in *how* they are done. Including too much detail in models does not meet the simplicity that nonexperts require[12] because it just makes models more difficult to be understood. With the UITD, users can and should assume that these system tasks are executed "somehow" internally, but the UITD provides a sufficiently detailed specification of the functional

requirements due to the association of transitions with not only user actions but also with the relevant internal system states. Therefore, the software developers can start the detailed definition of the internal system tasks *from* a UITD. The UITD does not intend to replace other formal modeling notations used for design, neither to be a source for automatic code generation (although it can, and we are working on that), but rather to hide the implementation details, so that the customer has a clearer vision of what the user can do on the system regardless of what happens internally, and this is done strictly without removing any requirement. The UITD offers the simplicity that nonexperts require and the completeness that developers require.[14]

We can see from Table 1 that the UITD notation is simpler than the other formal notations mentioned above in terms of the number of graphical symbols that are defined for them.

One of the principles of the physics of notations theory[32] is *graphic economy*, which means that the total number of graphical symbols must be cognitively manageable, and, according to Miller's law, this number must be ≤9.[34] As can be seen from Table 1, among the formal notations that can be used to define the transitions between the UIs, the UITD is the only one that complies with the graphic economy property.

As stated by Nelson et al[20] and Krogstie,[35] language quality is a balance between (among other things) expressiveness, comprehensibility, and fit for purpose (e.g., for use in code generation). Given this trade-off between expressiveness and comprehensibility and considering that the UITD purpose is to improve customer–developer communication at the requirements specification phase, the UITD notation is deliberately designed to have limited expressiveness in favor of higher comprehensibility.

The UITD's *graphic economy* should make it easy to be understood. In this study, we investigate to what extent respondents can understand the UITD concepts represented by this short list of symbols.

3.3 | Other UITD features

3.3.1 | An interface contained within another interface

One of the UITD features is the ability to represent an interface within another interface. The meaning of this is that all transitions with origin in the contained interface are also available from the containing interface (and not vice versa). For example, when a system has several kinds of users, it is frequently required that the same interface is given with additional features (e.g., buttons) with which only certain authorized users can interact. Figure 2 shows an example in which there are two kinds of users: supervisor and cashier. Cashiers have access to interface #6 where they can only "register a purchase" or "check a price," whereas the supervisor has access to interface #5, which has the same options as interface #6, but additionally it has the option "modify product features." Another example where this feature is useful is when several UIs contain elements or sets of elements that are the same in each UI. In this case, one can define a UI that contains these elements and place it inside each of those UIs that replicate them. A typical example is a navigation menu that is shown in several UIs.

3.3.2 | Bold and non-bold UI

A bold UI box means that all the possible transitions to/from this UI are visible in the current UITD page or subdiagram. For instance, in Figure 2, UI #7, #8, and #9 are not in bold because some transitions to/from them are missing in this figure. This feature provides constructability to UITDs because the full diagram can be created by a set of subdiagrams. The idea behind the bold border is to be able to divide the UITD into several parts. A UI can be present in various fragments. When a particular UI has its border in bold in a certain fragment, it is said to be complete. This means that all the transitions that are connected to/from this UI are documented in that fragment. Otherwise, it means that only a subset of its transitions are visible in that fragment. It is recommended that all UIs appear in bold at least in one fragment. In the full UITD (the one that contains the entire system), if done, the borders of all the UIs should be in bold.

TABLE 1 Graphical symbols used by modeling notations

Modeling notation	Number of graphical symbols
CPN[8]	9**n* (see the note below)
IFML[7]	26
UML state machines[11]	15
Behavior trees[9]	12
UITD[14]	5

Note: There is a 9-tuple, where one of the elements is a non-empty color set of size *n*.

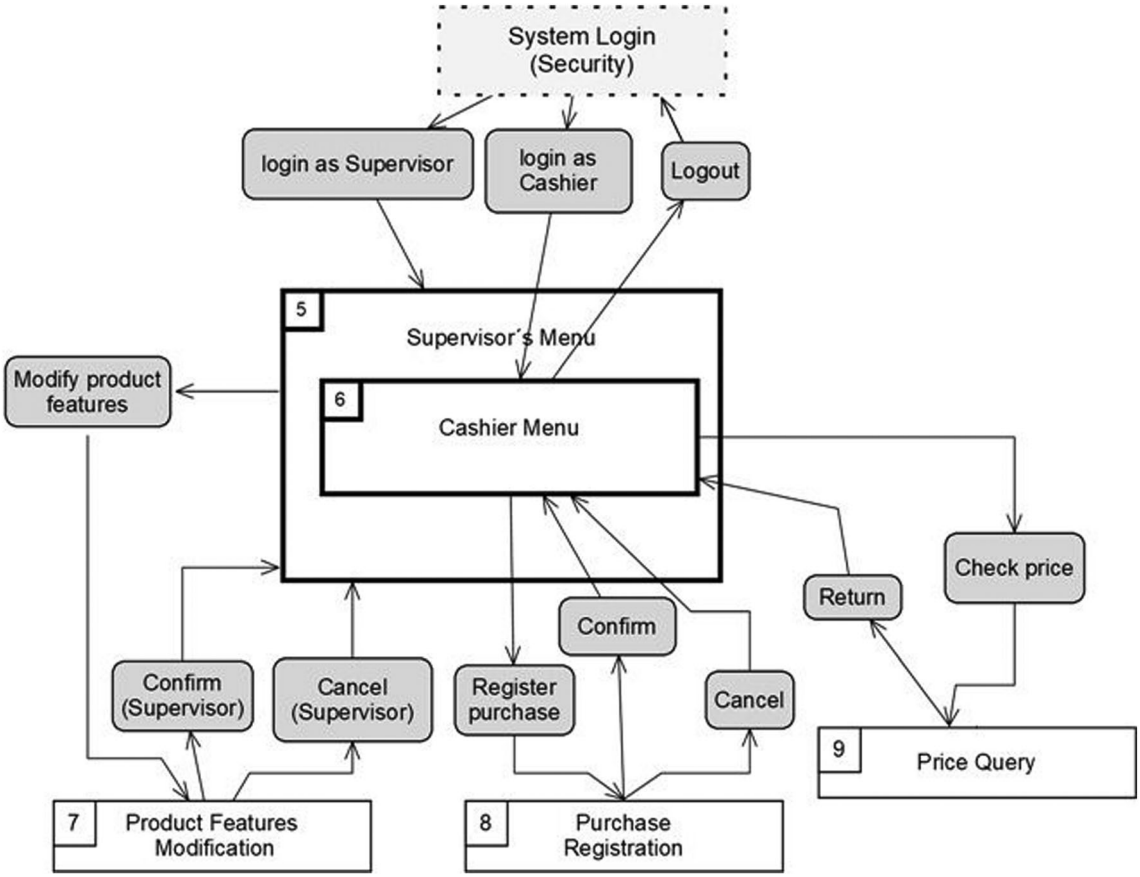


FIGURE 2 Example of an interface contained within another

3.3.3 | Representing a subset of a UITD as a box

Note that in Figure 2, there is a rectangle with a dotted line called "System login (security)." This is a block that represents a subset of the UITD that is dedicated to the security subsystem. Transitions from/to this block are indicated without specifying the particular UI inside the block.

