



OPEN ACCESS

EDITED BY

Stamatios Papadakis,
University of Crete, Greece

REVIEWED BY

Antti Luoto,
Tampere University, Finland
Bert Zwaneveld,
Open University of the Netherlands,
Netherlands

*CORRESPONDENCE

Mária Csernoch
✉ csernoch.maria@inf.unideb.hu

SPECIALTY SECTION

This article was submitted to
Digital Learning Innovations,
a section of the journal
Frontiers in Education

RECEIVED 31 May 2022

ACCEPTED 20 September 2022

PUBLISHED 15 March 2023

CITATION

Nagy K and Csernoch M (2023) Pre-testing
erroneous text-based documents: Logging
end-user activities.
Front. Educ. 7:958635.
doi: 10.3389/feduc.2022.958635

COPYRIGHT

© 2023 Nagy and Csernoch. This is an
open-access article distributed under the terms
of the [Creative Commons Attribution License
\(CC BY\)](https://creativecommons.org/licenses/by/4.0/). The use, distribution or reproduction in
other forums is permitted, provided the original
author(s) and the copyright owner(s) are
credited and that the original publication in this
journal is cited, in accordance with accepted
academic practice. No use, distribution or
reproduction is permitted which does not
comply with these terms.

Pre-testing erroneous text-based documents: Logging end-user activities

Keve Nagy and Mária Csernoch*

Faculty of Informatics, University of Debrecen, Debrecen, Hungary

An extremely high number of erroneous text-based documents are in circulation, being multiplied, serving as samples both in the office world and in education. Creating, editing, and modifying these documents generates a serious financial loss when both human and machine resources are considered. Our research team developed an application and a testing method for building up an objective measurement system, in order to see the rate of loss in the handling of erroneous documents. In the pre-testing period, the error factor of a sample text was set up based on the logged activities of an expert researcher. It was found that first level modifications require about five times more human and machine resources in a short, one-paragraph text burdened with layout errors than in its properly formatted version. Further testing is required to find out how demanding longer texts and more serious modifications are, but it is already obvious that erroneous text-editing, and the lack of fundamental computational thinking skills involve unnecessary use of our resources.

KEYWORDS

end-user computing, erroneous text-based documents, computational thinking skills, logging end-user activities, sunk cost fallacy, Dunning-Kruger Effect

1. Introduction

Many believe that handling natural language texts with computers is a straightforward procedure. Meanwhile, researchers of this field within social sciences already know that such thoughts are only a wishful daydream. And, once we wake up from it, we are faced with the most challenging problem of the entire computer technology industry, namely: end-users.

What is an end-user? Who are these end-users? What roles do they play in the digital world? What are their initiatives (if there are any)? What are their expectations (if there are any)? On what bases are their expectations rooted? People asking these questions may initially feel lucky, because short definitions revealing the characteristics officially assigned to end-users exist. Those, however, do not provide the practical essence of end-users in reality, who tend to weaponize digital tools for creating, modifying, and masterfully messing up (bricolage) natural language digital texts primarily in word processing and presentation applications.

“An end user is the person that a software program or hardware device is designed for. [...] To simplify, the end user is the person who uses the software or hardware after it has been fully developed, marketed, and installed” (TechTerms, 2021).

“end user: the person ultimately intended to use a product, as opposed to people involved in developing or marketing it” (Downing et al., 2017).

Rigdon uses the term without defining it and without paying attention to its consistent spelling (both the “end user” and the “end-user” forms are used in the dictionary, arbitrarily) (Rigdon, 2016). The same is true for the original [Microsoft Language Portal \(2021\)](#).

Recent studies clearly prove that erroneous digital texts do not primarily originate from the shortcomings of software solutions – e.g., machine translation, spell checker, character recognition, and lemmatization – but arrive from the general population, the end-users (Ben-Ari and Yeshno, 2006; Csernoch, 2010; Ben-Ari, 2015; [EuSpRIG Horror Stories, 2022](#)), and is due to their lack of computational thinking skills, which should be as fundamental nowadays as the 3Rs (Reading, wRiting, aRithmetic) (Wing, 2006). End-users do not fulfill their dictionary-assigned role because they are not able to efficiently and effectively use the computer programs designed to serve them – if they can use them at all. The problem is not that they cannot use these applications; the problem is when they firmly believe that they can use these applications properly.

Millions of maltreated, erroneous digital texts are born day after day (Csernoch, 2009), often created by confident end-users. The most ignorant and aggressive end-users proudly publish these on the internet, in the belief that the documents are correct. Teachers with the same attitude and dubious digital education can do more harm than good in schools [e.g., (Wood, 2007; Correy, 2011; Gareth, 2017; [National Center for Computing Education, 2019](#); [EA SEND Implementation Team, 2021](#); [GCFGlobal, 2022](#); [Goodwin, 2022](#); [McKinney, 2022](#); [tabaks, 2022](#); [The Skills Factory, 2022](#))] by presenting these documents as examples of high-quality work, and at times discourage students who try to do things the right way. In these cases the Dunning-Kruger Effect (DKE) (Kruger and Dunning, 1999) and the sunk cost fallacy (SCF) (Kahneman, 2011) show their impact.

Considering both human and computer resources, the poor quality digital texts made by overconfident DKE-end-users and SCF victims cause serious financial losses. The question repeatedly arises, how can we convince people that their text-treatment practice, habits, concepts, and approaches do more harm than good? Even fully developed pieces of software lose their power by misuse (Panko, 1998, 2013; Csernoch and Dani, 2022; [EuSpRIG Horror Stories, 2022](#); [Sebestyén et al., 2022](#)). On the one hand, scientific papers presented on this subject reach neither the target population (Ben-Ari and Yeshno, 2006; Wing, 2006; Csernoch, 2009, 2010; Ben-Ari, 2015; [EuSpRIG Horror Stories, 2022](#)) nor the appropriate segment of education (Malmi et al., 2019, 2022). On the other, most of the researchers who are aware of this problem and publish studies on its severity, face difficulties, as their results do not change the behavior of those who produce these documents (Malmi et al., 2019, 2022).

1.1. Detecting end-user activities

We have decided to investigate this problem with a different approach: studying the input activities of the end-users in digital texts of various error-levels. Our hypothesis is that both creating and updating a properly structured text (Csernoch, 2009, 2017) require significantly fewer atomic end-user actions – defined as first level

tasks in [Hankiewicz and Butlewski \(2014\)](#) – than editing an erroneous form of the same content with identical or very similar printed appearance. Consequently, when end-user activities do not need to include anything other than entering the intended content, both human and computer resources can be used more economically. In the case of properly edited texts, less frustration, time, and computer use are required to construct and modify documents, which would be the ultimate goal of end-user education and activities.

