Article

Domain Usability Evaluation

Michaela Bačíková *, Jaroslav Porubän, Matúš Sulír, Sergej Chodarev, William Steingartner and Matej Madeja

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice, Letná 9, 042 00 Košice, Slovakia; jaroslav.poruban@tuke.sk (J.P.); matus.sulir@tuke.sk (M.S.); sergej.chodarev@tuke.sk (S.C.); william.steingartner@tuke.sk (W.S.); info@madeja.sk (M.M.)
* Correspondence: michaela.bacikova@tuke.sk

Abstract: Contemporary software systems focus on usability and accessibility from the point of view of effectiveness and ergonomics. However, the correct usage of the domain dictionary and the description of domain relations and properties via their user interfaces are often neglected. We use the term *domain usability (DU)* to describe the aspects of the user interface related to the terminology and domain. Our experience showed that poor domain usability reduces the memorability and effectiveness of user interfaces. To address this problem, we describe a method called *ADUE (Automatic Domain Usability Evaluation)* for the automated evaluation of selected DU properties on existing user interfaces. As a prerequisite to the method, metrics for formal evaluation of domain usability, a form stereotype recognition algorithm, and general application terms filtering algorithm have been proposed. We executed ADUE on several real-world Java applications and report our findings. We also provide proposals to modify existing manual usability evaluation techniques for the purpose of domain usability evaluation.

Keywords: human–computer interaction; user experience; usability evaluation methods; domain usability; domain-specific languages; graphical user interfaces

check for updates

Citation: Bačíková, M.; Porubän, J.; Sulír, M.; Chodarev, S.; Steingartner, W.; Madeja, M. Domain Usability Evaluation. *Electronics* **2021**, *10*, 1963. https://doi.org/10.3390/electronics10161963

Academic Editor: George A. Tsihrintzis

Received: 19 July 2021
Accepted: 10 August 2021
Published: 15 August 2021

1. Introduction

User experience (UX) and usability is already ingrained in our everyday lives. Nielsen's concept of "usability engineering" [1] and Norman's [2] practical user interface (UI) design has become an inseparable part of design policies in many large companies, setting an example to the UX field throughout the world. Corporations such as Apple, Google, Amazon, and Facebook realized that when designing UIs, it is not only about how pretty the UI looks like, but from a long-time perspective, usability and UX bring economic benefits over competitors. Usability and UX are related to many aspects of the design, including consistency, efficiency, error rate, learnability, ease of use, utility, credibility, accessibility, desirability, and many more [1–5].

However, when analyzing common UIs of medium and small companies, we still find such UIs that are developed with respect to the practical usability and UX but not to the user's domain. From our experience, such cases are very common. The situation has slowly slowly become better with the introduction of UX courses into the curricula of universities and with the foundation of UX organizations spreading the word. The more specific the domain, the more evident is the problem of designs focused on usability that neglects the domain aspect. This fact has been identified by multiple researchers around the globe [6–9].

1.1. Domain Usability

We describe *Domain Usability (DU)* in terms of five UI aspects: domain content, consistency, world language, an adequate level of specificity, language barriers, and errors. For the purpose of clarity, we will present the full definition [10] of all five aspects here:

- *Domain content*: the interface terms, relations, and processes should match the ones from the domain for which the user interface is designed.
- *Consistency*: words used throughout the whole interface should not differ—if they describe the same functionality, the dictionary should be consistent.
- *The language used in the interface*: the language of the interface should be the language of the user, and the particular localization of the UI should be complete, i.e., there should be no foreign words.
- *Domain specificity*: the interface should not contain too general terms, even if they belong to the target domain. The used terms should be as specific as possible.
- *Language barriers and errors*: the interface should not create language barriers for the users, and it should not contain language errors.

Domain usability is not a separate aspect of each UI. On the contrary, it is a part of the general usability property. The *overall usability* is defined as a *combination* of ergonomic and domain usability. Successful completion of a task in a UI is affected by both ergonomic and domain factors:

- *Ergonomic aspect*: without the proper component placement, design, and ergonomic control, it is not possible to perform tasks effectively.
- *Domain aspect*: without the proper terminology, it is harder (or not possible at all) to identify the particular features needed to complete the chosen task. This results in total prevention of the task or at the very least, less effective user performance and lower memorability.

As we described in our previous works, all aspects of the overall usability (as defined by Nielsen [1]) are affected by DU. For more details on the definition of DU, we encourage the reader to see our earlier work [11].

1.2. Problem and Motivation

To summarize our knowledge, we identified the main issues in this area as follows:

(i) There are no clear *rules* to design the term structure of an application, so it would correspond with the domain.
(ii) There are no official *guidelines* explicitly describing UIs that should match the real world or map domain terms and processes. References can be found in the literature [12,13], but they are either too general or not focused on domain usability as a whole.
(iii) The variety of human thinking, *ambiguity, and diversity of natural language* represents an issue in evaluating the correctness of UI terminology.
(iv) No clear *manual methods* exist for the formal DU evaluation of existing UIs.
(v) There are no standardized *metrics* to evaluate domain usability.
(vi) No comprehensive *automated methods* exist for domain usability evaluation. Automated and semi-automated methods were devised only for the evaluation of usability in general (e.g., SYNOP [14], AIDE [15]; see the broader reviews [16–18] and the reviews about web usability evaluation [19,20]).

We have addressed issues (i) to (v) and also partially (vi) in our previous works:

- We introduced the concept of *DU* and examples to illustrate our definition [11,21]: (i), (ii).
- To address (iii), we performed a feasibility analysis of approaches for analyzing separate DU aspects [22].
- We proposed and experimentally verified multiple novel manual techniques for DU evaluation [10,23] (iv).
- We designed a domain usability metric consisting of five aspects [24] (v).
- We proposed a conceptual design and a proof-of-concept implementation of a method for automated evaluation of DU [22] and later presented its preliminary evaluation [25] (vi).

The *main contribution* of this paper is concerning issue (vi), in which we focus on automated evaluation. We would like to summarize and put into context our existing findings in this area and to describe the final design, implementation, and validation of this method. The novel additions and improvements include but are not limited to the General Application Terms Ontology (Section 4.2), the Form Stereotype Recognition algorithm (Section 4.3), the computation and display of the Domain Usability Score in the ADUE tool (in Section 6), and detailed presentation of the evaluation results on real-world applications (in Section 7). As a secondary contribution, we will propose modifications of existing general usability evaluation techniques to make them suitable for domain usability evaluation (Section 8).

1.3. Paper Structure

In Section 2, we introduce our DU metrics that can be used for formal DU evaluation. The metrics were used to calculate the DU score in our automated evaluation approach.

In Sections 3–5, we explain the design of our automated approach to DU evaluation. First, we explain the concept (Section 3), then describe the prerequisites needed for the approach to work (Section 4), and then we describe the method itself (Section 5). To verify the approach and show its viability, we implemented its prototype (Section 6) and used it to analyze multiple open-source applications (Section 7).

We summarize both manual and automated techniques of usability evaluation in Section 8, and for some of them, we comment on their potential to evaluate DU. Section 9 represents related work focused on DU and its references in the literature.

2. Domain Usability Metrics Design

As we have mentioned, DU is defined by five main aspects. In our previous research [24], we tried to determine whether all DU aspects impact the usability equally. Several preliminary experiments we performed in the domain of gospel music suggested the invalidity of this hypothesis [10,23]; e.g., consistency issues had a stronger impact on usability than language errors.

We decided to conduct two surveys [24] to evaluate the effect of five DU aspects on DU. Using the results of the surveys, we designed a metric for formal evaluation of DU. The metrics can be used in manual or automatized evaluation to represent *formal measurement of target UI's DU*. Next, we will explain the design of the DU metrics.

To formally measure the target UI's DU, we first determine the number of all user interface components containing textual data or icons. Next, we analyze the components to find out which of them have DU issues. Since we have the number of all terms n and the number of DU issues, we can compute the percentage of the UI's correctness, where 100% represents the highest possible DU and 0% is the lowest one. Note that each component can have multiple issues at the same time (e.g., an incorrect term and a grammar error). If all UI components had multiple issues, the result would be lower than zero, so it is necessary to limit the minimum value. Given that each DU aspect has a different weight, we defined the formula to measure DU as follows:

$$du = \max\left(0,\ 100\left(1 - \frac{e}{n}\right)\right) \tag{1}$$

where e is calculated as:

$$e = w_{dc}n_{dc} + w_{ds}n_{ds} + w_c n_c + w_{eb}n_{eb} + w_l n_l \tag{2}$$

Coefficients w_x (where x stands for dc, ds, c, eb or l) are the weights of particular DU aspects as follows:

- n_{dc}—the number of domain content issues,
- n_{ds}—the count of domain specificity issues,
- n_c—the number of consistency issues,
- n_{eb}—the count of language errors and barriers,

- n_l—the number of world language issues.

The weights w_x were determined by performing two surveys, first a general one with 73 respondents aged between 17 and 44 years and then a domain-specific one with 26 gospel singers and guitar players aged between 15 and 44 years. The general group consisted of general computer users, and the domain-specific group was selected from the participants of previous DU experimentation with manual DU evaluation techniques [10,23], as they experienced DU issues first-hand.

The questionnaires consisted of two parts. The first part contained five DU aspects represented by visual examples—screenshots from domain-specific UIs. To ensure that participants understood the issues, supplementary textual explanations were provided. The task of the participants was to study the provided examples and rate the importance of a particular DU aspect using a number from the Likert scale [26] with a range from 1 to 5 (1 being the least important).

In the second part, the task was to order the five aspects of DU from the least to the most important. The questionnaires given to the general and domain-specific group can be found at: http://hornad.fei.tuke.sk/~bacikova/domain-usability/surveys (accessed on 9 August 2021). Details about the surveys can be found in [24].

We merged the results of the first (rating) and second (ordering) part of the domain-specific questionnaire and computed the weight of each aspect. Therefore, we can substitute the weights w_x (where $x \in \{dc, ds, c, eb, l\}$) in Equation (2):

$$e = 2.9\, n_{dc} + 2.6\, n_{ds} + 2.6\, n_c + 1.7\, n_{eb} + 1.54\, n_l \tag{3}$$

Equation (1) then represents *the metric of DU considering its aspects*, with the result as a percentage. To interpret the results, evaluators can follow Table 1. The interpretation corresponds to the scale on which the participants rated the particular aspects in the surveys.

Table 1. Interpretation of the rating computed via the proposed DU metric.

Rating	Interpretation
$100 \geq du \geq 90\%$	Excellent
$90 > du \geq 80\%$	Very good
$80 > du \geq 70\%$	Good
$70 > du \geq 55\%$	Satisfactory
less than 55%	Insufficient

3. Automatic Evaluation of Domain Usability Aspects

In this section, we will analyze the boundaries of DU evaluation automation and the possibilities related to individual DU aspects. We explain the design of an automated approach to DU evaluation at a high level of abstraction.

3.1. Domain Content and Specificity

Domain content and specificity are the most difficult aspects to evaluate in a domain-specific application. Since a particular UI is usually designed for a domain expert, the domain content in the UI must be specific for the particular domain. Because of the ambiguity of natural language, the line determining whether a given word pertains to a particular domain or not may be very thin. We admit that evaluation performed by a domain expert should be considered the most appropriate in such cases. However, when no expert is available or when first UI prototypes are going to be evaluated, automated evaluation might be a helpful, fast, and cheap way to remove issues in the early stages. We will try to outline the situation in which such an automated evaluation would be utilized.