3.4 | UITD graphical symbols

The UITD graphical symbols are summarized in Table 2.

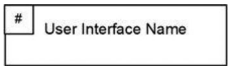
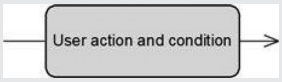
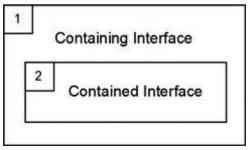
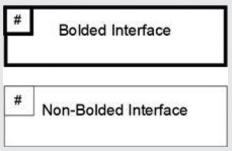
We assume that some of the UITD concepts will be easier to understand than others. So, which concepts are easy, and which are difficult? This is one of the questions that this study will have to answer.

In the next section, we describe our experimental study designed to evaluate the comprehensibility of the UITD. We used the recommendations in Wohlin et al[30] for the preparation and report of the study.

4 | EXPERIMENTAL DESIGN

Here, we describe the experimental design and the rationale behind it. First, we define the purpose of our study and the research questions that will be answered later according to our results. We then describe our group of respondents and explain why they are feasible for this study. Then, we describe the instrument used for the study in detail and its relations to the research questions. The selection of variables, hypothesis formulations, and experimental procedure are described next. Finally, the analysis procedure is given.

TABLE 2 UITD graphical symbols

Symbol	Meaning	UITD notation
Single UI	A state of the UI displayed to the user	
Transition	A change in the state of the UI from a user action (and sometimes a condition)	
Nested UI	Transitions with origin in the *contained* interface are also available from the *containing* interface and not vice versa	
Bold and non-bold UI	Bold UI box means that all the possible transitions to/from this UI are visible in the current UITD fragment or subdiagram	
UITD subdiagram	Transitions from/to a subset are indicated without specifying the particular UI inside the block.	

4.1 | Purpose of the study

The purpose of this study is to provide some evidence of the extent of understandability that a UITD has and its correlation with the programming skills of stakeholders.

4.2 | Research questions

There are two main aspects that we want to investigate:

- RQ1: After a brief explanation, when faced with the UITD for the first time, which UITD concepts are easy to understand, and which require further explanation to be better understood?
- RQ2: To what extent does the respondents' understanding level depend on their programming skill level?

4.3 | Selection of subjects

The source population of this study was the group of attendees at a session in a conference that was addressed to students of computer systems engineering, informatics, and robotics. We also had some volunteers of our computer engineering undergraduate program at the Autonomous Metropolitan University (in Mexico City). We had 334 respondents in total. It is worth noting that it is quite difficult for us to get that number of real-life software development customers willing to receive a training session and participate in a study. However, we consider that the subjects who participated in this study are very representative of the potential users of the UITD because they do not yet know how to create complex software systems. However, they do have an interest in understanding how a system will be operated. These characteristics are common to any nontechnical stakeholder.

4.4 | Instrument design

The instrument that we used in this survey is a questionnaire mostly based on the case study "Conference Review System."[36] This case study is simple enough for anyone to understand with a short explanation, yet it includes most of the situations that are commonly seen in any interactive software system. We did not want to introduce high complexity to our case study in order to guarantee the construct validity (see Section 5.4).

The UITD in Figure 3 is an improved version of the one used in Gómez and Cervantes[14] where some labels in the transitions are clearer. It describes some user interfaces and actions that can be performed in a system that helps in the organization of a conference. Figure 3 shows an example where the main UITD features are used. However, it does not represent the full operation of a real-life system. Those interested in seeing a real-life system modeled with UITD can consult.[15,16]

The questions included in our instrument are described below.

4.4.1 | UITD constructs understanding

This group of questions (in Table 3) has seven questions that aim to evaluate how well the respondents understand a UITD. Respondents are asked to choose the answers that they consider correct for each of the questions that are formulated in Table 3 (the correct answer is shown in bold).

Each question measures the understanding of one concept. We explain the goal of each question below.

Questions #1 and #2 are to evaluate how well people understand that transitions that arise from the *containing* interface cannot be triggered from the *contained* one.

Questions #4 and #6 detect whether people understand that transitions that are triggered from a *contained* interface can also be triggered from the *containing* interface.

Questions #3 and #5 detect whether the subject understands that some UIs may not be reachable depending on the given conditions.

Question #7 detects if people identify a rectangle with a dotted border as a block representing a subset of the existing interfaces in the full UITD.

4.4.2 | Subject's programming skills

We formulated Question #8 (in Table 4) in order to identify how the respondents programming skills influences the UITD interpretation.

