MDPI

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

Citation: Bačíková, M.; Porubän, J.; Sulír, M.; Chodarev, S.; Steingartner, W.; Madeja, M. Domain Usability Evaluation. *Electronics* **2021**, *1*, 0. https://doi.org/

Received:
Accepted:
Published:

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Submitted to *Electronics* for possible open access publication under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/ 4.0/).

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 brings 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 [1–3].

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 gets better slowly with the introduction of UX courses into the curricula of universities and with the foundation of organizations spreading the word such as SUXA[1]. The more specific 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 across the globe [4–8].

[1] Slovak User Experience Association, https://www.suxa.sk

1.1. Domain Usability

We describe *Domain Usability (DU)* in 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 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 DU. 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. That 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 DU definition, we encourage the reader to see our earlier work [9].

1.2. Problem and Motivation

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

- i-I There are no clear *rules* to design the term structure of an application, so it would correspond with the domain.
- i-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, but they are either too general or not focused on domain usability (DU) as a whole.
- i-III The variety of human thinking, *ambiguity and diversity of natural language* represents an issue to evaluating the correctness of UI terminology.
- i-IV No clear *manual methods* exist for the formal DU evaluation of existing UIs.
- i-V There are no standardized *metrics* to evaluate domain usability.
- i-VI No *automated methods* exist for domain usability evaluation.

We addressed the issues i-I to i-V and partially also i-VI in our previous works:

- We introduced the concept of *DU* and examples to illustrate our definition [9,10] (i-I, i-II).
- To address i-III, we performed a feasibility analysis of approaches for analyzing separate DU aspects [11].
- We proposed and experimentally verified multiple novel manual techniques for DU evaluation [12,13] (i-IV).
- We designed a domain usability metric consisting of five aspects [14] (i-V).
- We proposed a conceptual design and a proof-of-concept implementation of a method for automated evaluation of DU [11] and later presented its preliminary evaluation [15] (i-VI).

The *main contribution* of this paper is to the issue i-VI where we focus on automated evaluation. We would like to summarize and put in 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, 4, and 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 (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 literature.

2. Domain Usability Metrics Design

As we have mentioned, DU is defined by 5 main aspects. In our previous research [14], we tried to confirm or disprove hypothesis *H: All DU aspects have an equal impact on the overall usability and UX*. Several preliminary experiments we performed in the domain of gospel music suggested the invalidity of H [12,13].

We decided to perform two surveys [14] 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.

Suppose that the target UI's DU would be measured formally. To achieve that, we would count the number of all components of the UI that contain any textual information or icons. Then we would analyze all components for any DU issues. Having the number of all application terms n and erroneous terms e, we could determine the percentage of UI's correctness, while 100% would represent the highest DU and 0% would be the lowest. Any component might have multiple DU issues at once (e.g., an unsuitable term and a typo). If this would be the case for all UI components, then the result would be lower than zero, thus we have to limit the resulting 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 can be calculated as follows:

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

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

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

Table 1. Interpretation of the ratings achieved 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

The weights w_x were determined by performing two surveys, first a general one with 73 respondents of age between 17 and 44 years and then a domain-specific one with 26 gospel singers and guitar players of age between 15 and 44 years. The general group consisted of general computer users, the domain-specific group was selected from the participants of previous DU experimentation with manual DU evaluation techniques [12,13], as they experienced DU issues first-hand.

The questionnaires were composed of two parts. The first part contained 5 questions, in which the task of the participants was to rate particular DU aspects based on examples. Each of the 5 questions was aimed at one DU aspect in form of visual examples — screenshots from domain-specific UIs. Each example contained a particular DU issue (corresponding to the particular DU aspect) and, to be sure the participants understood the issue, a supplementary explanation was provided. The task of the participants was to study the provided examples and rate the importance of a particular DU aspect by a number from the 1-5 Likert scale [16] (with 1 being the least important). [2]

In the second part, the task was to order the five aspects of DU from the least to most important.

The results of both surveys have shown that DU has a significant impact on usability and UX. By merging the first and the second part of the domain-specific questionnaire, we gained overall weights of the particular aspects. After we substitute the weights w_x (where $x \in \{dc, ds, c, eb, l\}$) in the formula (2) as follows:

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

then the *formula (1) represents the metric of DU with the consideration of its aspects* and with the result in percentage.

The sum of multiplies between weights and errors is divided by the number of all components n. All components can contain domain information such as terms, icons, or tooltip descriptions. 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.

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 will 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

[2] The questionnaires given to the general and domain-specific group can be found at: http://hornad.fei.tuke.sk/~bacikova/domain-usability/surveys. Details about the surveys can be found in [14].