Imagine we have an existing user interface that has been used in some specific domain for ages. However, although this UI is usable, the used technology had become obsolete. The time has come to develop and deploy a new application version. The technologies will change, but for the domain users to accept the new UI, at least the terminology should be consistent with the previous version. However, testing the whole UI for domain-related content manually is a time-consuming, attention-demanding, and tiresome task. It would be helpful to have an automated way to compare both UIs. Suppose there is a way of extracting the terminology of both UIs into a formal form (e.g., an ontology). Then it would be possible to compare the results using a comparator tool. The result of the comparison would show the following:

- Any *new terms* are marked in the new UI ontology so that they can be checked by a domain expert.
- *Renamed* terms are marked for the same reason. We identify renamed items based on the representing component and its location in the component hierarchy.
- If the terms (UI components) were *moved*, then they are checked for consistency of their inclusion into the new group of terms (term hierarchy).
- *Removed terms* are marked in the old UI ontology because the domain experts, customers, or designers/developers should check whether their removal is reasonable.
- All terms (i.e., their representing components) that have undergone an *illogical change* are marked as a *usability issue*.

Illogical changes are the following: (i) from text input component (e.g., text boxes and text areas) to descriptional component (e.g., labels) and vice versa, (ii) from textual to functional (e.g., buttons and menu items) and vice versa, (iii) from functional to descriptional component and vice versa, and (iv) from grouping (containers, button groups, etc.) to other types of components and vice versa. For example, the term "Analyze results" which, in the old UI, was represented by a button, but in the new UI, it is a label—i.e., the representing component changed its type from functional to descriptional. When checking the mentioned type changes, we can confirm the term against its representing component in the old and new UI version.

The scenario described above is rather specific for situations in which there are two versions of the particular UI—whether it is an old UI and a new one, or two separate UIs from the same domain are developed by different vendors. However, when the UI is freshly designed specifically for the particular business area, there is usually only one UI available. In this case, some other source of ontological information is needed, which may be:

- a reference ontology modeling the specific domain and its language,
- generic ontological dictionaries or other sources of linguistic relations, such as web search.

In these cases, the feasibility of analysis strongly depends on the reference resources. The disadvantage of the first option is the necessity of the reference ontology, which would have to be created manually by the domain expert. On the other hand, such a manually created ontology would be of higher quality than an automatically generated one, presumably having defined all necessary domain objects, properties, and relations. Thus, it would be easier to check the correctness of the target UI than by applying the approach as with two UIs, since it is usually not possible to extract 100% of data from both UIs.

As for ontological dictionaries or web search, again, the analysis strongly depends on the resources. Current ontological dictionaries are quite good, but their size is limited and their ontologies are not very usable in any specific domain. It would be best to have a domain-specific ontological dictionary, but because we assume that in the future, domain-specific ontologies [27] would grow in both size and quality, and the approach proposed here will be applicable with greater value.

Current technologies and resources allow us only to use general ontologies to check *hierarchies of terms for linguistic relations using natural language processing*. Let us take an example of a *Person* form. The form has a list of check-box buttons for selecting a favorite color with values *red*, *yellow*, *blue*, and *green*. The task is to check whether the parent–child relation between the *Favorite color* term and individual color values is correct (Listing 1).

Listing 1. Hierarchy of terms for selecting favorite color in the domain dictionary of the Person form.

```
favoriteColor {children}: [
    red
    yellow
    blue
    green
]
```

From the linguistic point of view, *Favorite color* is a hypernym of the individual color values (or conversely, the latter are hyponyms of the *Favorite color*). Similar relations are *holonymy* and *meronymy* which represent a "part" or "member" relationship.

Suppose that we know and can automatically determine the hierarchy of terms in the UI (we know that components labeled by the color names are hierarchically child components of the container labeled by the term *Favorite color*), we can check if these linguistic relations exist between the parent term and its child terms.

Existing available ontological dictionaries (such as WordNet) usually provide a word-attribute relation of words including linguistic relations, such as hyponymy and holonymy. In the domain analysis process, all children and parents should be checked from the linguistic point of view, but mainly enumerations and button groups or menu items because they are designed with the "grouping" relation in mind. The same can be achieved by using web search instead of ontological dictionaries (more on using web search in Section 5.2).

As the last process of checking UI domain content, we propose to check the presence of *tooltips*. A tooltip is a small description of a graphical component, which explains its functionality or purpose. Tooltips are displayed after a short time when the mouse cursor position is over the component. Many times, tooltips are not necessary for general-purpose components, e.g., the OK, Cancel, Close, or Reset buttons. However, they can be extremely important for explaining the purpose of *domain-specific* functional components (components performing domain-specific operations) or when the description would take too much space when putting it on the component's label. Our experiment with open-source applications [25] showed that developers almost never use tooltips for functional components, even in cases when their label is not quite understandable even for domain-specific users. The common cases are acronyms and abbreviations used when the full name or description of the domain operation would take too much space on the display.

3.2. Consistency

All domain terminology should be checked for consistency and, thus, marked for checking. We can search for equal terms with case inconsistencies (Name-NAME-naMe) and/or similar terms (Cancel, Canceled) and their case inconsistencies.

Note: currently, it is not possible to automatically evaluate the so-called *feature consistency*, i.e., whether the same functionality is represented by the same term. The reason is the inability of current technologies to make this information available programmatically.

3.3. Language Barriers and Errors

Language errors and the completeness of language alternatives can be checked using standard spell-checking methods. For example, dictionaries (e.g., bundled with open-source text editors such as OpenOffice) may be leveraged to mark all incorrect and untranslated words similarly to spell checking in modern textual editors.

4. Prerequisites

In order to analyze the domain dictionary in any application, the means of extracting that dictionary into a formal form is necessary. For this extraction, we can use the DEAL (Domain Extraction ALgorithm) method described in [28,29].

In this section, we will describe the DEAL tool needed for extracting domain information from existing user interfaces. We also describe the design and implementation of supplementary algorithms that we implemented into DEAL to be able to focus on DU issues, namely:

(a) General Application Terms Ontology—serves for filtering out non-domain related terms from the user interface,
(b) Form Stereotype Recognizer—an algorithm making the analysis of forms more effective.

4.1. DEAL Method

DEAL (Domain Extraction ALgorithm) (https://git.kpi.fei.tuke.sk/michaela.bacikova/DEAL; accessed on 9 August 2021) is a method for extracting domain information from user interfaces of applications. Its implementation currently supports Java (Swing), HTML, and Windows applications (*.exe). The Windows application analyzer utilizes the output of Ranorex Spy (https://www.ranorex.com/help/latest/ranorex-studio-advanced/ranorex-spy/introduction/; accessed on 9 August 2021), which means it supports programs that are analyzable by Ranorex. The list of supported components is located at https://git.kpi.fei.tuke.sk/michaela.bacikova/DEAL/-/wikis/analyzing-windows-applications (accessed on 9 August 2021).

Except for the part of loading the input application, the whole process is fully automatized and takes place in two phases: *Extraction* and *Simplification*. The result of the *Extraction* phase is a domain model in the form of a graph. Nodes of the graph correspond to terms (concepts) of the analyzed user interface. Each such node contains information about:

- UI *component* that represents the term in the user interface;
- *name*—the label displayed on the component;
- *description*—the component's tooltip if it is present;
- *icon* (if present);
- *category of the component*—either functional, informative, textual, grouping (container), or custom;
- *type* of input data—in the case of input components, the type can be string, number, date, boolean, or enumeration;
- *relation* to other terms—mutual (non-)exclusivity;
- *parent* term (usually corresponds to lexical relation of hypernymy or holonymy);
- *child* terms (usually correspond to hyponyms or meronyms).

The extraction is followed by the *Simplification* phase, where structural components without domain information (e.g., panels and containers) are filtered out unless they are necessary to maintain the term hierarchy.

Properties of the terms and their hierarchy are used to check for the missing domain information in order to identify incorrect or missing data types and lexical relations between terms such as hyponymy, hypernymy, holonymy, and meronymy.

For example, let us have a form for entering the person's data such as name, surname, date of birth, marital status, or favorite color. The *Person* dialog contains the fields for entering the data. The resulting domain model can be seen in Listing 2. It contains the term *Person* with the child nodes corresponding to fields of the of form. The *status* term has the *enumeration* type with mutually exclusive values because in the UI it contains multiple options as radio buttons. The *favorite color*, on the other hand, uses check-box components, so the corresponding term contains *child terms* with all offered values, and they are not mutually exclusive. *Person* term also contains children corresponding to functional components, e.g., menu items or buttons (such as *OK* or *Close*). A similar graph of terms is created for every window in the user interface.

DEAL is able to export this hierarchy into the standard OWL ontological format.

Listing 2. The domain model of the Person form.

```
domain: 'Person' {children}: [
   'Name' {string}
   'Surname' {string}
   'Date of birth' {date}
   'Status' {mutually-exclusive}
       {enumeration}[
          'Single'
          'Married'
          'Divorced'
          'Widowed'
       ]
   'Favorite color' {mutually-not-exclusive}
       {children}: [
      'red'
      'yellow'
      'blue'
      'green'
   ]
   'OK'
   'Close'
   'Reset'
]
```

4.2. General Application Terms Ontology

In the Person form example in Listing 2, we have three terms (represented by three buttons) not related to the domain of Persons. If we are to analyze *domain* objects, properties, and relations, we need to filter out any terms potentially unrelated to the domain. To do so, we will use a new reference ontology that will list domain-independent general-purpose terms commonly used in applications, their alternatives, and their forms.

We built this ontology manually by analyzing 30 open-source Java applications from SourceForge, 4 operating systems and their applications (system applications, file managers, etc.), and 5 software systems from the domain of integrated development environments (IDEs). The specific domain of IDEs was selected to observe and compare the occurrence of domain-specific versus general application terms. We listed and counted the occurrence of all terms in all analyzed UIs. Then, we selected only those that had an occurrence rate over 50%.

The list of the most common terms can be seen in Table 2 (the *General Application Terms Ontology* can be found at https://bit.ly/2R6bm6p; accessed on 9 August 2021). According to this ontology, we implemented an additional module into DEAL, which is able to automatically filter out such terms from the domain model immediately after the domain model Extraction and Simplification phase and prior to the DU evaluation process.

The analysis of application terms in a specific domain showed that the domain-specific terminology is more common in a specific domain than general application terms.

4.3. Recognizing Form Stereotypes

Another drawback of the DEAL method is its insufficient form analysis. In more than 50 open-source applications we have analyzed, the most common problem were the missing references between the actual form data components (i.e., text fields) and their textual labels readable in the UI. Such a missing reference causes a component to be extracted without any label or description and, therefore, has no term to be represented by. As a result, it is filtered out in the DEAL's domain model Simplification phase as a component with no domain-specific content and is therefore excluded from the consecutive analyses.

Table 2. List of the most frequently occurring terms in UIs (the vertical bar character '|' denotes alternatives).