In this particular field, objective measurement requires that researchers know exactly how their subjects solve the given problems, what actions they apply to the content, and in precisely what order. This demands that one tracks every input action of the user, hence we need to log all the keypresses and mouse clicks our subjects initiate.

1.2. Search for the appropriate tool

In advance to our research study, we examined a handful of existing software solutions, and found that none of them were suitable for our complex purposes. Regardless of where we looked, considered tools always fell into either of the following categories.

- Macro-recorder: records keyboard and mouse actions, and allowing such series of actions to be replayed on demand. A few may even provide a browsable list of the recorded events but they do not allow analysis or processing of the logged events, because the format of the recordings is proprietary.
- Event-handler: allows to define keypress, key-combination, or mouse-click events to be monitored, and executes custom instructions (*in our case, just logging them would have been sufficient*) when they occur. However, it only handles input events that we specifically pre-programmed. Unfortunately, this is exactly what cannot be predicted. We do not know all the actions our end-users will perform, and expecting to pre-program every possible input combination is not realistic.
- Password-grabber: most are for stealing passwords, and some are generally designed to collect text entered into any input field. Hence, they focus on the resulting string-literal and do not capture how the symbols got generated. Take the semicolon as an example. On a UK or US keyboard, this is a single key press: the button on the right side of the L key. On a Dutch or Icelandic keyboard you press Shift + Comma on the key to the right of M, while on a Hungarian keyboard you press AltGr + Comma on that same key.

We could not find any available software that could clearly present all the atomic input-device activities an end-user carries out while they are modifying a digital text with previously set up requirements. Consequently, we were left with no other choice than to create our own tool.

1.3. Software requirements

Our aim was a specialized user input tracking software with the following criteria:

- Logs in sequential order what keyboard keys the user presses.

- Logs when modifier keys – left and right Alt, Ctrl, Shift, Win – are pressed and released.
- Logs when state-changing keys (switches) – Num Lock, Caps Lock, Scroll Lock – are turned on or off.
- Monitors mouse actions, and logs what is clicked and when.
- Logs in what application and for which document the input is performed.
- Does not require access to the source code of any application.
- Needs no patching, the officially distributed binaries of the Office Suite can be used.
- Works equally well with Microsoft Office, LibreOffice, and internet browser embedded office applications like Google Docs, Google Sheets or Word OnLine, Excel OnLine.
- The log is a human readable text file.
- The log allows post-processing or analysis in an automated way.

1.4. Adjusting expectations

The greatest advantage of developing your own software tool is that you can make a custom application that can do everything you want, the way you want. Yes, well ... not so much. While we managed to achieve most of what we desired, some expectations turned out to be quite unrealistic and needed to be adjusted.

The best example of this was capturing mouse actions. We thought it would be practical to include in our logfile what buttons or menu options are clicked on and what text portions are highlighted, etc. However, the mouse is a very dumb device, and you cannot interrogate it for information that it does not itself possess. When you click on a window title or a scroll bar or any other UI item (User Interface), the mouse is not aware of what is being pointed at. What it “knows” is a pair of screen coordinates, which button (left/right/middle) was used, whether that button was pressed down or released up, and precisely when this occurred. That is all. So, it doesn't even know what application that UI item belongs to. The operating system knows that. Whether that UI item was a push button, a checkbox, a drop-down list, a menu option, or a piece of text, is only known for the application which has drawn that particular UI item. There are ways to get some of those details from the operating system/application. For example, developer tools like Window Spy are able to tell the object-ID and text-label of the buttons the mouse hovers over, but it does not work all the time. In our experience with Microsoft Office applications, this just does not work at all. Our custom-made software tool is not (yet) able to tell what our users click on.

Since mouse activities are essentially important details for our research, we came up with the alternative of capturing the portion of the screen around the mouse pointer as a still image, upon every mouse click event. We intended to have these pictures analyzed in an automated way, with a backup plan to manually review these pictures at analysis time should the first option not be possible. Either way, identifying what exactly the user was clicking on would allow us to learn what they were doing at that particular step of completing a given task; however, this was not the case.

Most of the time, even in the possession of the screenshots — to keep the required storage space and capture time to a minimum, we took only a portion of the screen, 100 pixels around the pointer — we were still unable to tell what that click was for, and could not

identify what the user did or why. So, we generally ended up with more questions than what we started our analysis with. Which was very, very frustrating.

As an alternative to the alternative, we finally worked around this problem by capturing a full-screen video recording of how our subjects work and what exactly they do. For the moment, we borrow that capability from a ready-made FFmpeg binary, which is automatically started when logging begins. This solution saved us development time — more precisely, it allowed us to delay it for later — and enabled us to continue testing users. Whenever we encountered confusing actions in a logfile, we watched the corresponding screen recording to see exactly what was happening. Most of the time we arrived at the same conclusion: that there was simply no logical explanation for the users' actions. Users are extremely unpredictable, their actions often defy sense and logic, they are very innovative and/or do unreasonable things.

1.5. Error recognition model

In this context, errors – due to their extremely high number in digital texts – must be categorized. Considering the terminology of both natural and artificial languages in connection with computers, errors were sorted into six major categories, three for both requirements of the properly formatted texts, forming the quantitative and the qualitative hypernym categories. Quantitative errors are those that are recognizable in any printed or displayed form. As such, they can potentially come into contact with the target audience of the document, the person or legal entity who is supposed to ultimately read it. The primary printing surface is paper, and the secondary is electronic display (monitor, tablet, television, etc.); however, in this sense other printable surfaces are taken into account (advertisement board, t-shirt, mug, etc.). On the other hand, qualitative errors are invisible in printed or displayed form. These can only be seen in the editable form of a document, which also means that there must be suitable software to open and edit that document type. As such, qualitative errors do not affect the target audience and can be recognized or managed only by authors, updaters, and editors/correctors. The hypernym categories are listed in [Table 1](#), accompanied by a short description of the hyponyms, considering the rules that should be followed to avoid these errors.