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 there exists an older user interface used in some specific domain for ages. However, despite this UI is usable, the used technology got 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 older 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, (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, or two separate UIs from the same domain developed by different vendors. But when the UI is freshly designed specifically for the particular business area, there is usually only one UI available. In this case, there are two options to consider:

- using a reference ontology representing the domain language to be compared to,
- using ontological dictionaries or 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% 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,

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

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

domain-specific ontologies [17] would grow both in 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).

From the linguistic point of view, *Favorite color* is a hypernym of the individual color values (or vice versa, 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 [15] 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

To check the completeness of language alternatives and correctness of the language terms used in the UI, standard 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 [18] and [19].

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)[3] is a method for extracting domain information from user interfaces of applications. Its implementation currently supports Java (Swing), HTML, and Windows applications (*.exe).

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 graph of nodes — a domain model — where each node represents a term (concept) in the source user interface. Each such node contains information about:

- UI *component* — the instance of the component from which the information about the term was extracted,
- *name* — the label displayed on the component,
- *description* — the component's tooltip if it is present,
- *relation* to other terms — mutual (non-)exclusivity,
- *type* of input data — in case the representing component is an input field, the type can be string, number, date, boolean, or enumeration,
- *icon* (if present),
- *parent* term (usually a hypernym or holonym),
- *child* terms (usually hyponyms or meronyms),
- *category of the representing component* — either functional, informative, textual, container (logically/graphically grouping), or custom.

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, 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 will contain the term *Person* with the following child terms: *name, surname, date of birth, status*, and *favorite color*. If the *status* is represented by radio buttons, then this term will have the type *enumeration* and values *"Single"*, *"Married"*, *"Divorced"* and *"Widowed"*, which are mutually exclusive. If

[3] https://git.kpi.fei.tuke.sk/michaela.bacikova/DEAL

Listing 2. The domain model of the Person form

```
domain: 'Person' {children}: [
   name {string}
   surname {string}
   dateOfBirth {date}
   status {mutually-exclusive}
       {enumeration}[
          Single,
          Married,
          Divorced,
          Widowed
       ]
   favoriteColor {mutually-not-exclusive}
       {children}: [
      red
      yellow
      blue
      green
   ]
   OK
   Close
   Reset
]
```

favorite color is represented by check-box components, then the term *Favorite color* will contain all their values as *child terms*, and they will be in the mutually not exclusive relation. If there are any functional components, e.g., menu items or buttons (such as *OK, Close, Reset*), their terms will become children of the *Person* term. This way, a hierarchy of all terms extracted from the user interface is created, and each new window or a dialog represents a new graph of terms.

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

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. For that, we will use a new reference ontology that will list domain-independent general-purpose terms commonly used in applications, their alternatives and 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[4]. According to this ontology, we implemented an additional module into DEAL, which is able to filter out such terms from the domain model automatically 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] The *General Application Terms Ontology* can be found at https://bit.ly/2R6bm6p

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

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 that a component is 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.

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. 30 web applications were analyzed, in which we focused on registration forms, login forms, and their client applications. Based on the analyzed data we identified the 5 most common form stereotypes shown in Figure 1.

1. LEFT — most common where the text labels are located left to the form component.

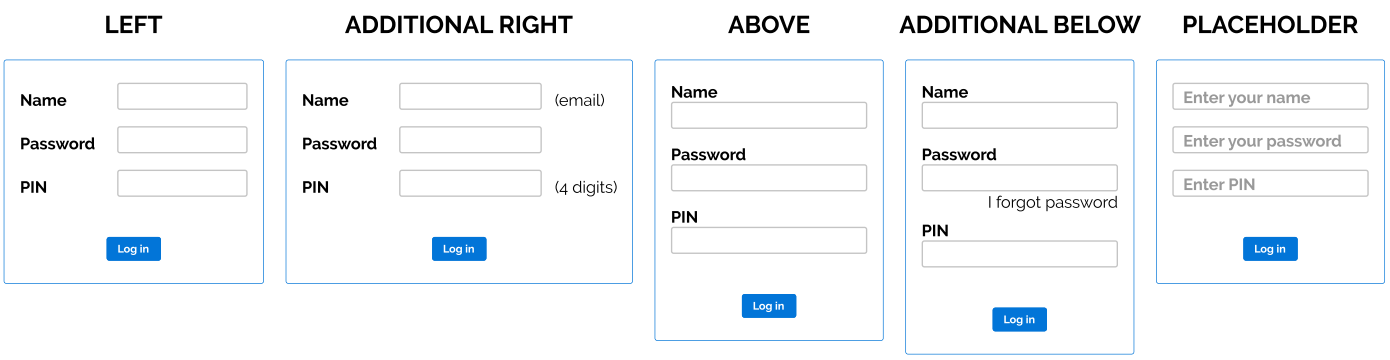

Figure 1. The most frequent form stereotypes

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.