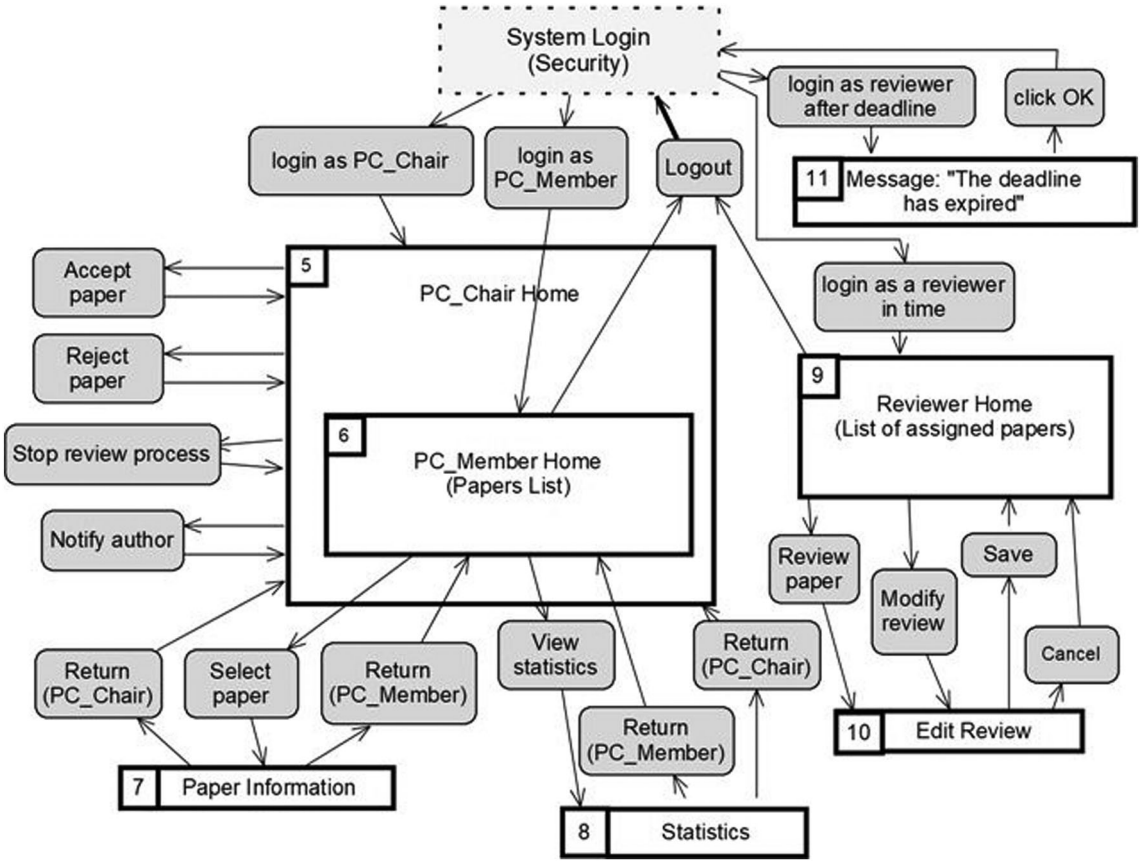


FIGURE 3 UITD of a system that helps to organize a conference

TABLE 3 Questions about UITD understanding

Question	Correct answer
1. Can the PC_Chair accept or reject a paper?	a) **Yes** b) No
2. Can a PC_Member accept or reject a paper?	a) Yes b) **No**
3. Who notifies authors about the final decision for their paper?	**a) PC_Chair** b) PC_Member c) Reviewer
4. Select the users who can view statistics (you can choose one or more options)	**a) PC_Chair** **b) PC_Member** c) Reviewer
5. Can the PC_Chair review a paper?	a) Yes b) **No**
6. Select the number of the user interface that is displayed after the PC_Chair or PC_member selects a paper, from the paper list	**7 (Paper information)**
7. Does the user need to pass through a security sub-system to be able to enter?	a) **Yes** b) No

TABLE 4 Subject's profile questions

Question	Options
8. If the highest level is a professional software developer, what is your programming level?	a) Null (I do not know how to program) b) Low (I'm taking a first programming course) c) Medium (I have taken a few programming courses) d) Advanced (I have already built some software systems) e) Professional (I work as a programmer)

4.5 | Variables selection

4.5.1 | Variables for RQ1

We measure the fraction of individuals μ that have correctly answered each questionnaire question.

4.5.2 | Variables for RQ2

In order to answer our RQ2, we want to demonstrate that there is no high correlation between the subjects' programming skill level and their understanding level of the UITD. So, in the following analysis, we consider the *programming skill level* as the *only* independent variable x and the *number of correct answers* as the response variable y.

4.6 | Hypothesis formulation

4.6.1 | RQ.1

With the answers to questionnaire Questions #1 to #7 (in Table 3), we detect which UITD concepts are easy to be understood and which require further explanations. To have confidence in the validity of the obtained results, we need to discard the hypothesis that the answers were given randomly. For each questionnaire question, we formulate a null hypothesis in which it is assumed that the answers were given randomly.

For Questions #1, #2, #5, and #7, there are only two possible answers. So, the null hypothesis for each of these questions is the same:

H_0: $\mu_0 = 0.5$ versus the alternative
H_1: $\mu_1 > 0.5$

For Question #3, there are three possible answers. So, the null hypothesis is the following:

H_0: $\mu_0 = 1/3$ versus the alternative
H_1: $\mu_1 > 1/3$

For Question #4, there are seven possible answers (the respondent can select one, two, or three responses out of three). Question #6 has also seven possible answers (because there are seven UIs in Figure 3). So, the null hypothesis for Questions #4 and #7 is the following:

H_0: $\mu_0 = 1/7$ versus the alternative
H_1: $\mu_1 > 1/7$

4.6.2 | RQ.2

There are two parameters that can be used to measure the level of association between two random variables. One is the coefficient of determination R^2, which is the proportion of the total variation in the dependent variable that can be explained by variations in the independent variable of the regression model. And the other is the correlation coefficient R. In order to show that UITD comprehensibility is not strongly conditioned to the programming skill level of its users, these parameters must have low values.

Because we use a statistical inference method to measure the correlation between our variables x and y, we set our null and alternative hypotheses as follows:

H_0: $R = 0$ versus the alternative
H_1: $R \neq 0$

4.7 | Experimental procedure

The questionnaire in Tables 3 and 4 was applied to a group of volunteers. A short 10-min explanation about what the UITD means was given beforehand. The questionnaire was answered by the participants within approximately 15–20 min. The results were processed anonymously.

In the presentation shown to respondents,* all the UITD concepts were explained, including some examples. Those examples were different from the one presented to them in the questionnaire to avoid respondents simply copying the answers from the given examples.

Randomness of the subjects. We asked a group of about 500 people (attendees at an invited talk in a conference) to answer the questionnaire, and 309 replied. Additionally, the instrument was also applied in two groups of computer engineering students from the Autonomous Metropolitan University (Mexico) adding 25 more replies (participation was also voluntary).

Data collection. Subjects could choose between answering the questionnaire online or on paper. At the end, of the responses we received, 194 were online, and 140 were on paper.

Data validation. According to Vannete,[37] including attention-check questions in the questionnaire may cause respondents to behave worse later in the survey. Therefore, we did not include them to avoid this risk, and, because our questionnaire is short, we considered that it is relatively easy for respondents to maintain a good level of concentration during the survey.