We must also note that there are errors that belong to more than one category. In such cases, the category is identified as primary where the applied rule is more definitive than the second or the third. For example, the incorrect or overuse of font styles – such as underline, bold, italic, etc. – belongs to the category of typographic errors, since typography set up the rules for the proper appearance of the printed documents (e.g., in [Figure 1](#) the whole text is formatted in italic and the section titles are underlined). However, in word processing, these errors are carried out by formatting, meaning that their secondary category is formatting. Quite often, the combination of typographic and formatting errors is accompanied by style errors ([Figure 1](#), Line 12). In these cases, the error can also be classified as style error, which indicates its third category.

The different development levels of digital texts can be measured based on the definition of properly formatted text, which has two criteria ([Csernoch, 2009](#)):

- The text fulfills the requirements of printed documents (quantitative requirements).

TABLE 1 The quantitative and qualitative errors and their hyponyms.

Quantitative error categories	
Syntactic	The grammar of the language
Semantic	The content of the text
Typographic	The appearance of the printed document
Qualitative error categories	
Layout	The breaking of the text into meaningful pieces
Formatting	The formatting commands of the software applied to the pieces of the text
Style	The style handling

- The content is invariant to modification (qualitative requirements): the document is editable, but the only allowed changes are those matching the original intention of the user.

1.6. The sample

For testing purposes, a short Microsoft Word document created by a 13-year-old Hungarian student was selected. The original version she created and submitted holds various errors concerning all the categories. However, the most serious are the layout errors (Csernoch, 2009, 2010, 2017) implemented by multiple Space characters, (lines 4–7, 12–13, 18–20, 25,30, 33, 34), multiple Enter characters (lines 1,

3, 8–9, 11, 14–15, 17, 21–29, 31–32, 34), end-of-line Enter characters (lines 4, 5, 6, 12, 18, 19), and manual hyphenation (line 19). Although we provide the entire original document in our tests (Figure 1), only the first five-line-long section (Figure 1, lines 4–7) – appearing as the second paragraph – is to be manipulated in order to avoid cumulative errors.

1.7. The layout errors of the selected section

The section selected for testing carries the following layout errors. Multiple Space characters are placed on the left side of the text to imitate a left indent (Figure 1, lines 4, 5, 6, 7; Figure 2, #1.). Every line ends with the end-of-paragraph character (Figure 1, lines 4, 5, 6; Figure 2, #2.), instead of allowing text to automatically flow and wrap, having only one end-of-paragraph mark at the real end of the paragraph. Furthermore, the starting and ending Space characters inside the parentheses are syntactic errors by definition according to the rules of the language, but generate a layout error (Figure 1, lines 5, 6; Figure 2, #3.). There is a capital letter at the beginning of the second line of the “paragraph,” which is a syntactic error (Figure 1, line 5; Figure 2, #4.), most likely originating from the combination of the end-of-paragraph mark at the end of line#4 (Figure 1) and the AutoCorrect setting of Microsoft Word (capitalize the first letter of sentences). The text is not quite coherent, as it was composed by a 13-year-old student, and contains

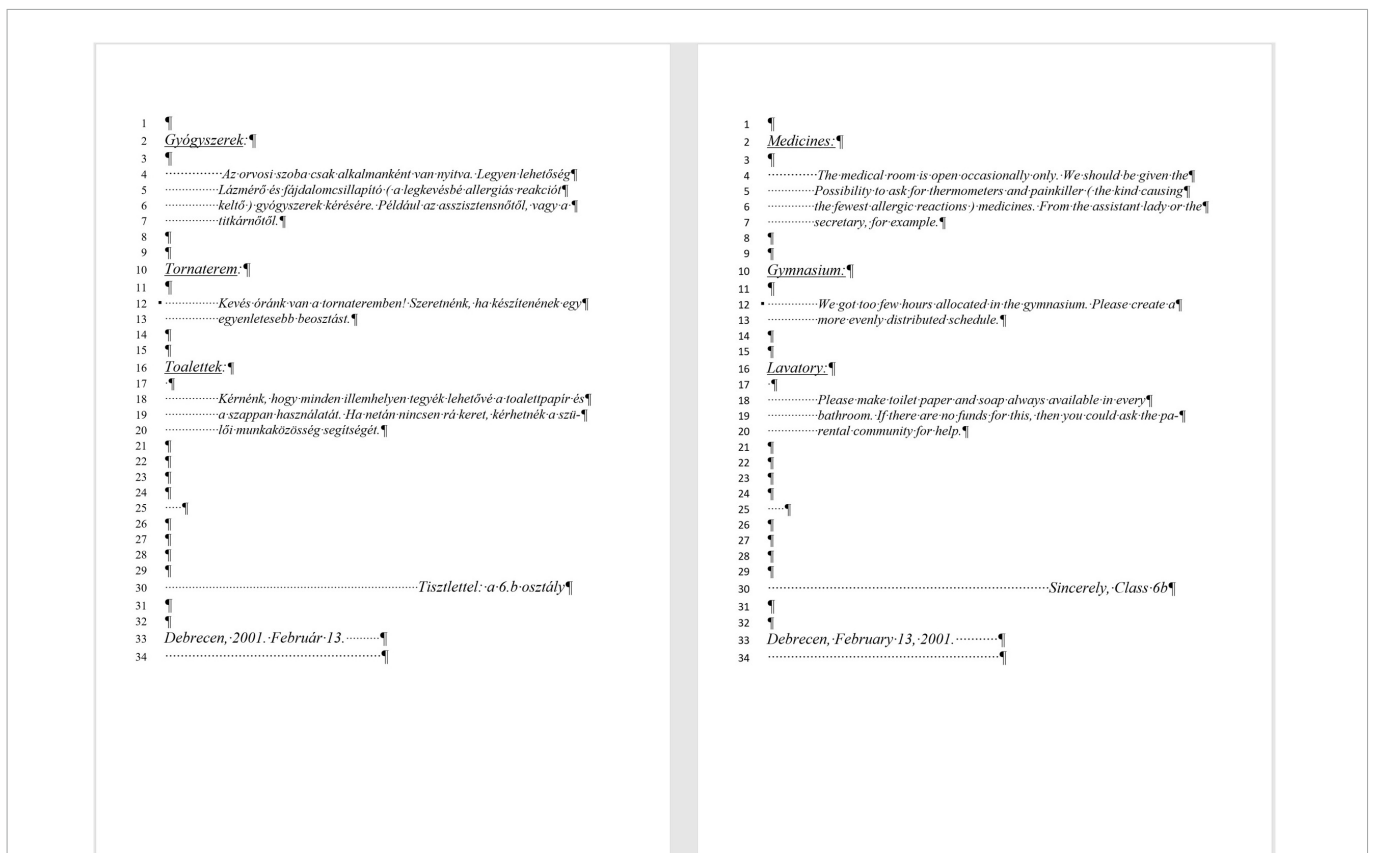
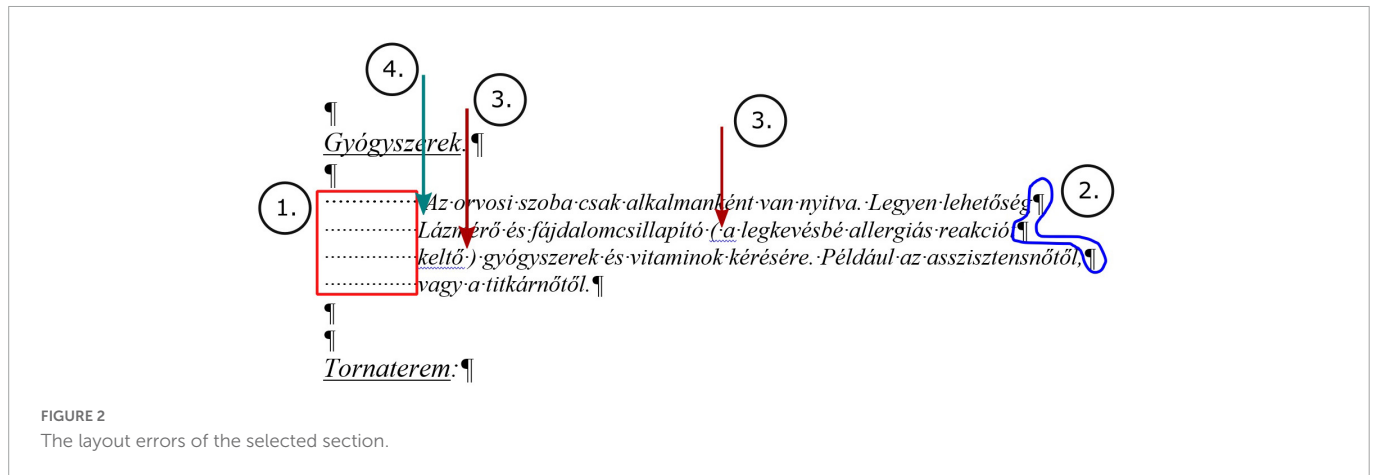