Term	Occurrence	Most Common UI Element
About|Credits	90%	Menu item
Apply	87%	Button
Cancel	97%	Button
Close|Exit|Quit	100%	Button|Menu item
Copy	70%	Menu item
Cut	70%	Menu item
Edit	70%	Menu
File	90%	Menu
Help	80%	Menu|Menu item
New	90%	Menu item
OK	97%	Button
Open	83%	Button|Menu item
Paste	70%	Menu item
Plug-ins|Extensions	40%	Menu|Menu item
Preferences|Settings	60%	Menu|Menu item
Redo	83%	Menu item
Save	83%	Button|Menu item
Save as	83%	Menu item
Tools	53%	Menu
Undo	83%	Menu item
View	63%	Menu
Window	70%	Menu

However, such components are necessary to determine the data type of their input values, which is reflected in the domain model. For example, in Listing 2, *name* is of data type *string* and *dateOfBirth* is of data type *date*.

For the developers of component-based applications, it is usually possible to set a "labelFor" (Java) or "for" (HTML) attribute of the label component (from this point, we will refer to this attribute as to *labelFor*). However, since this attribute is not mandatory in most programming languages, the result is usually a large number of components with no label assigned.

To solve this issue, we designed a *Form Stereotype Recognition (FSR) algorithm* to recognize form stereotypes in target UIs and implemented it into the DEAL tool.

Prior to the implementation, we manually analyzed the source code of existing user interfaces for the most common *form stereotypes*. We selected web applications instead of desktop ones for better accessibility and higher occurrence of forms. Thirty web applications were analyzed, and we focused on registration forms, login forms, and their client applications. Based on the analyzed data we identified the five most common form stereotypes shown in Figure 1.

1. LEFT—most common, the text labels are located left to the form component.
2. ADDITIONAL RIGHT—similar to LEFT, but some form components have additional information added to the right of the component, e.g., validation messages.
3. ABOVE—labels are located above the form components.
4. ADDITIONAL BELOW—sometimes the cases with additional information occur under the particular form component. Usually, it is a text for showing another application window, in which the particular item is further explained, or it is a link with which the users are sent an email with new password activation in case of forgetting the old one.
5. PLACEHOLDER—labels are located inside the designated form component. In HTML, this property is called a *placeholder*. This stereotype is becoming more and more

common in modern web applications, although it is marked as less usable. In this case, there is rarely any other label around the form component.

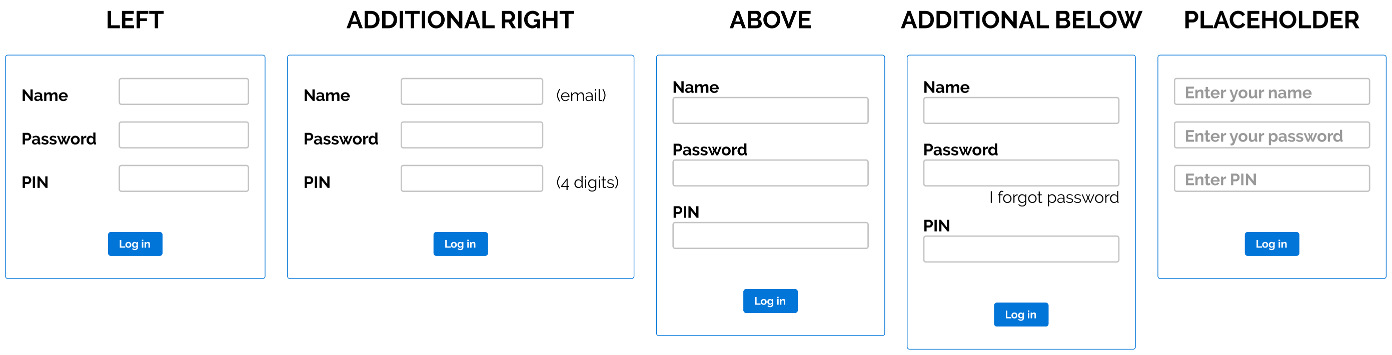


Figure 1. The most frequent form stereotypes.

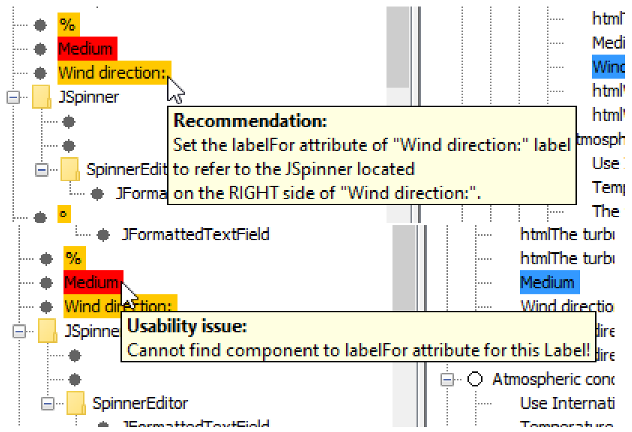
The FSR algorithm analyzes these form stereotypes in the target UI, and based on the identified stereotype, it assigns a label to each form data component. In short, the main principle of the FSR algorithm is to find all form components around each particular label in a form container. Then for all labels (excluding the ones that have the *labelFor* attribute set), the FSR counts the number of components around them as displayed in the UI. The resulting form stereotype is the direction in which the largest number of form components is located relative to each label. If there is no explicit maximum (e.g., five components have labels on their left and five other components have labels on their right), then the form stereotype cannot be identified and is marked as MIXED.

The targets of the FSR algorithm are common form components, namely:

- a descriptional text component (label),
- textual components (input fields, text fields, text areas, password fields, etc.),
- switches (radio buttons, checkboxes),
- spinners,
- tables,
- lists and combo-boxes.

If the target container was identified as a form stereotype, FSR pairs the form components with their labels by defining their *labelFor* attribute. This step also enables us to mark all form components that have no automatically assignable label and represent them as *recommendations* for correction to the user. If there is any label that has no stereotype, then it is considered a *usability issue*, and a recommendation for assigning a label to the most probable component (closest according to one of the possible stereotypes) is displayed. An example of both issues can be seen in Figure 2 extracted from the OpenRocket (https://sourceforge.net/projects/openrocket/; accessed on 9 August 2021) user interface.

By using the FSR algorithm, we were able to successfully recognize the correct stereotypes of most of the tested form components.



Recommendation:
Set the labelFor attribute of "Wind direction:" label to refer to the JSpinner located on the RIGHT side of "Wind direction:".

Usability issue:
Cannot find component to labelFor attribute for this Label!

Figure 2. DEAL—Example of a recommendation indicating the successful recognition of a form stereotype and an issue because of a missing *labelFor* attribute. The domain model shown in this figure was extracted from OpenRocket.

5. ADUE Method

The ADUE method uses the techniques mentioned in Sections 3 and 4. To sum up, we propose the following approaches to the automatized analysis of DU:

- Ontological analysis with two ontologies (Section 5.1),
- Specificity evaluation by analyzing the term hierarchies using ontological dictionaries or a web search (Section 5.2),
- Grammar evaluation by searching for grammar errors and typos using an existing linguistic dictionary of the target language (Section 5.3),
- Analysis of form components and their labels based on the form stereotype recognition method (Section 4.3)
- Tooltip analysis (Section 5.4).

In the next subsections, we describe each of the methods in more detail (except the form analyzer that was already explained in Section 4.3). We use example applications to explain each approach and show the identification of usability issues and recommendations for fixing them.

5.1. Ontological Analysis

As mentioned in Section 3.1, the first option is to use two ontologies extracted from new and old application versions. In case there is only one ontology, only specificity (Section 5.2) and grammar evaluation (Section 5.3) are executed for this ontology. If there are two ontologies, both specificity and grammar evaluations are performed on the newer one along with ontological comparison. Now we will describe the ontological comparison approach.

The process is depicted in Figure 3. For technological reasons, DEAL is able to run only one application at a time; therefore the ontology extraction happens in two steps. First, we use the DEAL tool to extract domain information from the first application without any DU analysis, and export it into an ontological format (the top-left part of Figure 3). Then, we run DEAL again with the new application version (the top-right part), import the previously extracted ontology, and run the ADUE comparison and evaluation algorithm (the bottom part of Figure 3). The ontology evaluation results are then displayed to the user.

Each item in an extracted ontology represents one component of the UI, and it contains:

- The term's *text* representation. A term has such a text representation only if its representing component has a description in the form of a label or a tooltip.
- *ID* of the representing component. This is mainly because of the ontology format, where every item has to have an identifier. We used the text attribute as an identifier and added numbering to ensure uniqueness.
- The *class* of the component: a button, label, text field, check box, radio button, etc.

- The term's *parent* term.
- *Children*, i.e., the child terms.

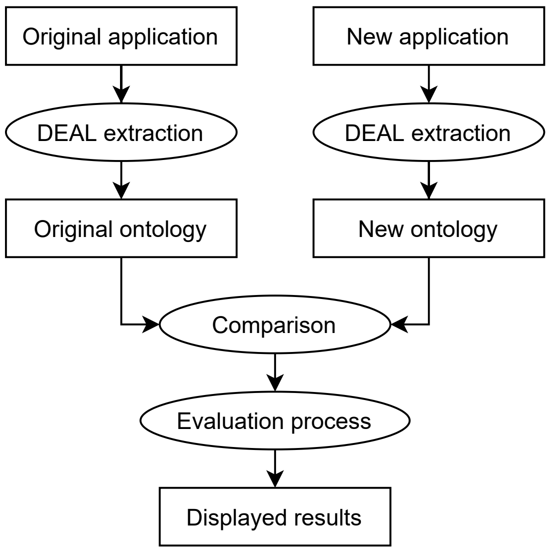


Figure 3. ADUE method—a high-level overview of the ontological evaluation process with two ontology versions. Processes are marked as ellipses, data as rectangles.

The algorithm compares the original ontology with the new one, searching for new, deleted, changed, and retained elements. We consider two elements equal if all their attributes (text, ID, class, parent, children) are equal. As a part of the evaluation, we consider the impact of the changes as follows:

- New elements—we do not consider newly added elements an issue. It is common that as user interfaces evolve in time, they get new and new features. However, the ADUE user should know about these changes to be able to check their correctness.
- Removed elements—these might or might not introduce an issue and feature depletion, depending on the evaluator whether the removal was justified.
- Changed elements—we consider *correctly* and *incorrectly* changed elements; incorrect changes are considered a usability issue. Incorrect changes include the illogical component type of changes described in Section 3.1.

The whole process is noted as the "Evaluation process" in Figure 3.

All results are stored in a list and then displayed to the evaluator in the UI. There, the user can see a list of all terms in the application. After selecting a specific term, details about the changes between the old and new ontology versions are shown, along with an error or a warning in case a potential issue was found.

After the comparison, *specificity evaluation* (Section 5.2) and *grammar evaluation* (Section 5.3) are performed on the new ontology version.

5.2. Specificity Evaluation

The goal of the *specificity evaluation* is to linguistically verify hierarchical relations found in the user interface. It uses ontological dictionaries and web search as a source of linguistic relations.

The algorithm traverses all grouping elements in the domain model graph. For each group, it selects the names of child terms and creates a *child word set*. From each child word set, we remove all forms of reflexive pronouns and auxiliary verbs (is, are, have, etc.) to get more precise results. The algorithm also uses natural language processing to recognize the word class of each word and keeps only nouns, verbs, and adjectives.