4.8 | Analysis procedure

4.8.1 | Data set preparation

With the aim of reducing the amount of data that is handled by the persons who collected it from the survey sheets, we distributed the data registration in three teams of two people each. This facilitates the acquisition of clean data, which is a prerequisite for any data analysis.[38] Each team handled a block with the responses of around 100 subjects. Each team transferred their data to a spreadsheet. To promote accuracy in the data, one person was reading results while another was capturing them. After doing this, each team verified that the captured data were correct.

4.8.2 | Hypothesis test for RQ1

The results of the hypothesis test for each question are shown in Table 5. The last column indicates whether or not the null hypothesis can be rejected based on the obtained p-value, with a confidence level of 99%.

TABLE 5 Hypothesis test results

Question	μ_0	μ_1	*p*-value	Discard H_0
1	0.5	0.96	<0.01	Yes
2	0.5	0.80	<0.01	Yes
3	0.3333	0.65	<0.01	Yes
4	0.1428	0.53	<0.01	Yes
5	0.5	0.52	0.2	No
6	0.1428	0.75	<0.01	Yes
7	0.5	0.72	<0.01	Yes

Based on Table 5, we can say that our results were not obtained randomly, except for Question #5. So, for this question, there is a 20% of probability of getting the result we obtained, or better, if the respondents answered randomly, which does not allow us to reject the null hypothesis in this case.

4.8.3 | Hypothesis test for RQ2

In order to test the relation between the declared skill of the participants (x) and the number of correct answers (y), it is standard to create a linear regression model between x and y.[39] However, for the hypothesis test to be based on an estimator with a Student t-distribution, we would have to assume a bivariate normal joint distribution between variables. For this, we verify whether the samples are normal through the so-called Shapiro–Wilk test for which we need the order statistics $x_{(i)}$ (and similarly for $y_{(i)}$) defined by sorting the values of the sample in increasing order, that is, $x_{(i)} \leq x_{(i+1)}$ for all the sample elements. Thus, the test statistics are defined as follows:

$$W = \frac{\sum_i \left(a_i x_{(i)}\right)^2}{\sum_i (x_i - \bar{x})^2}, \tag{1}$$

where the denominator is just the sample variance and the coefficients a_i are functions of the mean values and the covariance matrix of the order statistics $x_{(i)}$. They can be calculated straightforwardly or found tabulated in Shapiro and Wilk,[40] for instance, for a sample size of 50. Values of W less than 1 lead to rejecting the null hypothesis while accepted when the statistic is equal to 1.

We now present the results of the tests for both variables: (i) the declared skill level and (ii) the questionnaire scores. These were calculated using the online tool at https://www.statskingdom.com/shapiro-wilk-test-calculator.html with $\alpha = 0.05$.

a. Shapiro–Wilk test, using normal distribution (right-tailed) for the skill-level variable:

Because $n > 50$, we used the normal approximation to calculate the p-value.

1. H_0 hypothesis: The sample was taken from a normal distribution.

p-value $= 0 < \alpha$, we reject the H_0. The difference between the data sample and the normal distribution is great enough to be statistically significant.

2. *P*-value:

The p-value equals **0**, ($P(x \leq 8.3199) = 1$). This means that the chance of a Type I error (rejecting a correct H_0) is small: 0 (0%). The smaller the p-value, the more it supports H_1.

3. Test statistic:

The test statistic ***W*** equals **0.8497**, which is not within the 95% region of acceptance: [0.9912: 1].

4. Effect size:

The observed effect size **KS - D** is **large**, **0.3025**. This indicates that the magnitude of the difference between the sample distribution and the normal distribution is large.

b. Shapiro–Wilk test, using normal distribution (right-tailed) for the scores:

Because $n > 50$, we used the normal approximation to calculate the *p*-value.

1. H_0 hypothesis: The sample was taken from a normal distribution.

Because *p*-value = 6.234e-12 < α, we reject the H_0. The difference between the data sample and the normal distribution is great enough to be statistically significant.

2. *P*-value:

The *p*-value equals **6.234e-12**, ($P(x \leq 6.7747) = 1$). This means that the chance of a Type I error (rejecting a correct H_0) is small: 6.234e-12 (6.2e-10%). The smaller the *p*-value, the more it supports H_1.

3. Test statistic:

The test statistic **W** equals **0.9246**, which is not within the 95% region of acceptance: [0.9914: 1].

4. Effect size:

The observed effect size **KS - D** is **large**, **0.1642**. This indicates that the magnitude of the difference between the sample distribution and the normal distribution is large.

Based on these results, the null hypothesis is rejected, and, in both cases, we must assume we do not have normal samples. If standard assumptions about the distributions of the underlying random variables or sample data cannot be guaranteed, nonparametric methods allow testing hypotheses on samples with unspecified distributions.[39] The Spearman's rank correlation coefficient R_s, also called Spearman's rho, is a nonparametric test used to measure the strength of association between two variables. The value $R_s = 1$ means a perfect positive correlation, and the value $R_s = -1$ means a perfect negative correlation. Small values of R_s mean a low correlation between variables. The Spearman's rank correlation coefficient R_s can be calculated by substituting the ranks as the paired measurements in the expression of the sample correlation coefficient (2).

$$R = \frac{\sum (x_i - \overline{x})(y_i - \overline{y})}{\sqrt{\sum (x_i - \overline{x})^2 \sum (y_i - \overline{y})^2}}, \tag{2}$$

This coefficient can be reduced to a more manageable expression as follows:

$$R_s = 1 - \frac{6 \sum_i (r(x_i) - r(y_i))^2}{n(n^2 - 1)}, \tag{3}$$

where *r* is the rank of the corresponding data, which is obtained by ordering data from smallest to largest value for each variable; when there are repeated data, a simple arithmetic average is calculated to establish the rank. So, we will use Spearman's R_s as a statistical test to measure the relationship between our variables *x* and *y*.

5 | RESULTS AND ANSWERS TO RESEARCH QUESTIONS

5.1 | UITD understanding and RQ1

According to the results obtained in Section 4.8.2, out of seven questionnaire questions, six were not answered randomly with a confidence level of 99%. The null hypothesis could not be rejected only in questionnaire Question #5. This means that, for this question, we cannot make inferences from the data. So, this is one of our findings that we can use because now we know that more emphasis should be used when explaining the concept related to Question #5.

Question #4 has a low group average score of 53%, but the subjects did not answer randomly. If they had answered randomly, the score would have been about 14% (according to Table 5). This means that those subjects who answered correctly did understand the concept with which this question is related: "transitions that are triggered from a *contained* interface can also be triggered from the *containing* interface." Therefore, we believe that a better group score is possible with further explanations.