FIGURE 1 The original Hungarian document (left) and an English translation (right), with the non-printing characters shown. Line numbers are not visible in the original document, they are presented here to better describe the structure of the document and refer to its parts.



semantic errors but these issues are irrelevant from our current point of view.

1.8. The translated section

To aid easier understanding, we have included English translations in this publication. However, the original document was in Hungarian, and so far only that Hungarian-language version was tested. We must mention though, that layout errors #1 and #2 (Figure 2) are language independent and consequently, neither the language nor the content affects the efficacy of text handling and modification.

2. Materials and methods

2.1. Testing package

For our tests, a folder of seven files was prepared in a ZIP archive. The folder contains the following files (Figure 3):

- Four editable sample text documents – *gyogyszerek1.docx*, *gyogyszerek2.docx*, *gyogyszerek3.docx*, *gyogyszerek4.docx* (meaning *medicine1.docx*, *medicine2.docx*, etc.). Documents #1 and #2 are the original erroneous text, while in #3 and #4 all the layout errors have been corrected and properly formatted with the contents and the overall appearance of the page preserved (this is how the document should have been done in the first place).
- *WinANLITA.exe* – the custom-developed app that logs and starts the recording.
- A read-only document, *gyogyszerek_feladat.pdf* (means *medicines_tasks.pdf*), which briefly describes the document editing tasks our user was expected to complete on the four DOCX files.
- A copy of *ffmpeg.exe* – started by *WinANLITA* to make the screen recording. To avoid user confusion, we preferred to set this file to hidden (leaving only one visible EXE in the folder for the test subject to start).

The test consists of four tasks. Tasks #1 and #2 are to be performed in files #1 and #2 (erroneous), while tasks #3 and #4 are to be

performed in files #3 and #4 (correct). Tasks #1 and #3 are identical, and Tasks #2 and #4 are also identical. Essentially asking the user to perform the same tasks once in both the erroneous and the correct documents, respectively. In Tasks #1 and #3, a two-word-long expression was inserted into the text, as shown in the PDF file (Figures 4, 5), causing the last two lines to re-wrap. In Tasks #2 and #4, the font size of the whole section (paragraph) must be increased to 16 pts, causing all the lines to re-wrap.

Similar sample figures are included in the PDF document to explain the font-size-changing task.

2.2. Sample texts

The chosen section of files #1 and #2 are shown in Figure 6 while files #3 and #4 are in Figure 7 with the ruler and non-printing characters visible.

2.3. The procedure of modification

All four tasks are broken down into the following four steps.

1. Open the indicated DOCX file.
2. Perform the required changes – add the two-word-long expression or increase the font size based on the description and the figure shown.
3. If it is necessary, adjust the text appearance to make it look like the provided sample as much as possible.
4. Save and close the file.

2.4. The procedure of testing

In preparation, the user needed to download or copy the ZIP file from a provided location, unpack it, and add his/her name to the folder name. Following that, she/he opened the PDF file and read the tasks to become familiar with them. We also advised (but did not enforce) that they close any applications not connected to this test, since they might distract or confuse participants. If the user had any questions, they were invited to raise them and we answered at this point. After that, they were required to complete the task alone.







Name	Type	Size
 gyogyszerek_feladat.pdf	Adobe Acrobat dokumentum	211 KB
 gyogyszerek1.docx	Microsoft Word Document	13 KB
 gyogyszerek2.docx	Microsoft Word Document	13 KB
 gyogyszerek3.docx	Microsoft Word Document	13 KB
 gyogyszerek4.docx	Microsoft Word Document	13 KB
 WinANLITA.exe	Application	3,947 KB

FIGURE 3
Visible files of the ANLITA software package folder for testing.

Gyógyszerek:

*Az orvosi szoba csak alkalmanként van nyitva. Legyen lehetőség
Lázmérő és fájdalomcsillapító (a legkevésbé allergiás reakciót
keltő) gyógyszerek és vitaminok kérésére. Például az asszisztensnőtől,
vagy a titkárnőtől.*

FIGURE 4
Position of the new two-word-long expression inserted into the erroneous text.