We use the *Ontological Dictionaries and Google Search evaluation algorithm (OD&GS)* to get a list of the *most probable parent terms* (hypernyms or holonyms) for each child word set.

The algorithm combines three sources: WordNet, Urban Dictionary, and Google web search. To optimize the results, it defines the following order in which the sources are utilized:

1. If any word from the input term set is a number, Google search is used first because it is optimal for numeric values.
2. In other cases, WordNet is used first since it is effective and available without restrictions.
3. If the probability of the result correctness using WordNet is lower than 80%, Urban Dictionary is tried as the next search engine.
4. Because of the restricted automated use, Google search is used as a last option in case the probability of the result correctness using Urban Dictionary is lower than 80%.

After that, the *OD&GS* algorithm returns the list of possible parent terms. The number of the results is limited to 9. This number was determined empirically based on the number of correct results in our experiments with the terminology of multiple existing UIs.

For each child term set, it is checked if the parent of the set is found in possible parent terms generated by *OD&GS* algorithm. If it is not the case, a warning is shown, and terms obtained by the *OD&GS* are suggested as alternatives.

The results of the *OD&GS* algorithm strongly depend on the quality of the used ontological dictionaries. In the next sections, we explain how each of the data sources is used.

5.2.1. WordNet

WordNet (https://wordnet.princeton.edu; accessed on 9 August 2021) is a dictionary and a lexical database. The dictionary provides direct and inherited hypernyms as a part of word definition for nouns, adjectives, and verbs. As a query result, WordNet returns so-called *synsets*, containing the information about words including the given word class. We filter out synsets with different word classes compared to the child word. To ensure higher accuracy of the results, we include only direct hypernyms. As a result, we construct a list of hypernyms for each child word set.

5.2.2. Urban Dictionary

Urban Dictionary (http://www.urbandictionary.com; accessed on 9 August 2021) is a crowdsourced dictionary. For each queried word it returns seven most popular definitions based on the votes of the Urban Dictionary users. For each query, we collect all meaningful words from the definitions. The words are sorted by the frequency of their occurrence. The result is a list of the words with the highest frequency that can be considered possible parent terms.

5.2.3. Google Web Search

While Google is not a linguistic tool, the current state of its multi-layered semantic network—*Knowledge Graph* [30,31]—enables gaining quite accurate results to confirm linguistic relations such as hyponymy, hypernymy, meronymy, and holonymy by using web search queries. The efficiency of data collection of Google's semantic network database enables it to grow its data into gigantic dimensions as opposed to any semantic network, including WordNet and UrbanDictionary, and for that reason, we see greater potential in web search than in current ontological dictionaries.

Based on our tests, Google search provides the most precise results compared to other sources we have used. On the other hand, it is not very suitable for automated requests. Because the Google web search approach provides results with high reliability, we present it in this paper despite the restrictions.

To search potential parent terms, we use two queries with the list of child words:

- `{words separated by commas} are common values for`
- `{words separated by commas} are`

For example: "`red, green, blue, brown are common values for`" or "`red, green, blue, brown are`".

We parse the returned HTML documents and count the most common words. The probability of each word in the result is based on the frequency of its occurrence. Additionally, we ignore words of a different word class from the class of child words.

To verify the gained results we use the reverse queries for each child word: "`is {a possible parent term} value/kind of {word}`", for example, "`is color kind of blue`", "`is color kind of yellow`".

The number of occurrences of both words found in the resulting HTML page is used to determine the probability of the found words being the correct parent terms for the particular child word set. If there is low or no occurrence of a particular pair, this pair has the lowest probability in the result list.

5.3. Grammar Evaluation

There are two common grammatical issues occurring in user interfaces: an incorrectly written word (a typo), or a word that was not translated into the languages of the user interface. The second case is especially common in applications that are localized in multiple languages.

For this reason, usual spell checking is supplemented with the translation checking. If some word is not found in the dictionary for the current language, the algorithm checks the default language (usually English). If it is found, its translations are added to recommended replacements. Otherwise, the recommendations are based on similar words in the same way as it is done in modern text editors. In the end, a list of recommended corrections is provided to the evaluator.

5.4. Tooltip Analysis

The Tooltip analysis algorithm (TTA) selects all *functional* terms, i.e., terms extracted from functional components, from the domain model. Then for every such term, the presence of a tooltip is checked—either by inspecting the representing component or by checking the description property of the term node, where the component's tooltip text is usually stored. If no tooltip is found, this information is added to the list of warnings, and we advise the developer to add it.

Because general-purpose components (*OK, Open, Save, Exit, Cancel*, etc.) are common, frequently used, and generally understood, we presume that the importance of tooltips for such components is very small. Their purpose is clear from their description and/or icon. For this reason, we only analyze domain-specific components. General-purpose components are removed in the DEAL's *Extraction* phase using the general application terms ontology described in Section 4.2.

If no tooltip is found for some functional component, the result is displayed to the evaluator in one of two ways:

- *recommendation to add a tooltip*—if the component has at least one user-readable textual description (e.g., label),
- *usability issue*—if either the component is general-purpose and has *only* an icon, or it is a domain-specific one with *only* an icon or *only* a textual label, this is considered a *domain usability issue* and is displayed to the evaluator.

An example of the usability issue and its report to the user can be seen in the JSesh interface menu items (Figure 4) where there are two items with no visible textual information and/or tooltip.

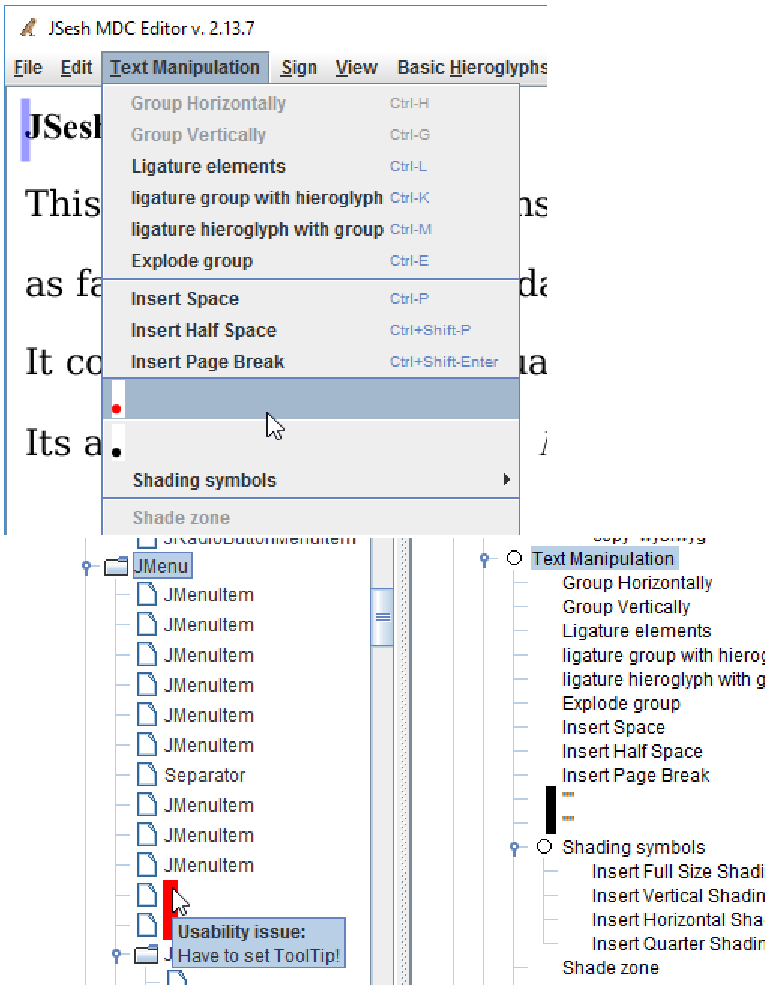


Figure 4. Example of JSesh menu items both without a tooltip and label (**top**) and a usability issue reported to the user (**bottom**).

6. Prototype

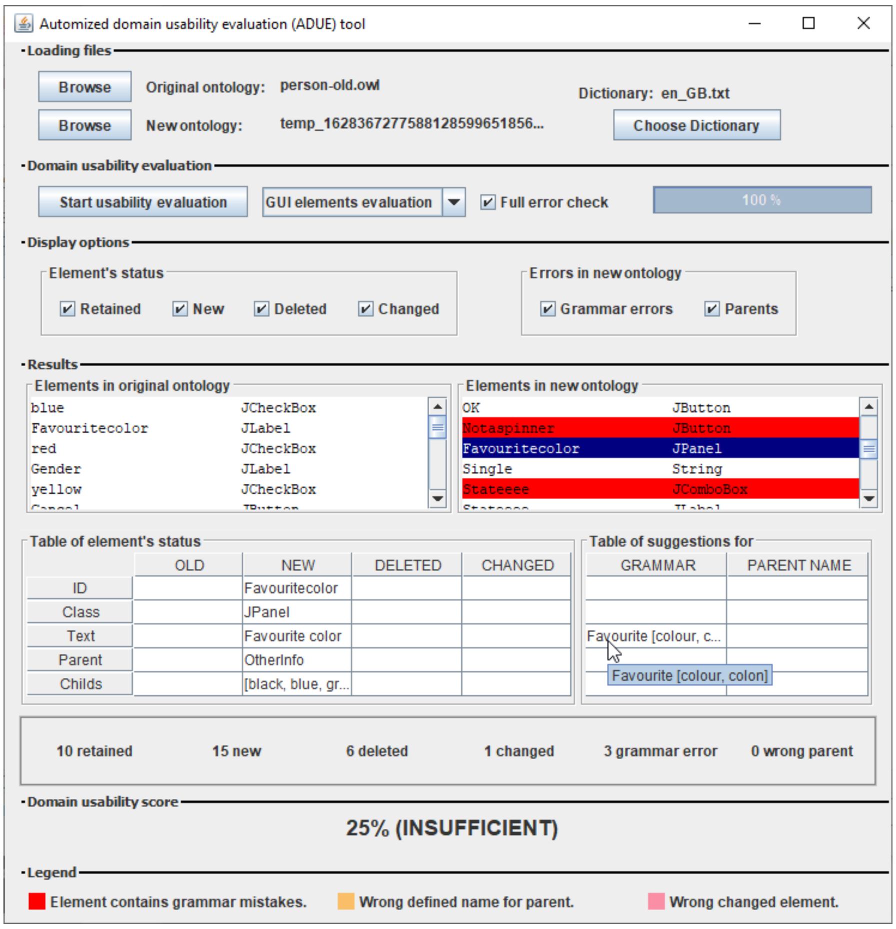
All processes mentioned in Section 5 were implemented and integrated into the DEAL tool. The results of tooltip and form stereotype analysis are displayed as tooltips in the DEAL's domain model as seen in Figures 2 and 4.

The process of domain usability evaluation can be activated using a menu item in DEAL. Results of the analysis are displayed as errors (highlighted with red color) and recommendations (highlighted with orange) in the DEAL's component tree. Recommendations for corrections are displayed in tooltips. DEAL enables us to look up any component in the application by clicking on it in the component tree. As a result, the component is highlighted by the yellow color directly in the analyzed application. This way the analyst can locate the component needing the recommended modification.