In Table 6, we show a list of UITD concepts sorted from the easiest to the hardest to understand according to our results. The easiest questions for the subjects were 1 and 2.

Based on the aforementioned results, we can answer RQ1 as follows: The most difficult concept to understand is that "Transitions that are triggered from a *contained* interface can also be triggered from the *containing* interface," while the easiest concept is that "transitions that arise from the *containing* interface cannot be triggered from the *contained* one."

The results in Figure 4 show the obtained frequencies of subjects for each possible number of correct answers, there is a general shift toward high scores. This means that very few respondents had few correct answers and most of them obtained passing scores. The overall average score of the group was 70.42%. The above suggests that most of the UITD concepts are clear enough to be understood when people receive a brief explanation.

5.2 | Influence of people's programming skills on their understanding level and RQ2

In Figure 5, we show the distribution of the respondents' declared skill level. As can be seen, most of them are in the low skill level, which corresponds to people taking their first programming courses. This, along with Figure 4, reinforces the assumption that it is not necessary to know how to program to be able to understand the UITD.

In this section, we investigate the correlation between programming skill level and the UITD understanding level of individuals in order to respond to our RQ2. To measure this correlation, we use Spearman's correlation coefficient as in Equation (3). We measure the degree to which skill level influences the results. As our scale of values for the *x*-axis (programming skill level) is arbitrary, we calculate Spearman's correlation

TABLE 6 UITD concepts sorted by average score

Q#	Score	Related concept
1	0.96	Transitions that arise from the *containing* interface cannot be triggered from the *contained* one
2	0.80	
6	0.75	Transition labels show the conditions needed to trigger a transition
7	0.72	A rectangle with dotted line is a block representing a subset of the existing interfaces in the full UITD
3	0.65	The available set of transitions to a certain user type can be different from other user types
4	0.53	Transitions that are triggered from a *contained* interface can also be triggered from the *containing* interface

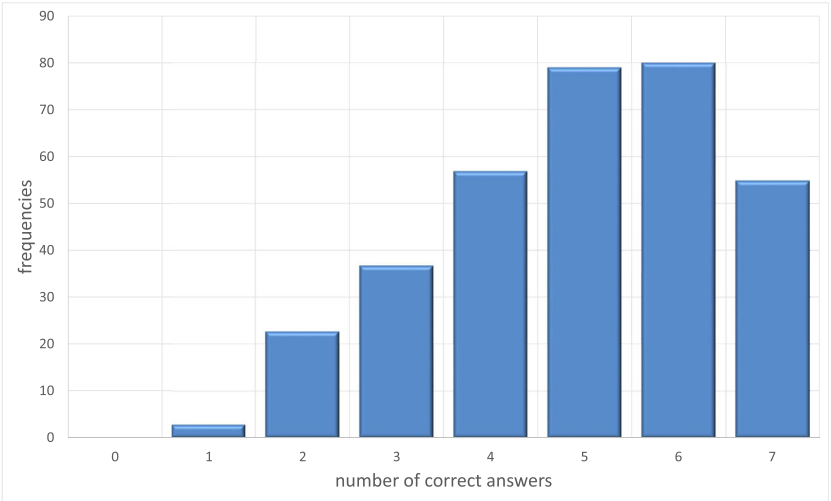


FIGURE 4 Frequencies by number of correct answers

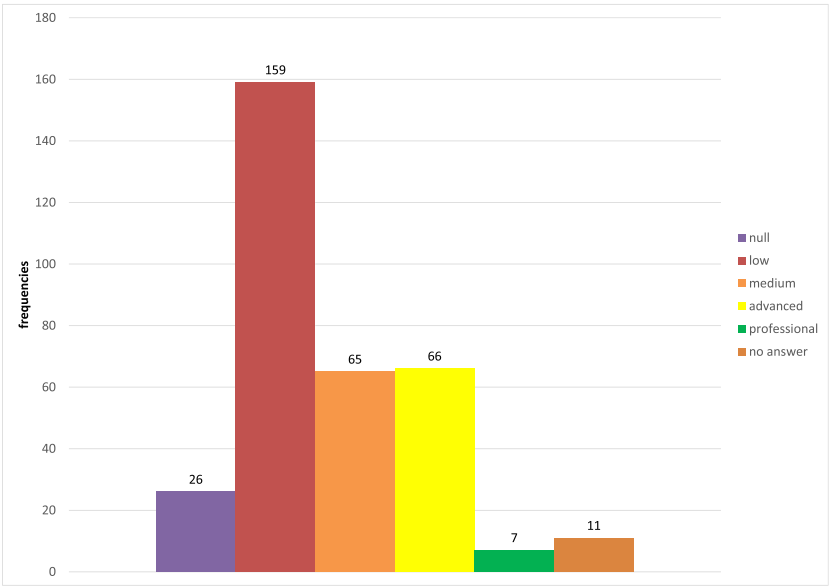


FIGURE 5 Frequencies by programming skill level

coefficient, which depends solely on the place that each value occupies within the set of values, as in Equation (3). We obtained Spearman's correlation coefficient $R_s = 0.2020488$ and $(R_s)^2 = 0.04080597 = 4.08\%$, which is a very low value. Because this is a very low value, we now test the null hypothesis for Spearman's correlation coefficient H_0: $R_s = 0$, against the alternative hypothesis H_1: $R_s \neq 0$. The hypothesis test in this case was done for the test statistic $\sqrt{n-1}R_s$, which approximately has a standard normal distribution. Thus, for a two-tailed normal test, the corresponding significance level or p-value in this case is 0.0002 (less than 0.01), so we can conclude that there is nonzero correlation between the skill level and the correct answers for individual subjects. However, variations in the skill level can only explain a very low percentage (4.08%) of the variation in the obtained score.

The interpretation of Spearman's correlation coefficient has some variations depending on the specific research area and specialty.[41] So, there are different tables that measure the relative strength of a linear relationship between two variables based on the correlation coefficient between them. Our result $R = 0.20205738$ indicates a low or very low correlation in all the consulted tables.[42–44] Therefore, we can say that the correlation between x and y is weak and that the variations in the questionnaire results depend little on the participant's skill level. Then, regarding RQ2, we can say that *the people's understanding of the UITD has a weak dependency on their programming skill level*.