Gyógyszerek

Az orvosi szoba csak alkalmanként van nyitva. Legyen lehetőség
lázmérő és fájdalomcsillapító (a legkevésbé allergiás reakciót keltő)
gyógyszerek és vitaminok kérésére. Például az asszisztensnőtől vagy a
titkárnőtől.

FIGURE 5
Position of the new two-word-long expression inserted into the correct text.

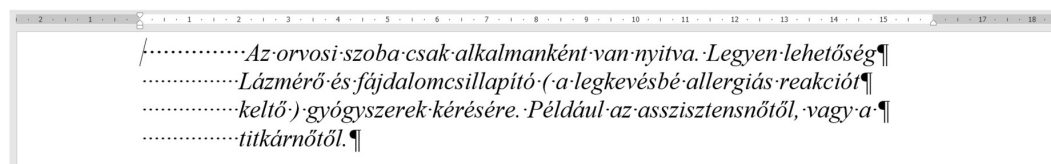


FIGURE 6
The ruler and the non-printing characters reveal that there is no left indent set up, and the section consists of four paragraphs.

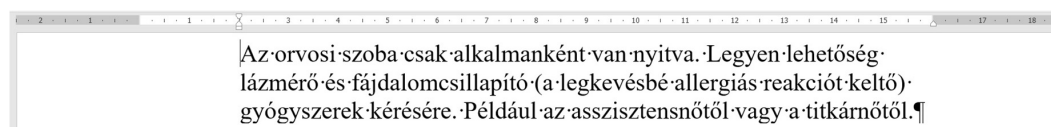
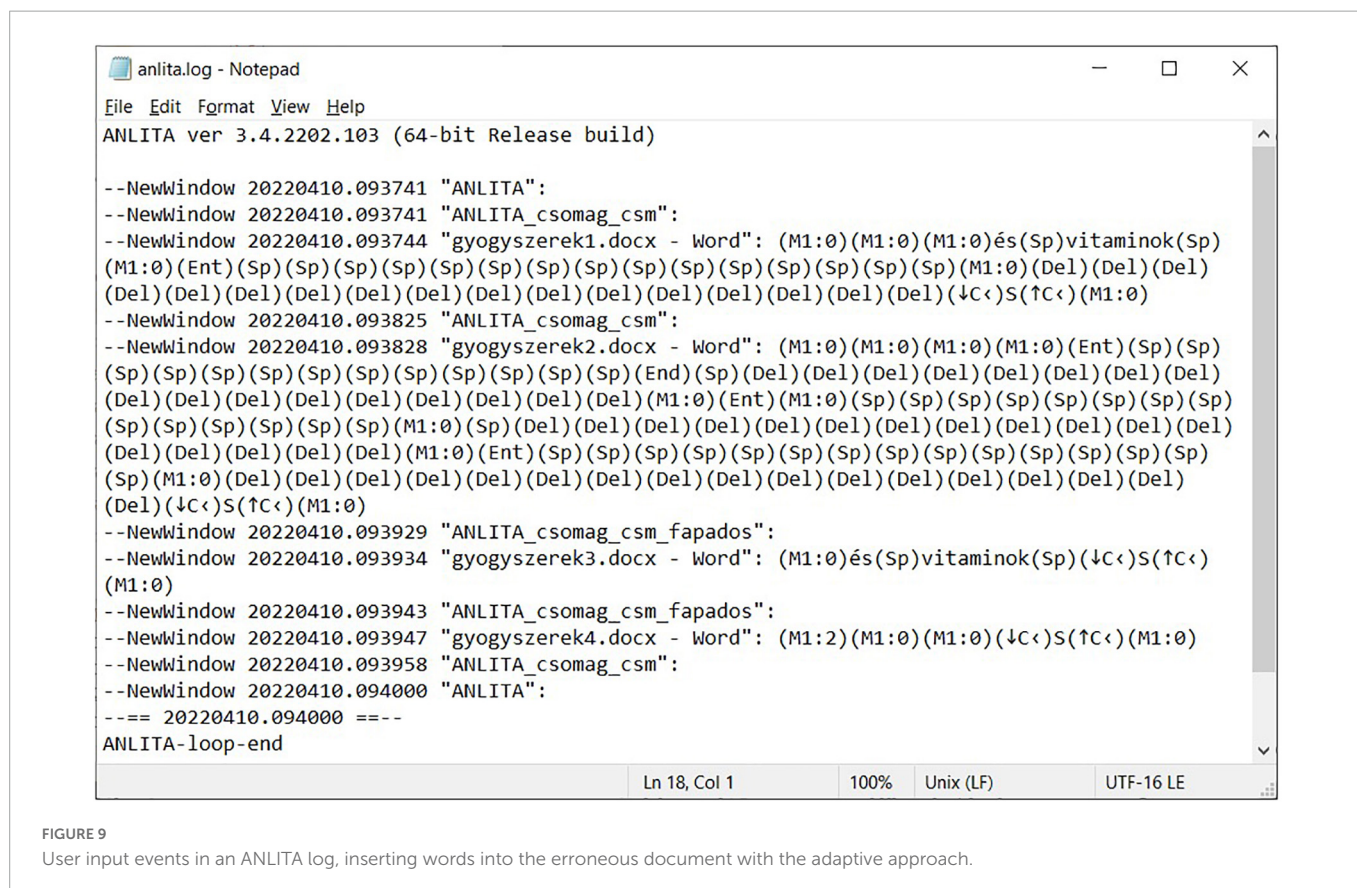
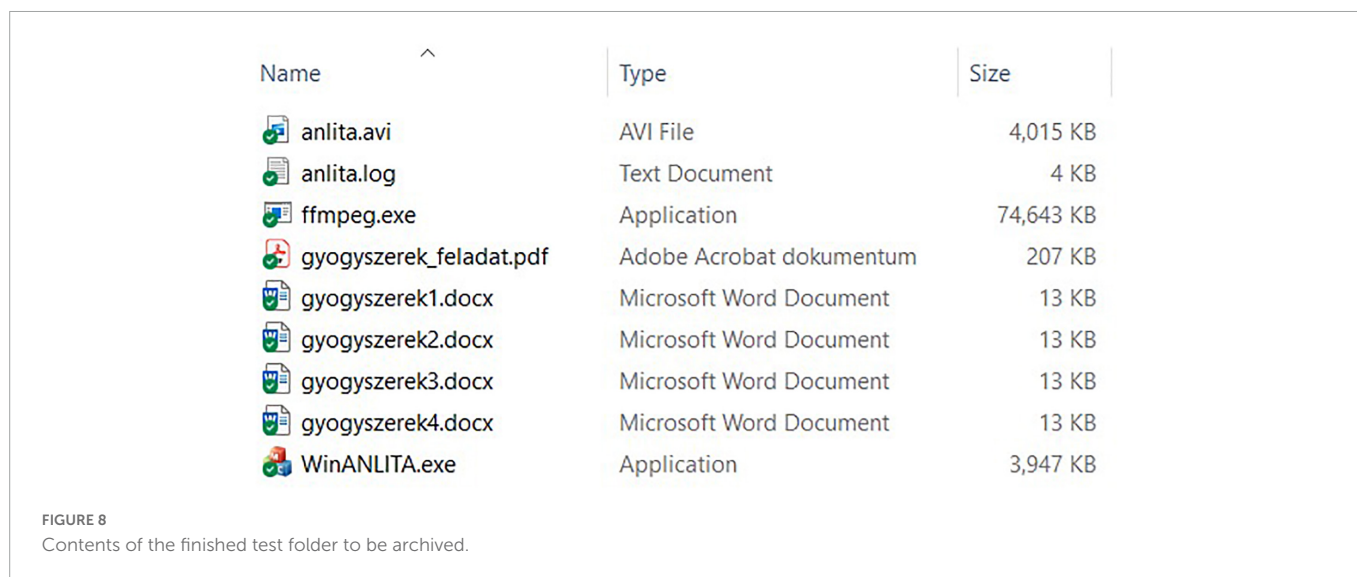