Ontological evaluation, grammar evaluation, and specificity evaluation are implemented in a tool called ADUE (Figure 5), which can be started directly from DEAL or as a standalone process. In the case of starting from DEAL, the newest ontology is automatically extracted from the application currently analyzed by the DEAL tool. In the latter case, both ontologies (old and new) have to be imported manually.

When running the process with only one ontology, then only grammar and specificity evaluation is performed, and results are displayed only in the right column.

When loading two ontologies, the former processes are performed on the newer ontology as an additional process, and both ontologies are compared. Results are similar to one ontology analysis, but in the left column, we can see the components (terms) in the older application.



Automized domain usability evaluation (ADUE) tool

Loading files

Original ontology: person-old.owl

New ontology: temp_1628367277588128599651856...

Dictionary: en_GB.txt

Domain usability evaluation

GUI elements evaluation ✓ Full error check 100 %

Display options

Element's status: ✓ Retained ✓ New ✓ Deleted ✓ Changed

Errors in new ontology: ✓ Grammar errors ✓ Parents

Results

Elements in original ontology

blue	JCheckBox
Favouritecolor	JLabel
red	JCheckBox
Gender	JLabel
yellow	JCheckBox

Elements in new ontology

OK	JButton
🟥 Notaspinner	JButton
Favouritecolor	JPanel
Single	String
🟥 Stateeee	JComboBox

Table of element's status

	OLD	NEW	DELETED	CHANGED
ID		Favouritecolor		
Class		JPanel		
Text		Favourite color		
Parent		OtherInfo		
Childs		[black, blue, gr...		

Table of suggestions for

GRAMMAR	PARENT NAME
Favourite [colour, c...	

Favourite [colour, colon]

10 retained 15 new 6 deleted 1 changed 3 grammar error 0 wrong parent

Domain usability score

25% (INSUFFICIENT)

Legend

🟥 Element contains grammar mistakes. 🟧 Wrong defined name for parent. 🟥 Wrong changed element.

Figure 5. The ADUE evaluation tool displaying the results from comparing two sample applications.

Different types of errors are displayed using colors. Red is used for grammar errors. Orange means an incorrectly defined parent term (hypernym, holonym). Recommendations are displayed in a tooltip. The pink color is used for illogically changed components. The evaluator can also see all terms that were retained, added, deleted, or changed. In all cases, we display recommendations for change in the *Table of suggestions* (bottom right).

We used the metrics described in Section 2 to calculate the overall DU score of the evaluated user interface (the percentage in the bottom part of Figure 5). The errors are included in the DU as follows:

- the number of missing tooltips and incorrectly changed or deleted components is counted as domain content issues;
- the number of incorrectly defined parents is counted as domain specificity issues;
- the number of grammar errors is counted as language errors and barriers.

As explained in the paper, we were not able to analyze consistency issues, and world language issues are indistinguishable from grammar errors; therefore, the number of errors for these two aspects remains 0 and does not affect the DU score calculation.

ADUE for Java Applications

To be able to extract data from Java applications, DEAL uses Java *reflection* and *aspect-oriented programming* (AOP). AOP in load-time enables us to weave and also to analyze applications with custom class loaders, which would be problematic using a simple reflection. There are still limitations in some cases; e.g., AOP is not able to weave directly into Java packages such as *javax.swing*. Weaving directly into the JDK source code and thus creating our own version of Java to run the target application would solve the issue.

To extract, traverse, and compare ontologies, we used the *OWL API* library (https://github.com/owlcs/owlapi/wiki; accessed on 9 August 2021). As a dictionary in the grammar evaluation, we used the US English dictionary from the *OpenOffice* text editor (https://www.openoffice.org; accessed on 9 August 2021). We chose this dictionary because of the simple textual format with words separated by newline characters and because it can be freely edited and complemented by new words. In the same package, there are also multiple languages available, so they can be used for the evaluation of applications in other languages. To check the grammar, the *JAZZY* library (http://jazzy.sourceforge.net; accessed on 9 August 2021) was used. After identifying a typo in a text, *JAZZY* returns multiple replacement recommendations of the incorrect word. For natural language processing needed in the specificity evaluation, we used the *Apache OpenNLP* library (https://opennlp.apache.org; accessed on 9 August 2021), which can identify the word classes such as verbs, nouns, or adjectives. To query the WordNet dictionary, the *JAWS* library (https://github.com/jaytaylor/jaws; accessed on 9 August 2021) was used. Urban Dictionary does not provide a special API for machine usage. Therefore, we used standard HTTP GET requests to query the dictionary and then analyzed the source code of the response pages statically. To query the Google search engine, we used the publicly available API (https://developers.google.com/custom-search/v1/overview; accessed on 9 August 2021).

Ontologies were used because of good support for export and a comparison engine. However, in our approach, the main *limitation* of ontologies is considered the inability to use special characters and spaces in identifiers. In the case of comparing ontologies, it does not represent a problem. However, when analyzing grammar and specificity, this is usually the main issue.

7. Evaluation

In this section, we will assess the possibility of using ADUE on existing applications. Our main questions are whether ADUE is applicable to real-world programs and to what degree these programs contain domain usability errors.

7.1. Method

Since the implementation of ADUE for Java program analysis is the most mature one, we used several open-source Java GUI applications as study subjects. To obtain such applications, we utilized the SourceForge website (http://sourceforge.net; accessed on 9 August 2021). We selected programs from diverse domains and of various sizes to maximize generalizability. To simplify the interpretation of the results, we focused only on applications in the English language.

Specifically, the following applications were used to evaluate the ADUE prototype: Calculator, Sweet Home 3D, FreeMind (2014), FreePlane (2015), Finanx, JarsBrowser, JavaNotePad, TimeSlotTracker, Gait Monitoring+, Activity Prediction Tool, VOpR (a virtual optical rail), GDL Editor 0.9, and GDL Editor 0.95. The specific versions of the applications can be downloaded from https://git.kpi.fei.tuke.sk/michaela.bacikova/DEAL/-/tree/master/DEALexamples/examples (accessed on 9 August 2021).

We executed the complete analysis using our implemented ADUE tool and recorded the results. The form stereotype analysis, tooltip detection, grammar error evaluation, parent term evaluation, and the overall domain usability computation were executed on all applications. For some of the applications, we performed an ontology comparison between two different versions (GDL Editor 0.9 and 0.95) or editions (FreeMind and FreePlan). We also recorded the execution times of the analysis process. All results were written in a spreadsheet.

7.2. Results

We were able to successfully execute ADUE on all mentioned applications. Table 3 presents an overview of the obtained results. For each application, we can see the number of extracted terms and different kinds of errors and warnings detected by the ADUE prototype. The is also a weighted number of errors (e) calculated using Equation (3) and final domain usability index (du). The results of the two-ontology comparison are available in Table 4. The complete results can be viewed via Google Sheets using the following URL: http://bit.ly/3hZBImy (accessed on 9 August 2021).

Table 3. Results of the evaluation (applications where ontology comparison was used are marked with *).

Application	Terms	Tooltip Errors	Tooltip Warnings	Grammar Errors	Incorrect Parents	e	du	Execution Time
Calculator	40	0	0	1	0	1.7	96	0 s
Sweet Home 3D	200	13	11	4	17	84.4	58	2 m 0 s
FreeMind 2014	273	1	94	14	17	68.3	75	1 m 50 s
FreePlane 2015 *	873	13	323	128	33	833.5	5	5 m 6 s
Finanx	74	39	9	4	8	140.7	90	36 s
JarsBrowser	19	0	8	2	5	16.4	14	8 s
BaseFormApplication	74	0	8	11	8	42.1	43	42 s
JavaNotePad	19	0	17	0	5	13.0	32	32 s
TimeSlotTracker	62	6	36	7	10	55.0	11	55 s
Gait Monitoring+	70	0	17	0	7	18.2	74	29 s
Activity Prediction Tool	98	1	84	2	11	33.2	66	1 m 19 s
VOpR	96	0	21	23	8	59.9	38	44 s
GDL Editor 0.9	73	4	8	4	11	45.3	38	58 s
GDL Editor 0.95 *	75	4	8	4	11	61.5	18	15 s

Table 4. Results of the ontology comparison.

Application	Original Application	New Terms	Deleted Terms	Changed Terms	Incorrectly Changed Terms
FreePlane 2015	FreeMind 2014	748	168	93	0
GDL Editor 0.95	GDL Editor 0.9	7	5	4	0

7.2.1. Tooltip Analysis

By using the tooltip verifier process, we extracted 136 components per application on average. From those, 52 function components per application on average had no tooltip defined (38%), from which 46 were a recommendation (34%) and 6 were an error (4%). We manually checked the components associated with the errors and confirmed that these issues were correctly identified.

The results show that DU issues concerning tooltips are very common in applications. Developers are probably not fully aware that tooltips are necessary for application usability.

7.2.2. Grammar and Specificity Evaluation

From each listed application, we extracted an ontology using the DEAL tool and performed the grammar evaluation on it. On average, we extracted 146 items per application from which 15 grammar errors and 11 incorrectly defined parents were identified.

Some of the detected issues represented acronyms, abbreviations, and proper nouns. It is questionable to what degree acronyms and abbreviations are comprehensible to the application users. A portion of the grammar errors was caused by the fact that we were using the US English dictionary, but some applications used British English (or possibly used a combination of US and British English, which is inconsistent).

7.2.3. Ontological Comparison

The two-ontology evaluation was applied only to the FreeMind/FreePlane and GDL Editor 0.9/0.95 applications since they are two versions of the same applications. As we can see in Table 4, numerous elements were added, deleted, or changed in the case of Free-

Mind/FreePlane since this version change represents a major redesign of the application. On the other hand, in GDL Editor, a smaller proportion of the terms was changed because this version update is minor.

Note that there were no incorrectly changed components detected in either application.

7.2.4. Overall Domain Usability

As we can see in Table 3, the computed domain usability ranged from 5% to 96%. The computed mean value is 47%. Therefore, the variability of the overall domain usability among the analyzed applications is relatively large.

Applications with low computed domain usability tend to have mainly a high number of detected grammar errors but also incorrectly defined parent terms and missing tooltips in places where they are necessary.

7.2.5. Execution Time

The execution process of DEAL and ADUE includes the traversal of GUI elements of the applications, querying Web services, and other time-consuming operations. For this reason, we would like to know whether the execution time of the domain usability evaluation process is not prohibitive with respect to its practical utilization.

According to our results, the execution time on the listed applications ranges from 0 s to 5 min and 6 s, with a mean of 1 min and 7 s. This means that automated domain usability evaluation could be potentially performed in a variety of contexts, including continuous integration (CI) builds.

7.3. Examples of Issues

To help the reader understand the nature of domain usability issues, we will now mention a few examples of specific issues found by ADUE.

OpenRocket is a model rocket simulator, containing buttons to zoom out and zoom in. Each of them contains an icon with a magnifying glass and a small sign "−" and "+", respectively. However, these buttons do not contain any textual label or a tooltip. ADUE suggests adding tooltips to these buttons.

OpenRocket also contains multiple sliders, e.g., to control the wind direction or various angles. Next to each slider, there is a numeric input field and a textual descriptive label. However, there is no programmatical connection between the label ("Wind direction:"), the numeric value ("0°"), and the graphical slider. ADUE reports the missing *labelFor* attributes.