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., 5 components have labels on their left and 5 other components have labels on their right), then the form stereotype cannot be identified and is marked as MIXED.

The target 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[5] user interface.

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

[5] https://sourceforge.net/projects/openrocket/

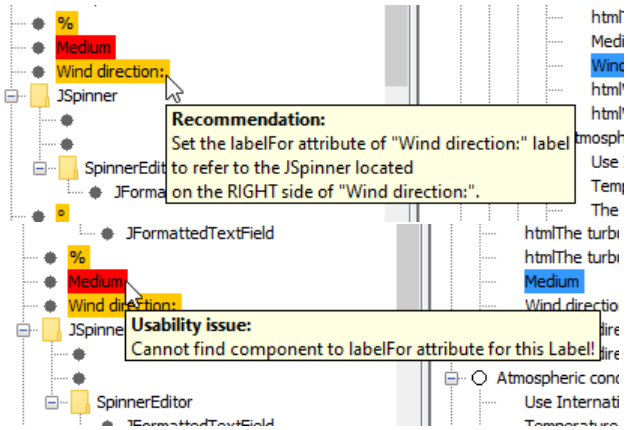

Figure 2. DEAL — Example of a recommendation indicating the successful recognition of a form stereotype and an issue because of a missing *labelFor* attribute. The domain model shown in this figure was extracted from OpenRocket.

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 will describe each of the methods in more detail (except the form analyzer that was already explained in section 4.3). We will 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.

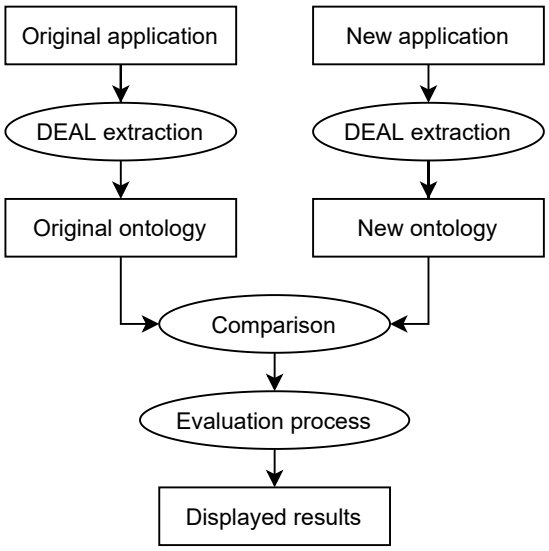

Figure 3. ADUE method — a high-level overview of the ontological evaluation process with two ontology versions. Processes are marked as ellipses, data as rectangles.

- *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 representing component: a button, label, text field, check box, radio button, etc.
- The term's *parent* term.
- *Children*, i.e., the child terms.

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 those changes to be able to check their correctness.
- Removed elements — might or might not introduce an issue and feature depletion. It depends on the evaluator whether the removal was justified.
- Changed elements — we consider *correctly* and *incorrectly* changed elements, while incorrect changes are considered a usability issue. Incorrect changes include the illogical component type 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* (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 Google 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.

During the ontological evaluation, if the parent of any child term list is not found in the list of possible parent terms returned by *OD&GS*, a warning is shown and probable parent terms returned by the *OD&GS* algorithm are recommended as replacements.

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[6] 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. After that, we filter the results to only direct hypernyms to ensure higher accuracy. The result for each child word list is a list of recommended parent terms.

5.2.2. Urban Dictionary

Urban Dictionary[7] is a crowdsourced dictionary. For each queried word it returns 7 definitions sorted according to the votes of the Urban Dictionary users. These definitions are collected for each query and the frequency of their occurrence in the result is searched. For every word list, the result is a list of possible parent terms sorted by the highest occurrence frequency.

5.2.3. Google web search

While Google is not a linguistic tool, the current state of its multi-layered semantic network — *Knowledge Graph* [20,21] — enables to gain 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.

[6] https://wordnet.princeton.edu
[7] http://www.urbandictionary.com

Based on our tests, Google is the most precise of all three sources. 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 the following 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 analyze the returned HTML pages and count the most common words. The probability of each result word is based on the frequency of its occurrence. We only accept words of the same word class as 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.