FIGURE 7
The ruler shows that there is a left indent configured, a single paragraph, and no extra space characters.

With the preparations complete, the user starts ANLITA, and from this point on, keyboard and mouse activities are logged until ANLITA is stopped. For the sake of fairness, ANLITA displays an

initial message, informing the user that their keyboard and mouse actions will be recorded, and the user can either opt out by closing the application or click the start button to continue. To avoid misuse



and bad intentions, ANLITA pauses input logging when the user switches to an application other than Microsoft Excel, Microsoft Word, LibreOffice Writer, or LibreOffice Calc, and continues to log input when the user switches back to any of these. This meant that ANLITA could not log passwords, bank card details, or any other sensitive data.

For the final steps, the user was required to archive the folder (now containing two additional files: the logged input and the recorded video) (Figure 8), and then upload or copy the resulting ZIP file to a designated location. Home users were allowed to send it in an e-mail.

3. Results

3.1. Logging application

The software tool we developed for the specific purpose of logging end-user input activities in text-based documents is entitled Atomic Natural Language Input Tracking Application (ANLITA). To date, it has complete monitoring capability of the keyboard, regardless of the complexity of key combinations pressed, held, or released, and the basic monitoring capability of the mouse. The output of the logging

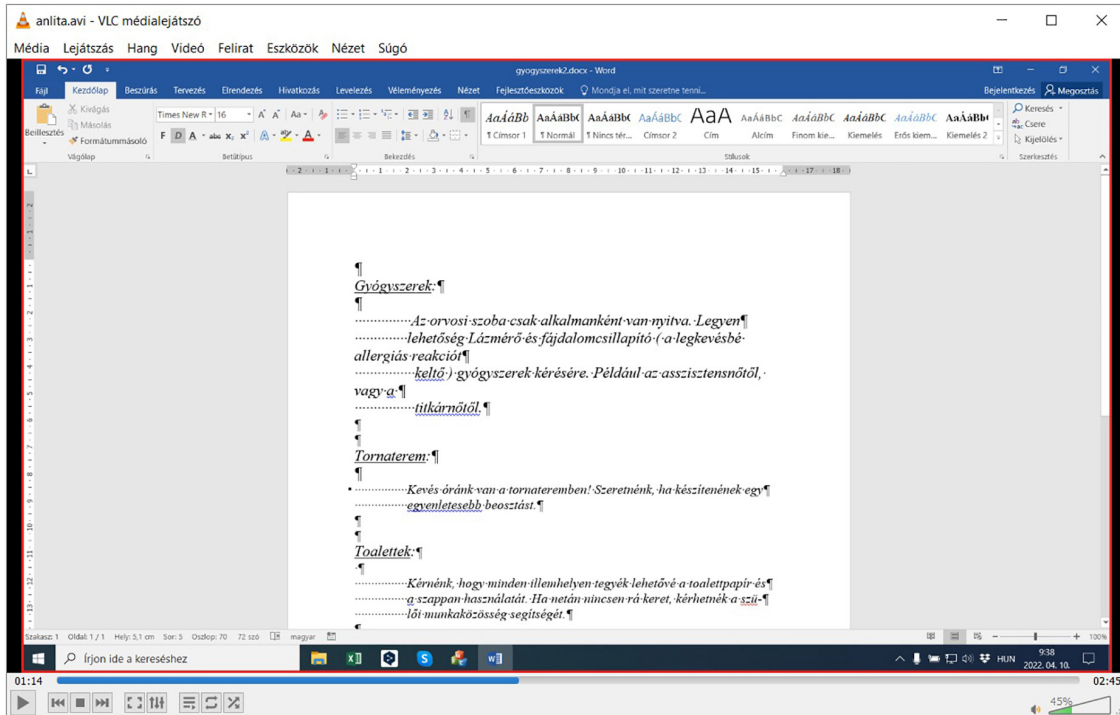


FIGURE 10 Video playback of what ANLITA (or rather FFmpeg) recorded when Figure 9 was logged.



FIGURE 11 User input events in an ANLITA log, inserting words into the erroneous document with the corrective approach.

process is in plain-text format. Screen recording is undertaken by an external FFmpeg binary, automatically started by ANLITA. The detailed text-log recorded by the software has already proven its usefulness in many tests. It also enabled us to record the difficulties faced by end-users in this study. These problems are primarily rooted in the lack of their computational thinking skills (Wing, 2006), and ANLITA had to be adjusted/updated several times to handle these issues properly.