An example of a questionable grammar error can be found in the financial calculator Finanx. It contains a list of languages that are translated to the corresponding language instead of English (e.g., Français instead of French). Technically, the word is incorrect, and it should be translated into the language of the application (English). On the other hand, in some contexts, e.g., UI language selection, it can practically help the user to find hist or her language in the list, particularly if the person does not speak English.

7.4. Threats to Validity

Regarding the internal validity, a portion of the detected issues might have been false positives. To mitigate this threat, for selected analysis types, we manually verified a subset of the results to check their correctness. To improve grammar error detection, in the future, we should implement an option to add a word to the dictionary in ADUE, similarly to traditional spell-checking applications.

The largest threat to the external validity is the selection of applications, which might not be representative of the whole set of Java GUI programs. However, we tried to select applications from multiple different domains and ranging from small one-window utilities to complex software systems.

7.5. Evaluation Conclusion

From the results, we can conclude that ADUE can be successfully used on existing real-world Java applications with graphical user interfaces. The tool discovered many domain usability errors, including tooltip errors and warnings, grammar errors, or incorrect parent terms. The overall domain usability of the analyzed applications has high variability (5–96%), which points to the fact that developers are often not aware of domain usability problems, and we need to raise awareness about domain usability issues among them.

8. Potential of Existing Methods for DU Evaluation

After describing the results of the evaluation of our prototype, in the next two sections, we will try to put our work into the context of existing approaches and propose their extensions if suitable.

The goals of usability evaluation methods are usually to specify the requirements for the UI design, evaluate design alternatives, identify specific usability issues, and improve UI performance [16]. In this section, we will summarize existing general techniques of usability evaluation, and for some of them, we will propose modifications that could make them suitable to evaluate domain usability.

8.1. Universal Techniques

Simple, universally usable techniques that include users, such as *thinking aloud* [32], *question-asking protocol* [33], *performance measurement*, or *log file analysis* [34,35], can be easily altered to focus on domain dictionary by just changing the questions or tasks included in the process to obtain the desired outcome. If there is a recording output, it can be analyzed with respect to DU. Informal or structured *interviews* and *focus groups* [36] might also be directed on the domain user dictionary by asking the participants (i) whether they understand such or such terminology in the UI, (ii) whether they use it in their everyday work life in their own domain, and (iii) if not, what would they use instead.

8.2. User Testing Techniques

There are multiple types of *user testing* [37] differentiated by automation, distance from the user (in the room, in the observation lab, remote testing), and recording outputs (sound or image recording of user and/or screen, user logs, notes, software usage records, eye tracking, brain waves, etc.). All of them are usually connected by a more or less functioning system or prototype and users performing pre-prepared scenarios.

Possible alterations to the user testing technique are the following:

- Before the testing begins, the user is instructed to focus on domain terminology issues when performing the test.
- In the types of testing where the usability expert is present during the test, questions about term understandability are asked by the usability expert during each task of the scenario.
- The subject user is prompted to express proposals for new terminology for any item in the system and to explain why (s)he thinks the new terminology is appropriate for the particular item (incorrect, inapposite, does not reflect the given concept, etc.). Proposals from all users are recorded and evaluated for the most common ones that should serve as future replacements in the UI.
- If alternative translations of the UI are being tested, the testing should take place with the users naturally speaking the language of the translation. The users are prompted to propose a different translation for any item in the system and explain why they think the new translation is more appropriate for the particular item (incorrect or erroneous translation, more suitable term). Proposals from all users are recorded and evaluated for the most common ones. They can also be evaluated in a second phase where participants see the replaced terminology directly in the UI and check for correctness.

- In A/B testing, multiple versions of UIs with different terminology alternatives are created and tested by the users.

8.3. Inspection Methods

In general usability inspection methods described by Boehm et al. [38] and Nielsen and Mack [39], the expert in usability usually performs the inspection of guidelines, heuristic rules, product consistency, or standards compliance of a prototype.

8.3.1. Specializations of General Methods

Narrowing to DU, we propose the following alternations to the general techniques:

- Guideline review, cognitive walkthrough, heuristic evaluation techniques, formal usability inspection, and standards inspection: an expert performs the check focusing on domain terminology, consistency, and errors.
- To achieve the best results on the aforementioned techniques, the expert needs to be a domain expert.
- Another option is a pluralistic walkthrough technique, where one evaluator is an expert on usability and UX and the other is a domain expert. They both cooperate to imagine how the user would work with the design and try to find potential DU issues.
- Consistency inspection: the expert performs consistency checks across multiple systems *and* across the same system. The focus should be on the terminology, including:
  - different terms naming the same functionality or concepts (e.g., *OK* on one place, *Confirm* on the other);
  - same terms naming different functionality or concepts;
  - uppercase and lowercase letters consistency (e.g., *File, file, FILE*);
  - consistency of term hierarchies, properties, and relations.

8.3.2. Cognitive Walkthrough

As for the *Cognitive Walkthrough (CW)*, we propose an alternation of the latest Wharton et al.'s method [40], marked by Mahatody et al. [41] as *CW3* (this notation will be used further in this subsection).

The evaluator in CW3 should imagine a specific scenario for each action that the target users must accomplish to achieve the completion of their task. To achieve the best results, again, the evaluator should be a domain expert. A scenario should also be credible according to Wharton et al. [40], which means that the user's background knowledge and the feedback from the interface should be justified when evaluating each action. When evaluating domain usability, we recommend focusing on the user's background and knowledge first.

We propose to answer the following supplements to CW3's questions [41] related to various user thoughts and actions (Note: Question Q1 remains unchanged, and our supplements are marked by italic font):

1. What is the user thinking at the beginning of the action? (Q1: Will the user try to achieve the right effect?)
2. Is the user able to locate the command? (Q2: Will the user notice that the correct action is available? *Is the action appropriately and consistently described by a domain-related term and/or understandable to the user?*)
3. Is the user able to identify the command? (Q3: Will the user associate the correct action with the effect that (s)he is trying to achieve? *Is there any other action with a similar label and/or graphics, which would lead the user astray?*)
4. Is the user able to interpret the feedback? (Q4: If the correct action is performed, will the user see that progress is being made toward the solution of the task? *Is the feedback reported to the user expressed in consistent terms and/or graphics understandable to the user?*).

Provided the fact that the target user is the best source of domain knowledge, it would be possible to use an alteration in the "CW with users" approach by Gonz et al. [42]. However, it is questionable whether "CW with users" is still a CW, since the essence of CW techniques is the evaluation by experts, excluding users. If available, we recommend using domain experts instead of target users.

8.4. Inquiry

Inquiry techniques are those that focus on user feedback. They include focus groups [43], interviews and surveys [44], questionnaires [45,46], and others. There are two categories of inquiry techniques we would like to focus on: in-system user feedback, and surveys and questionnaires.

8.4.1. In-System User Feedback

General techniques are based on the user sending feedback in a form of recorded events [47], captured screens, or submitted comments. We propose the following techniques for evaluating DU:

- For web UIs, it is possible to create a system or a browser plug-in enabling the user to mark any inappropriate terminology in the UI and/or change the label or tooltip of the particular element in the UI. Every change is logged and sent to a central server where the evaluator can review the logs recorded from multiple users. The priority of change is calculated automatically by the number of users proposing a particular terminology change. The proposed terms can be assessed as a percentage according to the number of users proposing the same term.
- For any UI, a separate form can be made where the user selects one of the pre-prepared lists of application features (labeled and with icons for better recognizability) and sends comments on how and why to change the description of the particular feature. However, it is best to comment directly in the target UI because of the context.
- For both possibilities, the users can assess the *appropriateness* of a particular term using the approach by Isohella and Nissila. [8].

8.4.2. Surveys and Questionnaires

Most of the common standard usability surveys and questionnaires [48] are defined too generally to be usable for DU evaluation. This was the primary reason for our proposal of a novel SDUS (System Domain Usability Scale) technique in 2018 [10]. SDUS is based on the common standardized System Usability Scale (SUS) [49,50], which is widely used in the user experience evaluation practice.

Similarly to SUS, our proposal also included a questionnaire with 10 statements targeted at all DU aspects. We designed SDUS similarly to SUS, which means that odd questions were positive and even questions were negative statements. The answers are in the standard five-point Likert scale (*1—Disagree, 5—Agree*). The overall DU metric is a sum of values for all answers. The calculation of the SDUS score is the same as the standard SUS [51].

8.5. Analytical Modeling Techniques

The goal of *GOMS (Goals, operations, methods, selection rules)* analysis [52,53] is to predict user execution and learning time. Learning time is partially determined by the appropriate terminology, but without a domain expert, it is not easy to evaluate it either automatically or manually. Calculating the overall *appropriateness* of terms [8] per system might provide a good view of the system improvement since the last prototype.

In *cognitive task analysis* [54], the evaluators try to predict usability problems. We claim that it is partially possible to semi-automatically evaluate existing UIs to find potential DU problems. We propose several techniques to support this claim in Section 3.

Knowledge analysis is aimed at system learnability prediction. It is only logical that the more appropriate the domain content of the UI is, the more learnable it is. This relates not only to the terminology but also to icons, which should be domain-centric, especially in cases when the particular feature or item is domain-related. Several techniques proposed in Section 3 address this issue.

Design analysis aims to assess the design complexity. From the point of view of DU, the complexity of textual content in web UIs can already be assessed by multiple online tools such as Readable (http://readable.com; accessed on 9 August 2021). Readability Score evaluates the given text or a URL and determines multiple reading complexity indices including Flesch–Kincaid [55,56], Keyword Density, and similar.

The goal of *Programmable User Models* [57] is to write a program that acts similarly to a user. Currently, our proposed tool is able to simulate users on existing UIs using a domain-specific language [58]. This automated approach was developed with the goal of testing user interfaces from the domain task-oriented point of view, and it is not related to the main goal of this paper.

8.6. Simulation Techniques

Simulation techniques, similarly to *Programmable User Model*, try to mimic user interaction. Many tools for end-to-end user testing exist, e.g., Protractor (Protractor end-to-end testing framework: https://www.protractortest.org; accessed on 9 August 2021) for Angular. However, similarly to *Programmable user Models*, simulation represents a general technique that is not specifically related to DU and therefore exceeds the focus of this paper.

8.7. Automated Evaluation Methods

To date, we have focused on manual or semi-automatized techniques. As for automated approaches, as mentioned in the introduction, we found only one by Mahajan and Shneiderman [59] that enables consistency checking of UI terminology. Their tool is quite obsolete and does not evaluate whether different terms are describing the same functionality. However, the methodical approach is applicable to all UIs. In Section 3, we introduced a novel approach to semi-automatic DU evaluation of existing UIs that includes consistency checking similar to Mahajan and Shneiderman's style, but extends the approach with multiple evaluation techniques.

9. Related Work

In this section, we selected the most important state-of-the-art works that refer to the aspects of DU, although they might have used different terminology compared to our definition. The number of works referring to matching the application's content to the real world indicates the importance of DU.