All terms in the domain model are checked for correct grammar. If some word is not found in the dictionary, the algorithm tries to translate it to other languages. If a translation is found, the term with its translated versions are saved to a list of recommendations. Otherwise, a recommendation for correction is displayed and possible corrections are recommended in a way similar to modern text editors. In the end, the evaluator is presented with a list of correction recommendations.

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 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 was found for some functional component, then two types of results are displayed to the evaluator:

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

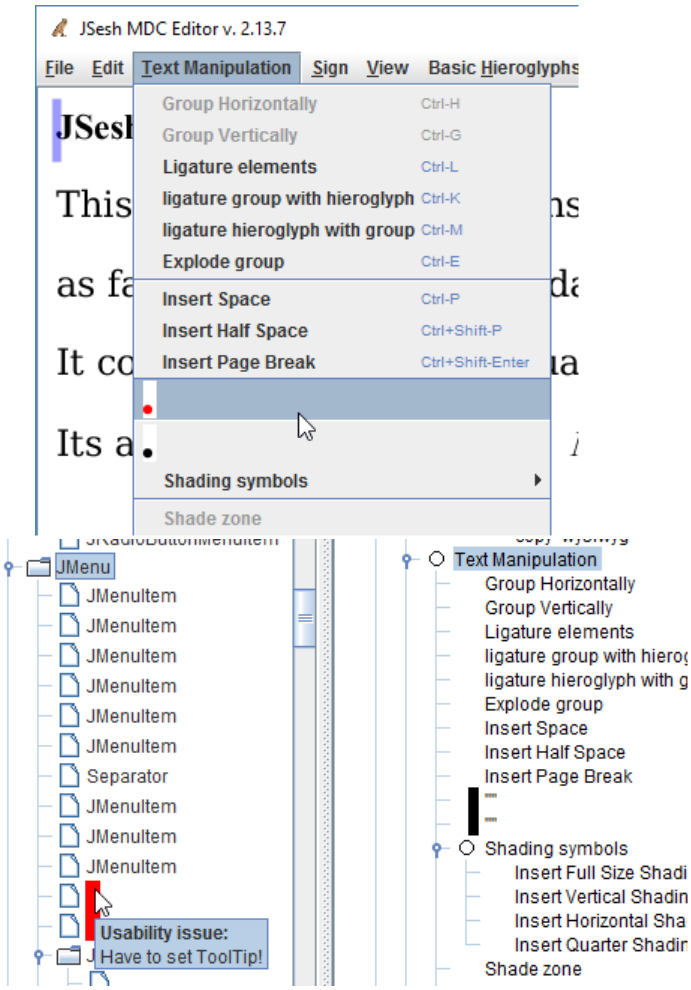

Figure 4. Example of JSesh menu items both without a tooltip and label (top) and a usability issue reported to the user (bottom).

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.

6. Prototype

We implemented all processes mentioned in section 5 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 Figure 2 and 4.

The whole process is triggered by a corresponding menu item in DEAL. In general, DU errors are marked red, and recommendations are marked 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.