3.2. The output of ANLITA

With the tasks complete, the user stops ANLITA. A log file (Figure 9) and a video file (Figure 10) are the output of the

testing. For reference purposes, the test was completed by one of our experienced researchers, working on the incorrect document in two different ways. The first approach adapted the incorrect pattern of the original layout errors (Figures 9, 10), and after the modification requested by the provided task, multiple Space and end-of-line Enter characters were inserted, deleted, or repositioned to imitate the desired appearance of the text with the hand-fabricated left indent (bricolage). The result is a document section as ill-structured as the original was (meant to be a single paragraph). This is the adaptive approach.

The second approach completes the same task with the intention of improvement. It begins by eliminating the layout errors (Figures 2, 11, 12). Hence the automatic re-flow of text content will no longer generate additional problems when further text editing is

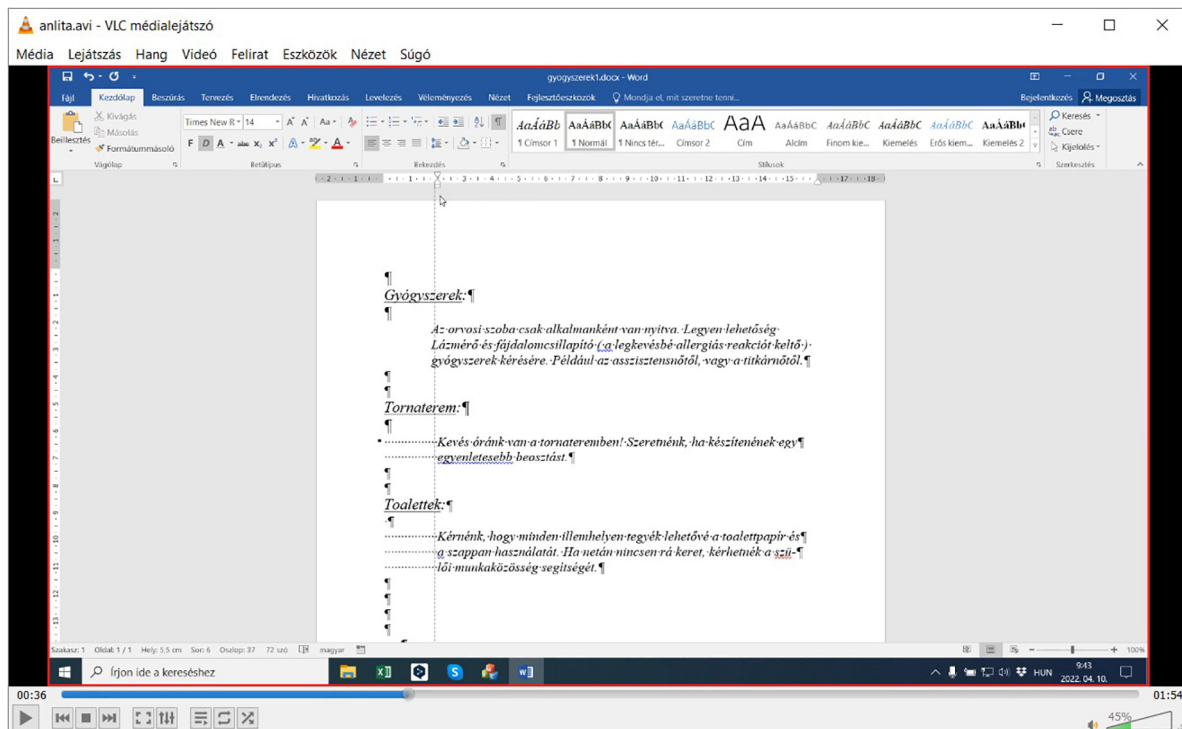


FIGURE 12 Video playback of what ANLITA (FFmpeg) recorded when Figure 11 was logged.

performed, resulting in a correct document section (a true, single paragraph). This is the corrective approach.

3.3. The evaluation of the results

According to the aims and the hypothesis of our research, pre-testing was carried out by an expert in text management. Her role was to complete the tasks in the least possible steps and in the shortest time. However, neither the number of steps nor the time were optimized, allowing space for minor detours, not altering the magnitude of the modification.

In *gyogyszerek1.docx*, dealing with the imitation of word processing, it took 41 s to add the extra two words and then manually adjust the hand-fabricated indentation (from 09:37:44 to 09:38:25) (Figure 9) using the adaptive approach. Eliminating the layout issues and then adding the extra two words, using the corrective approach took 38 s (from 09:43:05 to 09:43:43) (Figure 11). In *gyogyszerek2.docx*, increasing the font size using the adaptive approach took 61 s (from 09:38:28 to 09:39:29) (Figure 9), while doing the same with the corrective approach took 37 s (from 09:43:49 to 09:44:26) (Figure 11). Working with a properly structured and formatted document in *gyogyszerek3.docx*, inserting the two words took 9 s (from 09:39:34 to 09:39:43) (Figure 9), and increasing the font size in *gyogyszerek4.docx* took 11 s (from 09:39:47 to 09:39:58) (Figure 9).

Based on our reference solutions, carrying out these first level modifications (Hankiewicz and Butlewski, 2014) – typing two words and changing the font size in a short paragraph – takes about five times longer in the erroneous documents than in the properly formatted versions (section “1.5. Error recognition model”).

4. Discussion

With our logging tool, we found objective proof that qualitative errors are the most demanding, and texts carrying these errors require the largest number of collateral adjustments during a modification process, thereby wasting considerable amounts of human and computer resources.

Overconfident DKE end-users often claim that displaying non-printing characters in a word processor is only a distraction, hence these symbols must stay hidden. However, our testing with ANLITA revealed that those who intentionally keep these structural marks always turned off, perform a higher number of unnecessary editing steps than those who – periodically at least – have these symbols displayed. This finding leads to the conclusion that by neglecting non-printing characters, the editing of digital texts is carried out blindly, leaving more space for errors. Unfortunately, this phenomenon cannot be considered as the shortcoming of end-users only, but also as a shortcoming of most applications and software development. For example, some online word processors do not offer the Show/Hide formatting symbols command, consequently, the requirements of the properly edited text are extremely difficult to fulfill, even for those who are aware of them. Without visible structural marks, extra Space and Enter characters can easily lead to the loss of time and/or data in a presentation, spreadsheet, and text editor applications.

A further finding of our tests is that the majority of end-users generally do not know how to download a file to a designated folder, name a folder consciously (avoiding national characters, space, troublesome symbols), unzip an archive (difference between opening and extracting a ZIP file), compress a file or a folder, switch or navigate between applications efficiently, upload or otherwise share a large file.