5.3 | Power of the test

A power analysis was conducted using the G*Power 3 software.[45] We used the linear bivariate regression: one group, size of slope, one-tail test to determine the required sample size, and the power of the test for correlation, in Section 5.2. The data were transformed to rank average, and a linear trend line was adjusted from which we took its slope H1 = 0.2133, with α err prob = 0.05, power (1 − β err prob) = 0.95, slope H0 = 0, std dev σ_x = 86.4475, std dev σ_y = 91.3136. The G*Power result indicates that 256 samples are needed to get power = 0.9500391. We had a total sample size of 323 subjects (excluding 11 subjects who did not report their skill level), which means we can conclude that our sample size was sufficient for this power.

5.4 | Validity threats

The main validity threats[46] for this research concern uncontrollability and construct validity.

Uncontrollability: Our experiment was designed to evaluate the ease of comprehension of the UITD. To do so, we measured the UITD comprehension in terms of the correctness of the answers in the multiple-choice questionnaire. There is a risk that some of our respondents have guessed the system behavior for certain questions. This is something that we could not control. However, the high statistical power of the test given by the high number of participants makes us think that this risk does not significantly alter the trend of the results.

Construct validity: One of the UITD features was excluded from this study. This feature is the bolding/non-bolding of nodes (see Section 3.3.2), which is used when the modeled system is too large to be described in a single UITD, and so, it has to be constructed by means of several partial fragments. We believe that the exclusion of this feature does not affect our validity because it is not an essential feature for building a UITD. In the applied questionnaire, we used a diagram of a system that can be modeled with a single UITD. Additionally, it should be noted that we draw conclusions based on the results obtained from one particular system. There is a trade-off between the number of people willing to participate in a study and its duration. We wanted to have a short study, with many volunteers willing to participate, and thus have a sample as large as possible. Our results help us to get an idea of how well the UITD symbols are understood, which ones are best understood, and which require further explanations. However, a similar study that includes two or more systems would reinforce our conclusions. Of course, this implies that it will be quite difficult to match the number of participants in this study. On the other hand, there is also a comprehension challenge when models are large, so another study will be devoted to this particular issue. Recently, we released the UITD editor[33] that has specialized features for managing complexity (bold/non-bold and subdiagrams), and we will use this new tool to carry out the study.

External validity: Our group consists of those beginning their engineering careers and does not represent the full set of potential UITD users; however, the results of this study are important because our group is mainly formed of people who are not SW developers, which is within our interest group, so it provides limited but valuable information about the purpose of this study. The group of respondents could be diversified in a later study to include people who do not belong to the engineering area but who are interested in having a SW system built, for example, business and media.

Note that the low number of subjects who declare to be experts makes the sample unrepresentative for this particular skill level. However, because nonexperts were able to understand UITDs quite well, it is unnecessary, we assume, to test the expert group extensively because it is very unlikely that they perform significantly worse than nonexperts.

6 | CONCLUSIONS

The UITD[14] is meant to help software developers to better communicate with the customer in order to specify the transitions between the different UI presentations (states) of an interactive system. Given that communication flaws between developers and the customer are one of the most important problems that have been identified in requirements engineering,[2] clear evidence is needed about the UITD's effectiveness with respect to its understandability mainly by nontechnical stakeholders so that the software development community can be aware of it and decide whether or not to use it. To do that, we took a sample of 334 subjects, all of them without any previous knowledge about the UITD but having received a brief explanation and tested their level of understanding of the concepts that a UITD communicates. All of our respondents are non-developers with few or no programming skills. We consider that our sample is representative because it reflects the customers' specificities. We believe that the obtained scores translate into a positive signal about the UITD claims considering that the time spent in teaching all the UITD symbols was *very* short. One can only expect the scores to be higher if that training session had been longer, which is completely plausible in a professional environment.

With our first research question, we explored which UITD concepts require further explanation to be understood (those reported in Section 5.1). The results tell us that the concept that presented the greatest challenge to people was that transitions that are triggered from a *contained* interface can also be triggered from the *containing* interface. But the easiest concept is that "transitions that arise from the *containing* interface cannot be triggered from the *contained* one."

With our second research question, we investigate the correlation between the understanding of the UITD and the individual's programming skills. According to our results in Section 5.2, this correlation is weak. This means that it is not necessary to be skilled in programming to understand a UITD. So, given that the UITD provides a complete and precise specification of the interactions between system UIs and its users,[14] we believe that the UITD can effectively help in the communication between software developers and customers, which could indeed be very useful during the requirements elicitation and specification phases of a software development project as claimed in Gómez and Cervantes.[14]

Empirical evidence is one of the proposed approaches in software engineering to transfer results from academia to industry.[47] Based on the results, we show that practitioners could see the UITD as a tool with potential to be put into practice because it requires little effort to be understood by people, regardless of their programming skill level.

7 | DISCUSSION AND FUTURE WORK

Modeling notations that can fully represent the transitions between the UIs of a software system, such as CPN, BT, IFML, and UML SM, have high expressive power. This means that they can represent a wide range of ideas, but it also means that they have many symbols to be known by the notation user. There are studies that show that nonexperts, for example, business stakeholders, management, and send users, prefer a simple notation to better understand the model of a system.[12] The UITD has a low expressive power because it can only be used to model user–system interactions in full, but this gives it the advantage of having few symbols, making it an easy to learn simple notation.

The UITD symbol set only contains those symbols that a nonexpert strictly needs to be able to define the UI–user interactions. In practice, a subset of constructs from other complex languages may be used, as reported by Fernández-Sáez et al,[48] depending on the type of task. However, for the UI transitions modeling case, there are some constructs in the UITD that are not present in other languages. For example, one cannot express Figure 3 with a single construct in IFML because it does not have a specialized one to represent nested interfaces. So, one can argue that it is plausible to use the UITD language for this situation rather than redefining or extending other notations.

Beyond the results presented here, there are other reasons for practitioners to start using UITDs systematically. The need for a well-structured methodology for requirements elicitation, especially when defining user interface transitions, is recognized by Reggio et al[49] who propose a method for requirements specification in which use case descriptions follow a rigorous template aiming to improve the specification quality. They say that UI sketches (screen mock-ups) should be linked to the steps of the use case. In this way, comprehension of the functional requirements is improved. We think that the UITD can be naturally linked to this methodology because it shows the subsequent step in the specification, that is, it models the UI flow integrating all use cases.