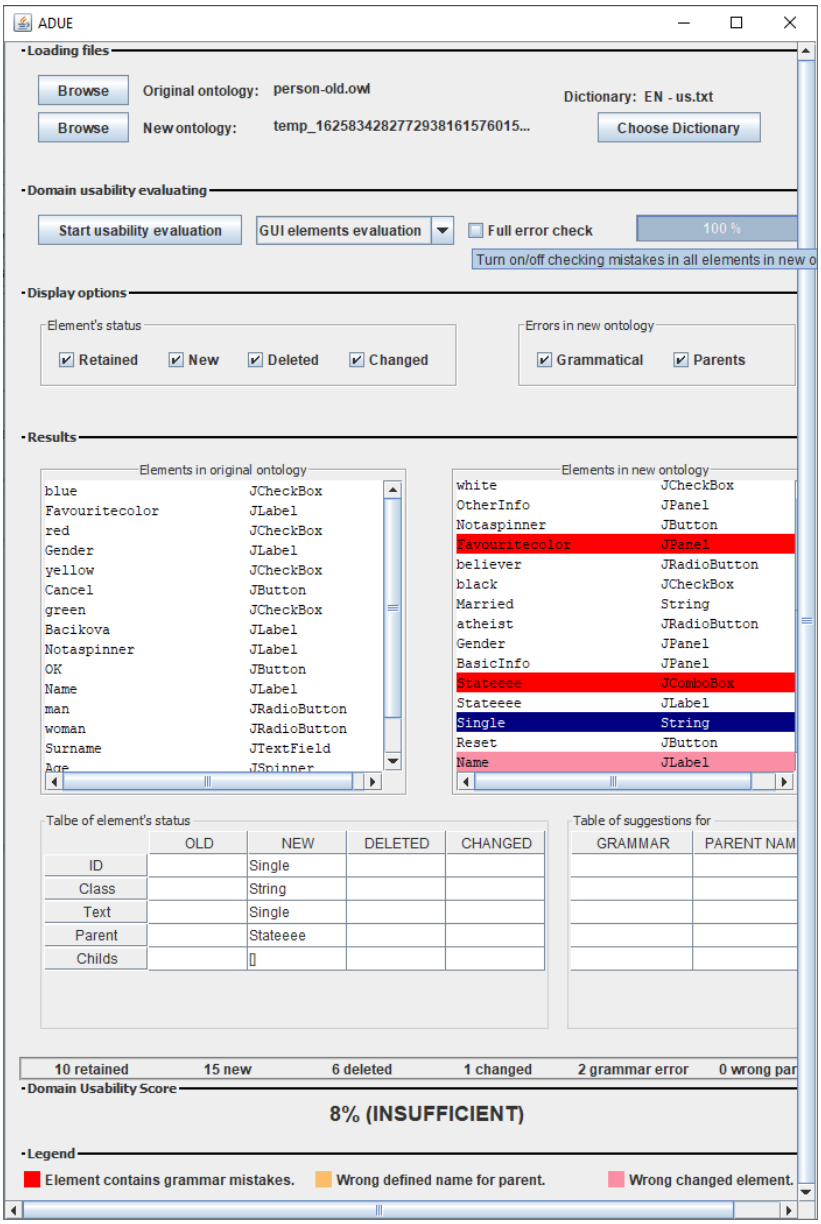

Figure 5. The ADUE evaluation tool displaying the results from comparing two sample applications.

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.

Grammar errors in the new ontology are marked red. An incorrectly defined parent term (hypernym, holonym) is marked orange, and a recommendation is displayed in a tooltip. In addition, if a term in the older UI version was represented by a textual component and it is of another type in the newer UI version (e.g., a button), then it is marked by pink color. If components change their type to other types (textual, informative, functional, etc.), this change might represent a DU issue. For example, if a "Cancel" term was represented by a button in the old UI, it should not be a text field in the new one. The evaluator can also see the retained, new, deleted, and changed terms in both ontologies. 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.

6.1. 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[8]. As a dictionary in the grammar evaluation, we used the US English dictionary bundled with the *OpenOffice* text editor[9]. We chose this dictionary because of the simplicity: it is a simple text file with words separated by newline characters, and 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[10] was used. After identifying a typo in a text, *JAZZY* returns multiple replacement recommendations of the incorrect word. For NLP with respect to the ontological dictionaries and Google search evaluation, we used the *Apache OpenNLP* library[11], which can identify the word classes such as verbs, nouns, or adjectives. To query the WordNet dictionary, the *JAWS* library[12] was used. Urban Dictionary does not support any public API. Therefore, we used 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 [13].

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 to use 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.

[8] https://github.com/owlcs/owlapi/wiki
[9] https://www.openoffice.org
[10] http://jazzy.sourceforge.net
[11] https://opennlp.apache.org
[12] https://github.com/jaytaylor/jaws
[13] https://developers.google.com/custom-search/v1/overview

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). 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.

Namely, 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, GDL Editor 0.95.

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 the formula (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.

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 point out the fact that common applications have DU issues concerning tooltips and developers are not aware that they are necessary.

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. Since we were using the US English dictionary and some applications used British English, there was a small number of incorrectly identified grammar errors. Aside from them, the tool identified grammar errors correctly and provided good recommendations for corrections.

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 both 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 FreeMind/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.

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

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 seconds to 5 minutes and 6 seconds, with a mean of 1 minute and 7 seconds. This means that automated domain usability evaluation could be potentially performed in a variety of contexts, including continuous integration (CI) builds.

7.3. Threats to Validity

Regarding the internal validity, some detected issues might have been false positives. However, to mitigate this threat, we always manually verified a subset of the results to ensure

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.

7.4. 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, 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

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 [22]. 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*, *question-asking protocol*, *performance measurement*, or *log file analysis* [23,24] can be easily altered to focus on domain dictionary just by changing the questions or tasks included in the process to get the desired outcome. If there is a recording output, it can be analyzed with respect to DU. Informal or structured *interviews* and *focus groups* [25] might also be directed on domain user dictionary 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, (iii) and if not, what would they use instead.

8.2. User Testing Techniques

There are multiple types of *user testing* 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.
- Subjected 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 proposed by Boehm et al. [26], the expert in usability usually performs the inspection of guidelines, heuristic rules, product consistency, or standards compliance of a prototype.