9.1. Domain Content

Most often, the existing literature refers to the *domain content* aspect of DU as to one of the following:

- *Textual content of UIs*—Jacob Nielsen refers to DU aspects only too generally and stresses the importance of "the system's addressing the user's knowledge of the domain" [1].
- *Domain dictionary, Ontology*—the importance of domain dictionary of UIs is stressed also by Artemieva [60], Kleshchev [61], and Gribova [62], who also presented a method of estimating usability of a UI based on its model. Her model is rather component-oriented than focused specifically on the domain, and she focuses primarily on general usability evaluation methods such as having too many menu items in a menu.
- *Domain structure*—by Billman et al. [63]. Their experiment with NASA users showed that there is a big difference in the performance of users with respect to the usability of the old application and that of the the new, as the new application was better in domain-specific terminology structure.
- *User interface semantics, Ambiguity*—Tilly and Porkoláb [64] propose using *semantic UIs* (SUI) to solve the problem of the ambiguity of UI terminology. The core of SUI

is a general ontology that is a basis for creating all UIs in the specific domain. User interfaces can have a different appearance and arrangement but the domain dictionary must remain the same. Ontologies in general also deal with the semantics of UIs.

- *Complexity, reading complexity*—Becker [65], Kincaid et al. [55], and Mahajan and Shneiderman [66] stress that the complexity of the textual content should not be too high because that would make the application less usable. Kincaid et al. refer to the reading complexity indices (ARI, Kincaid). Complexity is closely related to the *domain content* DU aspect: the UI should have the reading complexity appropriate for the target users.
- *Matching with the real world or correspondence to the domain*—Many of the above-listed authors, along with Badashian et al. [12], also stress the importance of applications corresponding to the real world and address the user's domain knowledge. In fact, this is a more general description of our *domain content* DU aspect. Hilbert and Redmiles [47] stress the correspondence of event sequences with the real world as well as the domain dictionary.
- *Knowledge aspect of UI design*—One of the attributes of Eason's usability definition [67] refers to the *knowledge* aspect of UI design representing the knowledge that the user applies to the task, and it may be appropriate or inappropriate. In general, the *task match* attribute of Eason's definition also refers to processes mapping but does not explicitly target the mapping of specific domain tasks.
- *Appropriateness recognizability*—defined by ISO/IEC-25010 [68] as an aspect referring to the user understanding whether the software is appropriate for their needs and how it can be used for particular tasks and conditions of use. The term was redefined in 2011 from *Understandability*. However, again, the term appropriateness recognizability does not specifically refer to the target domain match,
- Other definitions such as ISO-9241-11 [5] or definitions by Nielsen [1], Shackel [69], and others [70] are too general but we do not exclude DU as a subset of them.

9.2. Consistency

Among other aspects, Badashian et al. [12] stress the importance of *consistency* in usable UIs. The survey by Ivory and Hearst [16] contains a wide list of automatic usability methods and tools. From over 100 works, only Mahajan and Shneiderman [59] deal with the domain content of applications, and their Sherlock tool is able to automatically check the consistency of UI terminology. Sherlock, however, does not evaluate whether different terms describe the same functionality or not.

9.3. World Language, Language Barriers, Errors

In addition to complexity, Becker [65] also deals with the *translation* of UIs, which corresponds to the *world language* DU aspect. In the area of web accessibility [13], the *understandability* of web documents is defined by W3C. Compared to our definition, however, it deals only with some of the attributes: *world language* of web UIs, *language barriers*, and *errors*. It focuses on web pages specifically, not on UIs in general.

9.4. All Domain Usability Aspects

Isohella and Nissila [8] evaluate the *appropriateness* of UI terminology based on the evaluation of users. In a broader sense, appropriateness is equivalent to our DU definition but Isohella and Nissila do not go deeper into the definition's aspects. According to the authors, appropriate terminology can increase the quality of information systems. The terminology should be selected, formed, evaluated, and used.

10. Conclusions

In this paper, we described the design and implementation of a method for automatized DU evaluation of existing user interfaces. The method not only evaluates the user interfaces for domain usability but also (probably even more importantly) provides recommendations for their improvement. The method was verified using the implemented prototype on several existing open-source Java applications with graphical user interfaces. Among other findings, we conclude that the variability of the computed domain usability of individual applications is high. Many components do not contain tooltips or have grammatical errors.

As a secondary contribution, we proposed several modifications of existing manual techniques of usability evaluation to utilize them specifically for domain usability evaluation.

Ontologies provide good tools for content comparison, but they have restrictions (such as ID uniqueness) that restrict our approach and the ontological format is rather extensive. Therefore, in the future, we plan to define a new domain-specific language (DSL) for formal domain model description [71] and a custom comparison engine for domain models exported in the DSL.

We believe that the ADUE method contributes to the field of UX and usability and hope that it improves the situation in DU of new user interfaces.

Author Contributions: Conceptualization, M.B. and J.P.; methodology, M.B., J.P., M.S., S.C., W.S. and M.M.; software, M.B.; formal analysis, M.B.; investigation, M.B.; data curation, M.B.; writing—original draft preparation, M.B.; writing—review and editing, M.B., J.P., M.S., S.C., W.S. and M.M.; visualization, M.B. and M.S.; funding acquisition, J.P. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by Project VEGA No. 1/0762/19 Interactive pattern-driven language development.

Data Availability Statement: The evaluation results of ADUE can be found at http://bit.ly/3hZBImy (accessed on 9 August 2021). The General Application Terms Ontology can be found at https://bit.ly/2R6bm6p (accessed on 9 August 2021).

Conflicts of Interest: The authors declare no conflict of interest. The funders had no role in the design of the study, in the collection, analyses, or interpretation of data, in the writing of the manuscript, or in the decision to publish the results.

References

1. Nielsen, J. *Usability Engineering*; Morgan Kaufmann Publishers, Inc.: San Francisco, CA, USA, 1993.
2. Norman, D. *The Design of Everyday Things: Revised and Expanded Edition*; Basic Books: New York, NY, USA, 2013.
3. Morville, P. User Experience Design. 2004. Available online: http://semanticstudios.com/user_experience_design (accessed on 9 August 2021).
4. Lewis, J.R. Usability: Lessons Learned … and Yet to Be Learned. *Int. J. Hum. Comput. Interact.* **2014**, *30*, 663–684, doi:10.1080/10447318.2014.930311.
5. ISO-9241-11. *Ergonomics of Human-System Interaction—Part 11: Usability: Definitions and Concepts*; ISO: Geneva, Switzerland, 2018.
6. Chilana, P.K.; Wobbrock, J.O.; Ko, A.J. Understanding Usability Practices in Complex Domains. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '10), Atlanta, GA, USA, 10–15 April 2010; ACM: New York, NY, USA, 2010; pp. 2337–2346, doi:10.1145/1753326.1753678.
7. Gulliksen, J. *Designing for Usability—Domain Specific Human-Computer Interfaces in Working Life*; Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science & Technology; Acta Universitatis Upsaliensis: Uppsala, Switzerland, 1996; p. 28.
8. Isohella, S.; Nissila, N. Connecting usability with terminology: Achieving usability by using appropriate terms. In Proceedings of the 2015 IEEE International Professional Communication Conference (IPCC '15), Limerick, Ireland, 12–15 July 2015; pp. 1–5, doi:10.1109/IPCC.2015.7235849.
9. Lanthaler, M.; Gütl, C. Model Your Application Domain, Not Your JSON Structures. In Proceedings of the 22nd International Conference on World Wide Web, Rio de Janeiro, Brazil, 13–17 May 2013; ACM: New York, NY, USA, 2013; pp. 1415–1420, doi:10.1145/2487788.2488184.
10. Bačíková, M.; Galko, L. The design of manual domain usability evaluation techniques. *Open Comput. Sci.* **2018**, *8*, 51–67, doi:10.1515/comp-2018-0005.
11. Bačíková, M.; Porubän, J. Domain Usability, User's Perception. In *Human-Computer Systems Interaction: Backgrounds and Applications 3*; Springer International Publishing: Cham, Switzerland, 2014; pp. 15–26, doi:10.1007/978-3-319-08491-6_2.
12. Badashian, A.S.; Mahdavi, M.; Pourshirmohammadi, A.; Nejad, M.M. Fundamental Usability Guidelines for User Interface Design. In Proceedings of the 2008 International Conference on Computational Sciences and Its Applications (ICCSA '08), Perugia, Italy, 30 June–3 July 2008; IEEE Computer Society: Washington, DC, USA, 2008; pp. 106–113, doi:10.1109/ICCSA.2008.45.