As future work, other kinds of possible real-life users will be studied. It would also be interesting to study the limitations of the UITD, that is, to investigate if there is any kind of end-user application in which the user–system interactions cannot effectively be described with the UITD. Another step is to model systems in which real customers collaborate in the specification of interactive systems and provide feedback on the usefulness of the UITD. We are also working on the construction of a graphical tool for the edition of UITDs and the automatic code generation from them. A runnable skeleton of a web app will be generated from the UITD so that customers can see the modeled system in action. The missing details that cannot be expressed in a UITD would be left to be decided and implemented by developers, because these do not need to be discussed with the customer.

ENDNOTE

* http://148.206.168.145/UITD/what-is (contains the presentation and additional slides about the UITD editor)

DATA AVAILABILITY STATEMENT

The data that supports the findings of this study are available from the corresponding author upon reasonable request.

ORCID

Jorge Cervantes-Ojeda https://orcid.org/0000-0002-2267-7165
María Gómez-Fuentes https://orcid.org/0000-0003-0033-4476
Guillermo Chacón-Acosta https://orcid.org/0000-0002-7213-7386

REFERENCES

1. Bano M, Zowghi D. A systematic review on the relationship between user involvement and system success. *Inf Softw Technol*. 2015;58:148-169. doi:10.1016/j.infsof.2014.06.011
2. Mendez Fernandez D, Wagner S, Kalinowski M, et al. Naming the pain in requirements engineering. *Empir Software Eng*. 2017;22(5):2298-2338. PMID. https://link.springer.com/article/10.1007/s10664-016-9451-7
3. Ogunyemi AA, Lamas D, Lárusdóttir MK, Loizides F. A systematic mapping study of HCI practice research. *Int J Hum–Comput Int*. 2019;35(16):1461-1486. PMID. https://www.tandfonline.com/doi/abs/10.1080/10447318.2018.1541544?journalCode=hihc204
4. InVisionApp. Accessed 16 July 2019. https://www.invisionapp.com/
5. Sketch. Accessed 16 July 2019. https://www.sketch.com/
6. Constantine LL, Lockwood LAD. *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design*. Pearson Education; 1999.
7. IFML: interaction flow modeling language. Accessed 16 July 2019. https://www.ifml.org/
8. Kristensen LM, Christensen S, Jensen K. The practitioner's guide to coloured Petri nets. *Int J Softw Tool Technol Transf*. 1998;2(2):98-132. doi:10.1007/s10009-007-0038-x
9. Colvin, R, Hayes, IJ. A semantics for behavior trees. 2010, pp. 1-26. https://core.ac.uk/download/pdf/15090632.pdf
10. Dromey RG. Formalizing the transition from requirements to design, mathematical frameworks for component software – models for analysis and synthesis. In: He J, Liu Z, eds. *Series on Component-Based Development*. Singapore: World Scientific Publishing Co Pte Ltd; 2006:156-187. https://www.worldscientific.com/doi/abs/10.1142/9789812772831_0006
11. OMG: Object Management Group, UML Standard. The current UML specification. Accessed 16 July 2019. http://www.uml.org/#UML2.0
12. Van der Linden D, Hadar I, Zamansky A. What practitioners really want: requirements for visual notations in conceptual modeling. *Softw Syst Model*. 2019;18(3):1813-1831. doi:10.1007/s10270-018-0667-4
13. Randerson Queiroz, TC, Marques, AB. Using IFML for user interface modeling: an empirical study (S). Software Engineering and Knowledge Engineering (SEKE), 2018. doi:10.18293/seke2018-103
14. Gómez MC, Cervantes J. User interface transition diagrams for customer-developer communication improvement in software development projects. *J Syst Softw*. 2013;86(9):2394-2410. doi:10.1016/j.jss.2013.04.022
15. Gómez-Fuentes M. Cervantes-Ojeda J.: Application of user interface transition diagrams in the construction of a software system. 7th International Conference in Software Engineering Research and Innovation (CONISOFT), IEEE, October 2019, pp. 123-131. doi:10.1109/CONISOFT.2019.00026