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

- Guideline review, cognitive walk-trough techniques, heuristic evaluation techniques, formal usability inspection, and standards inspection: an expert performs the check focusing on domain terminology, consistency, errors.
- To get the best results on the aforementioned techniques, the expert needs to be a domain expert.
- Another option is a pluralistic walk-trough 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.

8.4. Inquiry

Inquiry techniques are those that focus on user feedback. They include Focus Groups [27], Interviews and Surveys [28], Questionnaires [29,30], and others. There are two categories of inquiry techniques we would like to focus on: in-system user feedback and surveys & questionnaires.

8.4.1. In-System User Feedback

General techniques are based on the user sending feedback in a form of recorded events [31], 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 in 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, the best is to comment it directly in the target UI because of context.
- For both possibilities, the users can assess the *appropriateness* of a particular term using the approach by Isohella and Nissila [7].

8.4.2. Surveys & Questionnaires

Most of the common standard usability surveys and questionnaires are too general defined to be usable for DU evaluation. That was the primary reason for our proposal of

a novel SDUS (System Domain Usability Scale) technique in 2018 [13]. SDUS is based on the common standardized System Usability Scale (SUS) [32,33], 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 are positive and even questions are negative statements. The answers are in the standard 5-degree 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 [34].

8.5. Analytical Modeling Techniques

The goal of *GOMS (Goals, operations, methods, selection rules)* analysis [35–37] 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 [7] per system might provide a good view of the system improvement since the last prototype.

In *cognitive task analysis* [38], 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 is the domain content of the UI, 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[14]. Readability Score evaluates the given text or a URL and determines multiple reading complexity indices including Flesch-Kincaid [39,40], Keyword Density, and similar.

The goal of *Programmable User Models* [41] 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 [42]. 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[15] 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

So far, 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 [43] 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.

[14] http://readable.com

[15] Protractor end-to-end testing framework: https://www.protractortest.org

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 [44], Kleshchev [45], and Gribova [46], 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. [47]. 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 the new, as the new application was better in domain-specific terminology structure.
- *User interface semantics, Ambiguity* — Tilly and Porkoláb [48] propose to use *semantic UIs* (SUI) to solve the problem of the ambiguity of UI terminology. The core of SUI is a general ontology which 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 [49], Kincaid et al. [39], and Mahajan and Shneiderman [50] stress that the complexity of the textual content should not be too high because the application will be 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. [51] 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 [31] 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 [52] refers to the *knowledge* aspect of UI design representing the knowledge that the user applies to the task and may be appropriate or inappropriate. In general, also the *task match* attribute of Eason's definition refers to processes mapping but does not explicitly target the mapping of specific domain tasks.
- *Appropriateness recognizability* — defined by ISO/IEC-25010 [53] 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*. But again, the term appropriateness recognizability does not specifically refer to the target domain match.
- Other definitions such as ISO-9241-11 [54] or defined by Nielsen [1], Shackel [55], and others [56] are rather too general but we do not exclude DU being a subset of them.

9.2. Consistency

Among other aspects, Badashian et al. [51] stress the importance of *consistency* in usable UIs. The survey by Ivory and Hearst [22] contains a wide list of automatic usability methods and tools. From over 100 works, only Mahajan and Shneiderman [43] 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 are describing the same functionality, or not.

9.3. World Language, Language Barriers, Errors

Besides complexity, Becker [49] also deals with the *translation* of UIs, which corresponds to the *world language* DU aspect. In the area of web accessibility [57], 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 [7] 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. Conclusion

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. 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 [58] 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. The General Application Terms Ontology can be found at https://bit.ly/2R6bm6p.