13. W3C. Web Content Accessibility Guidelines (WCAG) 2.0, Part 3 about Understandability, 2008. Available online: https://www.w3.org/TR/WCAG20/#understandable (accessed on 9 August 2021).
14. Kolski, C.; Millot, P. A rule-based approach to the ergonomic "static" evaluation of man-machine graphic interface in industrial processes. *Int. J. Man Mach. Stud.* **1991**, *35*, 657–674, doi:10.1016/S0020-7373(05)80182-8.
15. Sears, A. AIDE: A Step toward Metric-Based Interface Development Tools. In Proceedings of the 8th Annual ACM Symposium on User Interface and Software Technology (UIST '95), Pittsburgh, PA, USA, 15–17 November 1995; Association for Computing Machinery: New York, NY, USA, 1995; pp. 101–110, doi:10.1145/215585.215704.
16. Ivory, M.Y.; Hearst, M.A. The state of the art in automating usability evaluation of user interfaces. *ACM Comput. Surv.* **2001**, *33*, 470–516, doi:10.1145/503112.503114.
17. Tullis, T.S. The Formatting of Alphanumeric Displays: A Review and Analysis. *Hum. Factors* **1983**, *25*, 657–682, doi:10.1177/001872088302500604.
18. Paz, F.; Pow-Sang, J.A. Current Trends in Usability Evaluation Methods: A Systematic Review. In Proceedings of the 2014 7th International Conference on Advanced Software Engineering and Its Applications, Hainan, China, 20–23 December 2014; pp. 11–15, doi:10.1109/ASEA.2014.10.
19. Bakaev, M.; Mamysheva, T.; Gaedke, M. Current trends in automating usability evaluation of websites: Can you manage what you ca not measure? In Proceedings of the 2016 11th International Forum on Strategic Technology (IFOST), Novosibirsk, Russia, 1–3 June 2016; pp. 510–514, doi:10.1109/IFOST.2016.7884307.
20. Namoun, A.; Alrehaili, A.; Tufail, A. A Review of Automated Website Usability Evaluation Tools: Research Issues and Challenges. In *Design, User Experience, and Usability: UX Research and Design*; Springer: Cham, Switzerland, 2021; pp. 292–311, doi:10.1007/978-3-030-78221-4_20.
21. Bačíková, M.; Porubän, J. Ergonomic vs. domain usability of user interfaces. In Proceedings of the 2013 The 6th International Conference on Human System Interaction (HSI), Sopot, Poland, 6–8 June 2013; pp. 159–166, doi:10.1109/HSI.2013.6577817.
22. Bačíková, M.; Zbuška, M. Towards automated evaluation of domain usability. In Proceedings of the 2015 IEEE 13th International Scientific Conference on Informatics, Poprad, Slovakia, 18–20 November 2015; pp. 41–46, doi:10.1109/Informatics.2015.7377805.
23. Bačíková, M.; Galko, L.; Hvizdová, E. Manual techniques for evaluating domain usability. In Proceedings of the 2017 IEEE 14th International Scientific Conference on Informatics, Poprad, Slovakia, 14–16 November 2017; pp. 24–30, doi:10.1109/INFORMATICS.2017.8327216.
24. Bačíková, M.; Galko, L.; Hvizdová, E. Experimental Design of Metrics for Domain Usability. In Proceedings of the International Conference on Computer-Human Interaction Research and Applications (CHIRA 2017), Funchal, Portugal, 31 October 2017; Volume 1, pp. 118–125, doi:10.5220/0006502501180125.
25. Galko, L.; Bačíková, M. Experiments with automated evaluation of domain usability. In Proceedings of the 2016 9th International Conference on Human System Interactions (HSI), Portsmouth, UK, 6–8 July 2016; pp. 252–258, doi:10.1109/HSI.2016.7529640.
26. Tomoko, N.; Beglar, D. Developing Likert-Scale Questionnaires. In *JALT Conference Proceedings*; Sonda, N., Krause, A., Eds.; JALT: okyo, Japan, 2014; pp. 1–8.
27. Varanda Pereira, M.J.; Fonseca, J.; Henriques, P.R. Ontological approach for DSL development. *Comput. Lang. Syst. Struct.* **2016**, *45*, 35–52, doi:10.1016/j.cl.2015.12.004.
28. Bačíková, M. Domain Analysis of Graphical User Interfaces of Software Systems (extended dissertation abstract). In *Information Sciences and Technologies*; Bulletin of the ACM Slovakia; STU Press: Bratislava, Slovakia, 2014; Volume 6, pp. 17–23.
29. Bačíková, M.; Porubän, J.; Lakatoš, D. Defining Domain Language of Graphical User Interfaces. In Proceedings of the Symposium on Languages Applications and Technologies (SLATE), Porto, Portugal, 20–21 June 2013; pp. 187–202, doi:10.4230/OASIcs.SLATE.2013.187.
30. Vrandečić, D.; Krötzsch, M. Wikidata: A Free Collaborative Knowledgebase. *Commun. ACM* **2014**, *57*, 78–85, doi:10.1145/2629489.
31. Huynh, D.F.; Li, G.; Ding, C.; Huang, Y.; Chai, Y.; Hu, L.; Chen, J. Generating Insightful Connections between Graph Entitites. U.S. Patent 20140280044, 14 July 2020.
32. Lewis, C. *Using the "Thinking-Aloud" Method in Cognitive Interface Design*; Technical Report; IBM, T. J. Watson Research Center: New York, NY, USA, 1982.
33. Kato, T. What "question-asking protocols" can say about the user interface. *Int. J. Man Mach. Stud.* **1986**, *25*, 659–673, doi:10.1016/S0020-7373(86)80080-3.
34. Lund, A.M. Expert Ratings of Usability Maxims. *Ergon. Des. Q. Hum. Factors Appl.* **1997**, *5*, 15–20.
35. Marciniak, J. *Encyclopedia of Software Engineering*, 2nd ed.; Wiley: Chichester, UK, 2002.
36. Nielsen, J. The Use and Misuse of Focus Groups. 1997. Available online: http://www.nngroup.com/articles/focus-groups/ (accessed on 9 August 2021).
37. Stull, E. User Testing. In *UX Fundamentals for Non-UX Professionals*; Apress: New York, NY, USA, 2018; pp. 311–317, doi:10.1007/978-1-4842-3811-0_43.
38. Boehm, B.W.; Brown, J.R.; Lipow, M. Quantitative Evaluation of Software Quality. In Proceedings of the 2nd International Conference on Software Engineering (ICSE '76), San Francisco, CA, USA, 13–15 October 1976; IEEE Computer Society Press: Washington, DC, USA, 1976; pp. 592–605.
39. Nielsen, J.; Mack, R.L. (Eds.) *Usability Inspection Methods*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1994.

40. Wharton, C.; Rieman, J.; Lewis, C.; Polson, P. The Cognitive Walkthrough Method: A Practitioner's Guide. In *Usability Inspection Methods*; John Wiley & Sons, Inc.: Hoboken, NJ, USA, 1994; pp. 105–140.
41. Mahatody, T.; Sagar, M.; Kolski, C. State of the Art on the Cognitive Walkthrough Method, Its Variants and Evolutions. *Int. J. Hum.-Comput. Interact.* **2010**, *26*, 741–785, doi:10.1080/10447311003781409.
42. González, M.P.; Lorés, J.; Granollers, A. Assessing Usability Problems in Latin-American Academic Webpages with Cognitive Walkthroughs and Datamining Techniques. In *Usability and Internationalization. HCI and Culture*; Aykin, N., Ed.; Springer: Berlin/Heidelberg, Germany, 2007; pp. 306–316, doi:10.1007/978-3-540-73287-7_38.
43. Krueger, R.A.; Casey, M.A. *Focus Groups: A Practical Guide for Applied Research*, 5th ed.; SAGE Publications Inc.: Thousand Oaks, CA, USA, 2015.
44. Flanagan, J.C. The critical incident technique. *Psychol. Bull.* **1954**, *51*, 327–358, doi:10.1037/h0061470.
45. Harper, B.D.; Norman, K.L. Improving user satisfaction: The questionnaire for user interaction satisfaction version 5.5. In Proceedings of the 1st Annual Mid-Atlantic Human Factors Conference, Virginia Beach, VA, USA, 1993; pp. 224–228.
46. Tullis, T.S.; Stetson, J.N. A comparison of questionnaires for assessing website usability. In Proceedings of the Usability Professional Association Conference, Minneapolis, MN, USA, 7–11 June 2004; pp. 1–12.
47. Hilbert, D.M.; Redmiles, D.F. Extracting usability information from user interface events. *ACM Comput. Surv.* **2000**, *32*, 384–421, doi:10.1145/371578.371593.
48. Assila, A.; de Oliveira, K.M.; Ezzedine, H. Standardized Usability Questionnaires: Features and Quality Focus. *Electron. J. Comput. Sci. Inf. Technol.* **2016**, *6*, 15–31.
49. Bangor, A.; Kortum, P.; Miller, J. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *J. Usability Stud.* **2009**, *4*, 114–123.
50. McLellan, S.; Muddimer, A.; Peres, S.C. The Effect of Experience on System Usability Scale Ratings. *J. Usability Stud.* **2012**, *7*, 56–67.
51. Brooke, J. SUS: A Retrospective. *J. Usability Stud.* **2013**, *8*, 29–40.
52. John, B.E.; Kieras, D.E. The GOMS Family of User Interface Analysis Techniques: Comparison and Contrast. *ACM Trans. Comput. Hum. Interact.* **1996**, *3*, 320–351, doi:10.1145/235833.236054.
53. Kieras, D. Chapter 31—A Guide to GOMS Model Usability Evaluation using NGOMSL. In *Handbook of Human-Computer Interaction*, 2nd ed.; North-Hollan: Amsterdam, The Netherlands, 1997; pp. 733–766, doi:10.1016/B978-044481862-1.50097-2.
54. Clark, R.E.; Feldon, D.F.; van Merriënboer, J.J.G.; Kenneth, A.Y.; Early, S. Cognitive Task Analysis. In *Handbook of Research on Educational Communications and Technology*; Routledge: London, UK, 2007; Chapter 43, doi:10.4324/9780203880869.ch43.
55. Kincaid, J.P.; Fishburne, R.P.; Rogers, R.L.; Chissom, B.S. *Derivation of New Readability Formulas (Automated Readability Index, Fog Count and Flesch Reading Ease Formula) for Navy Enlisted Personnel*; Technical Report; University of Central Florida: Orlando, FL, USA, 1975.
56. Kincaid, J.P.; McDaniel, W.C. *An Inexpensive Automated Way of Calculating Flesch Reading Ease Scores*; Patient Disclosure Document 031350; US Patient Office: Washington, DC, USA, 1974.
57. Young, R.M.; Green, T.R.G.; Simon, T. Programmable User Models for Predictive Evaluation of Interface Designs. *SIGCHI Bull.* **1989**, *20*, 15–19, doi:10.1145/67450.67453.
58. Porubän, J.; Bačíková, M. *Definition of Computer Languages via User Interfaces*; Technical University of Košice: Košice, Slovakia, 2010; pp. 53–57.
59. Mahajan, R.; Shneiderman, B. Visual and Textual Consistency Checking Tools for Graphical User Interfaces. *IEEE Trans. Softw. Eng.* **1997**, *23*, 722–735, doi:10.1109/32.637386.
60. Artemieva, I.L. Ontology development for domains with complicated structures. In Proceedings of the First International Conference on Knowledge Processing and Data Analysis (KONT'07/KPP'07), Novosibirsk, Russia, 14–16 September 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 184–202, doi:10.1007/978-3-642-22140-8_12.
61. Kleshchev, A.S. How can ontologies contribute to software development? In Proceedings of the First International Conference on Knowledge Processing and Data Analysis (KONT'07/KPP'07), Novosibirsk, Russia, 14–16 September 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 121–135, doi:10.1007/978-3-642-22140-8_8.
62. Gribova, V. A Method of Estimating Usability of a User Interface Based on its Model. *Int. J. Inf. Theor. Appl.* **2007**, *14*, 43–47.
63. Billman, D.; Arsintescucu, L.; Feary, M.; Lee, J.; Smith, A.; Tiwary, R. Benefits of matching domain structure for planning software: The right stuff. In Proceedings of the 2011 Annual Conference on Human Factors in Computing Systems (CHI '11), Vancouver, BC, Canada, 7–12 May 2011; ACM: New York, NY, USA, 2011; pp. 2521–2530, doi:10.1145/1978942.1979311.
64. Tilly, K.; Porkoláb, Z. Automatic classification of semantic user interface services. In Proceedings of the Ontology-Driven Software Engineering (ODiSE'10), Reno, NV, USA, 17–21 October 2010; ACM: New York, NY, USA, 2010; pp. 1–6, doi:10.1145/1937128.1937134.
65. Becker, S.A. A study of web usability for older adults seeking online health resources. *ACM Trans. Comput. Hum. Interact.* **2004**, *11*, 387–406, doi:10.1145/1035575.1035578.
66. Shneiderman, B. Response time and display rate in human performance with computers. *ACM Comput. Surv.* **1984**, *16*, 265–285, doi:10.1145/2514.2517.
67. Eason, K.D. Towards the experimental study of usability. *Behav. Inform. Technol.* **1984**, *3*, 133–143, doi:10.1080/01449298408901744.

68. ISO/IEC-25010. *Systems and Software Engineering—Systems and Software, Quality Requirements and Evaluation (SQuaRE)—System and Software Quality Models*; ISO: Geneva, Switzerland, 2011.
69. Shackel, B. Usability—Context, framework, definition, design and evaluation. *Hum. Factors Inform. Usability* **1991**, *21*, 21–38, doi:10.1016/j.intcom.2009.04.007.
70. Madan, A.; Kumar, S. Usability evaluation methods: A literature review. *Int. J. Eng. Sci. Technol.* **2012**, *4*, 590–599.
71. Kordić, S.; Ristić, S.; Čeliković, M.; Dimitrieski, V.; Luković, I. Reverse Engineering of a Generic Relational Database Schema Into a Domain-Specific Data Model. In Proceedings of the Central European Conference on Information and Intelligent Systems, Varaždin, Croatia, 27–29 September 2017; pp. 19–28.