16. Gómez-Fuentes, M. C., Cervantes-Ojeda J. Sequence diagrams tailored for software design used to build a carpooling management system. 7th International Conference in Software Engineering Research and Innovation (CONISOFT) IEEE, October 2019, pp. 116-122. doi:10.1109/CONISOFT.2019.00025
17. González-Pérez PP, del Carmen Gómez-Fuentes M, Velázquez JH. A hybrid expert system for the estimation of the environmental impact of urban development. *J Adv Math Comput Sci*. 2015;7(1):1-17. doi:10.9734/BJMCS/2015/15239
18. Abelein U, Paech B. State of practice of user-developer communication in large-scale IT projects. In: Salinesi C, van de Weerd I, eds. *International Working Conference on Requirements Engineering: Foundation for Software Quality*. Cham: Springer; 2014:95-111. https://link.springer.com/chapter/10.1007/978-3-319-05843-6_8
19. Ricca F, Scanniello G, Torchiano M, Reggio G, Astesiano E. Assessing the effect of screen mockups on the comprehension of functional requirements. *ACM Trans Softw Eng Methodol*. 2014;24(1):1-38. doi:10.1145/2629457
20. Nelson HJ, Poels G, Genero M, Piattini M. A conceptual modeling quality framework. *Softw Qual J*. 2012;20(1):201-228. PMID. https://link.springer.com/article/10.1007/s11219-011-9136-9
21. Aranda, J., Ernst, N., Horkoff, J., Easterbrook, S. A framework for empirical evaluation of model comprehensibility. In: International Workshop on Modeling in Software Engineering, IEEE, 2007, May, pp. 1-6. https://ieeexplore.ieee.org/abstract/document/4273247
22. Recker J, Reijers HA, van de Wouw SG. Process model comprehension: the effects of cognitive abilities, learning style, and strategy. *Commun Assoc Inf Syst*. 2014;34(1), Article 9. https://aisel.aisnet.org/cais/vol34/iss1/9/, doi:10.17705/1CAIS.03409
23. Polančič G, Gregor J. The impact of the representatives of three types of process modeling tools on modeler's perceptions and performance. *J Softw Evol Process*. 2016;28(1):27-56. doi:10.1002/smr.1764
24. Mendling J, Recker J, Reijers HA, Leopold H. An empirical review of the connection between model viewer characteristics and the comprehension of conceptual process models. *Inf Syst Front*. 2019;21(5):1111-1135. PMID. https://link.springer.com/article/10.1007/s10796-017-9823-6
25. Felderer M, Herrmann A. Comprehensibility of system models during test design: a controlled experiment comparing UML activity diagrams and state machines. *Softw Qual J*. 2019;27(1):125-147. PMID. https://link.springer.com/article/10.1007/s11219-018-9407-9
26. Hashim NL, Ibrahim HR, Rejab MM, Romli R, Mohd H. An empirical evaluation of Behavioral UML diagrams based on the comprehension of test case generation. *Adv Sci Lett*. 2018;24(10):7257-7262. https://www.ingentaconnect.com/contentone/asp/asl/2018/00000024/00000010/art00054, doi:10.1166/asl.2018.12924
27. Liebel, G., Tichy, M. Comparing comprehensibility of modeling languages for specifying behavioral requirements. In HuFaMo@ MoDELS, 2015, pp. 17-24. https://pdfs.semanticscholar.org/0879/92f6a4d8a4d83a5988cadcfe6123f1f2f958.pdf
28. Scanniello G, Gravino C, Genero M, et al. Do software models based on the UML aid in source-code comprehensibility? Aggregating evidence from 12 controlled experiments. *Emp Softw Eng*. 2018;23(5):2695-2733. doi:10.1007/s10664-017-9591-4
29. Cruz-Lemus JA, Genero M, Manso ME, Morasca S, Piattini M. Assessing the understandability of UML statechart diagrams with composite states—a family of empirical studies. *Emp Softw Eng*. 2009;14(6):685-719. https://link.springer.com/article/10.1007%2Fs10664-009-9106-z, doi:10.1007/s10664-009-9106-z
30. Wohlin C, Runeson P, Höst M, Ohlsson MC, Regnell B, Wesslén A. *Experimentation in Software Engineering*. 1st ed. Springer Science & Business Media; 2012. doi:10.1007/978-3-642-29044-2
31. Franken S, Kolvenbach S, Prinz W, Alvertis I, Koussouris S. CloudTeams: bridging the gap between developers and customers during software development processes. *Proced Comput Sci*. 2015;68:188-195. https://www.sciencedirect.com/science/article/pii/S1877050915030793, doi:10.1016/j.procs.2015.09.234
32. Moody DL. The "physics" of notations: towards a scientific basis for constructing visual notations in software engineering. *IEEE Trans Softw Eng*. 2009;35(6):756-779 https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=5353439, doi:10.1109/TSE.2009.67
33. Cervantes-Ojeda J, Badillo-Salas A, Gómez-Fuentes M.C., 'Specialized tool for editing user interface transitions diagrams (UITD)', 9th International Conference in Software Engineering Research and Innovation (CONISOFT'21), San Diego U.S.A., October 23-25, 2021, pp. 10-16. doi:10.1109/CONISOFT52520.2021.00014
34. Van Der Linden D, Hadar I. A systematic literature review of applications of the physics of notation. *IEEE Trans Softw Eng*. 2019;45(8):736-759 https://ieeexplore.ieee.org/abstract/document/8283537, doi:10.1109/TSE.2018.2802910
35. Krogstie J. Specializations of SEQUAL. In: Krogstie J, ed. *Model-Based Development and Evolution of Information Systems*. London: Springer; 2012. doi:10.1007/978-1-4471-2936-3_6
36. Schwabe, D.: 'A conference review system. Case study for the International Workshop on Web Oriented Software Technology'. Valencia, 2001. Accessed 16 July 2019. http://users.dsic.upv.es/~west/iwwost01/files/ConferenceReviewSystem.pdf
37. Vannete, D.: 'Using attention checks in your surveys may harm data quality', 2017. Accessed 16 July 2019. https://www.qualtrics.com/blog/author/dave-vannette/
38. Rosenberg J. Statistical methods and measurement. In: Singer J, ed. *Guide to Advanced Empirical Software Engineering*. London: Springer; 2008:155-184. doi:10.1007/978-1-84800-044-5_6
39. Kutner MH, Nachtsheim CJ, Neter J. *Applied Linear Regression Models*. 4th ed. McGraw Hill; 2004.
40. Shapiro SS, Wilk MB. An analysis of variance test for normality (complete samples). *Biometrika*. 1965;52(3-4):591-611. doi:10.1093/biomet/52.3-4.591
41. Higgins JJ. *Introduction to Modern Nonparametric Statistics*. 1st ed. Duxbury Press; 2003.
42. Akoglu H. User's guide to correlation coefficients. *Turkish J Emerg Med*. 2018;18(3):91-93. https://www.sciencedirect.com/science/article/pii/S2452247318302164, doi:10.1016/j.tjem.2018.08.001
43. Hinkle DE, Wiersma W, Jurs SG. *Applied Statistics for the Behavioral Sciences*. 5th ed. Boston: Houghton Mifflin; 2003.
44. Rumsey DJ. How to interpret a correlation coefficient. In: Rumsey D, ed. *Statistics for Dummies*. 2nd ed. John Wiley; 2016. https://www.dummies.com/education/math/statistics/how-to-interpret-a-correlation-coefficient-r/
45. Faul F, Erdfelder E, Lang AG, Buchner A. G* Power 3: a flexible statistical power analysis program for the social, behavioral, and biomedical sciences. *Behav Res Methods*. 2007;39(2):175-191. https://link.springer.com/article/10.3758%2FBF03193146, doi:10.3758/BF03193146

46. Petersen K, Gencel C. Worldviews, research methods, and their relationship to validity in empirical software engineering research. In: 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement IEEE; 2013:81-89. https://ieeexplore.ieee.org/abstract/document/6693226
47. Brings J, Daun M, Brinckmann S, Keller K, Weyer T. Approaches, success factors, and barriers for technology transfer in software engineering—results of a systematic literature review. *J Softw Evol Process*. 2018;30(11):e1981. doi:10.1002/smr.1981
48. Fernández-Sáez AM, Chaudron MR, Genero M. An industrial case study on the use of UML in software maintenance and its perceived benefits and hurdles. *Emp Softw Eng*. 2018;23(6):3281-3345. doi:10.1007/s10664-018-9599-4
49. Reggio G, Leotta M, Ricca F, Clerissi D. DUSM: a method for requirements specification and refinement based on disciplined use cases and screen mockups. *J Comput Sci Technol*. 2018;33(5):918-939. PMID. https://link.springer.com/article/10.1007/s11390-018-1866-8

How to cite this article: Cervantes-Ojeda J, Gómez-Fuentes M, Chacón-Acosta G. Can non-developers learn a simplified modeling notation quickly? *J Softw Evol Proc*. 2022;e2481. doi:10.1002/smr.2481