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, 2013.
3. Morville, P. User Experience Design, 2004.
4. Chilana, P.K.; Wobbrock, J.O.; Ko, A.J. Understanding Usability Practices in Complex Domains. Proc. of the SIGCHI Conf. on Human Factors in Comp. Syst.; ACM: NY, USA, 2010; CHI '10, pp. 2337–2346. doi:10.1145/1753326.1753678.
5. Chilana, P.K.; Wobbrock, J.O.; Ko, A.J. Understanding Usability Practices in Complex Domains. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems; ACM: New York, NY, USA, 2010; CHI '10, pp. 2337–2346. doi:10.1145/1753326.1753678.
6. Gulliksen, J. Designing for Usability - Domain Specific Human-Computer Interfaces in Working Life. *Comprehensive Summaries of Uppsala Dissertations from the Faculty of Science & Technology* **1996**, p. 28.
7. Isohella, S.; Nissila, N. Connecting usability with terminology: Achieving usability by using appropriate terms. IEEE IPCC'15, 2015, pp. 1–5.
8. Lanthaler, M.; Gütl, C. Model Your Application Domain, Not Your JSON Structures. Proc. 22Nd Int. Conf. on WWW '13 Companion; ACM: NY, USA, 2013; pp. 1415–1420. doi:10.1145/2487788.2488184.
9. Bačíková, M.; Porubän, J., Domain Usability, User's Perception. In *Human-Computer Systems Interaction: Backgrounds and Applications 3*; Springer International Publishing: Cham, 2014; pp. 15–26. doi:10.1007/978-3-319-08491-6_2.
10. Bačíková, M.; Porubän, J. Ergonomic vs. domain usability of user interfaces. 2013 The 6th International Conference on Human System Interaction (HSI), 2013, pp. 159–166. doi:10.1109/HSI.2013.6577817.
11. Bačíková, M.; Zbuška, M. Towards automated evaluation of domain usability. 2015 IEEE 13th International Scientific Conference on Informatics, 2015, pp. 41–46.
12. Bačíková, M.; Galko, L.; Hvizdová, E. Manual techniques for evaluating domain usability. 2017 IEEE 14th International Scientific Conference on Informatics, 2017, pp. 24–30.
13. Bačíková, M.; Galko, L. The design of manual domain usability evaluation techniques. *Open Computer Science* **2018**, *8*, 51–67. doi:10.1515/comp-2018-0005.
14. Bačíková, M.; Galko, L.; Hvizdová, E. Experimental Design of Metrics for Domain Usability. Proceedings of the International Conference on Computer-Human Interaction Research and Applications - Volume 1: CHIRA. INSTICC, SciTePress, 2017, pp. 118–125. doi:10.5220/0006502501180125.
15. Galko, L.; Bačíková, M. Experiments with automated evaluation of domain usability. 2016 9th International Conference on Human System Interactions (HSI), 2016, pp. 252–258. doi:10.1109/HSI.2016.7529640.
16. Tomoko, N.; Beglar, D. Developing Likert-Scale Questionnaires. N. Sonda, A. Krause (Eds.), JALT Conference Proceedings. JALT, 2014, pp. 1–8.
17. Varanda Pereira, M.J.; Fonseca, J.; Henriques, P.R. Ontological approach for DSL development. *Computer Languages, Systems & Structures* **2016**, *45*, 35–52. doi:10.1016/j.cl.2015.12.004.
18. Bačíková, M. Domain Analysis of Graphical User Interfaces of Software Systems (extended dissertation abstract). *Information Sciences and Technologies, Bulletin of the ACM Slovakia Chapter, Extended abstracts of Dissertations* **2014**, *6*, 17–23.
19. Bačíková, M.; Porubän, J.; Lakatoš, D. Defining Domain Language of Graphical User Interfaces. Symposium on Languages Applications and Technologies (SLATE), 2013, pp. 187–202.
20. Vrandečić, D.; Krötzsch, M. Wikidata: A Free Collaborative Knowledgebase. *Commun. ACM* **2014**, *57*, 78–85. doi:10.1145/2629489.
21. Huynh, D.F.; Li, G.; Ding, C.; Huang, Y.; Chai, Y.; Hu, L.; Chen, J. Generating Insightful Connections between Graph Entitites. US patent 20140280044, Google Inc., 2013.
22. 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.
23. Lund, A.M. Expert Ratings of Usability Maxims. *Ergonomics in Design: The Quarterly of Human Factors Application* **1997**, *5*, 15–20.
24. Marciniak, J. *Encyclopedia of software Engineering, 2, 2nd edn*; Chichester: Wiley, 2002.
25. Nielsen, J. The Use and Misuse of Focus Groups, 1997.
26. Boehm, B.W.; Brown, J.R.; Lipow, M. Quantitative Evaluation of Software Quality. Proceedings of the 2nd International Conference on Software Engineering; IEEE Computer Society Press: Washington, DC, USA, 1976; ICSE '76, pp. 592–605.
27. Krueger, R.A.; Casey, M.A. *Focus Groups: A Practical Guide for Applied Research (5th edition)*; SAGE Publications Inc., 2015.
28. Flanagan, J.C. The critical incident technique. *Psychological Bulletin* **1954**, *51*, 327–358. doi:10.1037/h0061470.
29. Harper, B.D.; Norman, K.L. Improving user satisfaction: The questionnaire for user interaction satisfaction version 5.5. Proceedings of the 1st Annual Mid-Atlantic Human Factors Conference, 1993, pp. 224–228.
30. Tullis, T.S.; Stetson, J.N. A comparison of questionnaires for assessing website usability. Usability Professional Association Conference. Citeseer, 2004, pp. 1–12.
31. 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.
32. Bangor, A.; Kortum, P.; Miller, J. Determining What Individual SUS Scores Mean: Adding an Adjective Rating Scale. *J. Usability Studies* **2009**, *4*, 114–123.
33. McLellan, S.; Muddimer, A.; Peres, S.C. The Effect of Experience on System Usability Scale Ratings. *J. Usability Studies* **2012**, *7*, 56–67.
34. Brooke, J. SUS: A Retrospective. *J. Usability Studies* **2013**, *8*, 29–40.

35. 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.
36. John, B.; Kieras, D. The GOMS family of user interface analysis techniques: Comparison and contrast. *ACM Transactions on Computer-Human Interaction* **1996**, *3*, 320–351.
37. Kieras, D. Chapter 31 - A Guide to GOMS Model Usability Evaluation using NGOMSL. In *Handbook of Human-Computer Interaction (Second Edition)*; North-Hollan, 1997; pp. 733 – 766. doi:10.1016/B978-044481862-1.50097-2.
38. 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, 2007; chapter chapter 43. doi:10.4324/9780203880869.ch43.
39. 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, 1975.
40. Kincaid, J.P.; W.C., M. An inexpensive automated way of calculating Flesch Reading Ease scores. Patient Disclosure Document 031350. Us Patient Office, Washington, DC, 1974.
41. 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.
42. Porubän, J.; Bačíková, M. Definition of Computer Languages via User Interfaces. FEEI 2010 : Electrical Engineering and Informatics : Proceeding of the FEEI of the TU of Košice. Technical University of Košice, 2010, pp. 53–57.
43. 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.
44. Artemieva, I.L. Ontology development for domains with complicated structures. Proceedings of the First international conference on Knowledge processing and data analysis; Springer-Verlag: Berlin, Heidelberg, 2011; KONT'07/KPP'07, pp. 184–202.
45. Kleshchev, A.S. How can ontologies contribute to software development? Proceedings of the First international conference on Knowledge processing and data analysis; Springer-Verlag: Berlin, Heidelberg, 2011; KONT'07/KPP'07, pp. 121–135.
46. Gribova, V. A Method of Estimating Usability of a User Interface Based on its Model. *International Journal "Information Theories & Applications"* **2007**, *14*, 43–47.
47. Billman, D.; Arsintescucu, L.; Feary, M.; Lee, J.; Smith, A.; Tiwary, R. Benefits of matching domain structure for planning software: the right stuff. Proceedings of the 2011 annual conference on Human factors in computing systems; ACM: New York, NY, USA, 2011; CHI '11, pp. 2521–2530. doi:10.1145/1978942.1979311.
48. Tilly, K.; Porkoláb, Z. Automatic classification of semantic user interface services. Ontology-Driven Software Engineering; ACM: New York, NY, USA, 2010; ODiSE'10, pp. 6:1–6:6. doi:10.1145/1937128.1937134.
49. 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.
50. Shneiderman, B. Response time and display rate in human performance with computers. *ACM Comput. Surv.* **1984**, *16*, 265–285. doi:10.1145/2514.2517.
51. Badashian, A.S.; Mahdavi, M.; Pourshirmohammadi, A.; nejad, M.M. Fundamental Usability Guidelines for User Interface Design. Proceedings of the 2008 International Conference on Computational Sciences and Its Applications; IEEE Computer Society: Washington, DC, USA, 2008; ICCSA '08, pp. 106–113. doi:10.1109/ICCSA.2008.45.
52. Eason, K.D. Towards the experimental study of usability. *Behaviour & Information Technology* **1984**, *3*, 133–143 . doi:10.1080/01449298408901744.
53. ISO/IEC-25010. Systems and software engineering - Systems and software, Quality Requirements and Evaluation (SQuaRE) - System and software quality models, 2011.
54. ISO-9241-11. Ergonomics of human-system interaction - Part 11: Usability: Definitions and concepts, 2018.
55. Shackel, B. Usability–Context, framework, definition, design and evaluation. *Human Factors for Informatics Usability* **1991**, pp. 21–38.
56. Madan, A.; Kumar, S. Usability evaluation methods: a literature review. *International Journal of Engineering Science and Technology* **2012**, *4*.
57. W3C. Web Content Accessibility Guidelines (WCAG) 2.0, part 3 about understandability, 2008.
58. Kordić, S.; Ristić, S.; Čeliković, M.; Dimitrieski, V.; Luković, I. Reverse Engineering of a Generic Relational Database Schema Into a Domain-Specific Data Model. Central European Conference on Information and Intelligent Systems, 2017, pp. 19–28.