The pre-testing period of ANLITA made it clear that these tests should be instructor-led and strictly supervised by professionals who can verify all the end-users and their activities until the recording starts, and then do the same at the end, by stopping ANLITA and uploading the results. Providing only minimal guidance leads to data losses and wasted time, due to the lack of computational thinking skills (Wing, 2006) and/or the misconceptions carried by overconfident DKE (Kruger and Dunning, 1999) end-users and/or the victims of SCF (Kahneman, 2011).

5. Conclusion

End-user activities connected to erroneous office documents, primarily those that involve word processed texts, spreadsheets, and presentations, have been studied for decades without a real breakthrough. These matters and problems are invisible to IT professionals, corporate managers, and researchers in both the fields of information systems and computer education. Its effects in spreadsheets are more recognizable, compared to word processed texts and presentations, because problems in a spreadsheet can directly cause mentionable financial losses. In contrast, since they only have an indirect and hard-to-recognize financial impact, errors in text-based documents are ignored. To improve the quality of digital texts and the efficiency of digital text handling, proof must reach the wider community of end-users and education.

To carry out objective measurements in testing the quality of digital text-based documents, our research group developed a user input logging application entitled ANLITA and collected several hundred erroneous documents to be used as samples for future research. ANLITA logs all the keyboard and mouse events and uses an external tool to capture a screen recording of the test as it is being performed. The output of ANLITA is a human and machine-readable plain text file and a Supplementary Video.

Building upon ANLITA, a reference test was performed by a professional in our research group using two different solution approaches. Her reference solutions keep both the keyboard and the mouse activities to the necessary minimum. We found proof that carrying out simple first level modifications in a one-paragraph long text burdened with layout errors takes around five times longer than in a properly formatted text. In an economic sense, this result can be converted into companies needing to hire five times more employees, purchase and operate five times more computers, renting five times more office space, etc., to undertake jobs that could have been equally well completed with 20% of those resources, at one-fifth of the costs.

Estimating the magnitude of loss in a longer document with more planned modification requires further testing and research. However,

without the correction of the document, a linear regression is likely, predicting rapidly upscaling losses.

Data availability statement

The original contributions presented in this study are included in the article/supplementary material, further inquiries can be directed to the corresponding author.

Ethics statement

Ethical review and approval was not required for the study on human participants in accordance with the local legislation and institutional requirements. Written informed consent from the participants or participants' legal guardian/next of kin was not required to participate in this study in accordance with the national legislation and the institutional requirements. The piece of work used was originally produced by a Grade 7 student previously taught by one of the manuscript authors.

Author contributions

Both authors listed have made a substantial, direct, and intellectual contribution to the work, and approved it for publication.

Conflict of interest

The authors declare that the research was conducted in the absence of any commercial or financial relationships that could be construed as a potential conflict of interest.

Publisher's note

All claims expressed in this article are solely those of the authors and do not necessarily represent those of their affiliated organizations, or those of the publisher, the editors and the reviewers. Any product that may be evaluated in this article, or claim that may be made by its manufacturer, is not guaranteed or endorsed by the publisher.

References

- Ben-Ari, M. (2015). *Bricolage forever! PPIG 1999. 11th annual workshop. 5–7 January 1999. Computer-based learning unit*. Leeds: University of Leeds.
- Ben-Ari, M., and Yeshno, T. (2006). Conceptual models of software artifacts. *Interact. Comput.* 18, 1336–1350. doi: 10.1016/j.intcom.2006.03.005
- Correy, H. (2011). *Exercise: Word processing for data entry*. Available online at: <https://studyres.com/doc/5756468/exercise--word-processing-for-data-entry> (accessed June 17, 2022).
- Csernoch, M. (2009). Teaching word processing – the theory behind. *Teach. Math. Comp. Sci.* 7, 119–137.
- Csernoch, M. (2010). Teaching word processing – the practice. *Teach. Math. Comp. Sci.* 8, 247–262.
- Csernoch, M. (2017). Thinking fast and slow in computer problem solving. *J. Softw. Eng. Appl.* 10, 11–40.
- Csernoch, M., and Dani, E. (2022). Do you speak, write and think in informatics? *Acta Polytech. Hung.* 19, 113–131. doi: 10.12700/APH.19.1.2022.19.8

- Downing, D., Covington, M., and Covington, M. (2017). *Dictionary of computer and internet terms*. Hauppauge, NY: Barrons Educational Series.
- EA SEND Implementation Team (2021). *Word processing policy for examinations*. Available online at: https://www.ela.kent.sch.uk/files/ELA_Word_processing_policy_2021_22.docx (accessed June 17, 2022).
- EuSpRIG Horror Stories (2022). Available online at: <https://eusprig.org/research-info/horror-stories/> (accessed May 21, 2022).
- Gareth (2017). *Qualification component and record of learner achievement. Word processing software skills E3 (L/616/1318)*. Available online at: https://www.aim-group.org.uk/clientfiles/files/units/learner_achievements/Word%20Processing%20Software%20Skills%20E3%20CV2%20AIM%20Awards%20Component%20V1-1.docx (accessed June 17, 2022).
- GCFGlobal (2022). *Word basics*. Available online at: <https://edu.gcfglobal.org/en/word/> (accessed June 17, 2022).
- Goodwin, E. (2022). *Business technology applications. Word processing basics*. Available online at: <https://slideplayer.com/slide/6126983/> (accessed June 17, 2022).
- Hankiewicz, K., and Butlewski, M. (2014). "Efficiency in performing basic tasks using word processing programs by the elderly as a measure of the ergonomic quality of software," in *Human-computer interaction, Part I, HCII 2014, LNCS 8510*, ed. M. Kurosu (Cham: Springer International Publishing Switzerland), 481–488.
- Kahneman, D. (2011). *Thinking, fast and slow*. New York, NY: Farrar, Straus; Giroux.
- Kruger, J., and Dunning, D. (1999). Unskilled and unaware of it: how difficulties in recognizing one's own incompetence lead to inflated self-assessments. *J. Pers. Soc. Psychol.* 77, 1121–1134. doi: 10.1037//0022-3514.77.6.1121
- Malmi, L., Sheard, J., Kinnunen, P., Simon, and Sinclair, J. (2019). "Computing education theories: what are they and how are they used?," in *ICER '19: Proceedings of the 2019 ACM conference on international computing education research*, Toronto ON, 187–197.
- Malmi, L., Sheard, J., Kinnunen, P., Simon, and Sinclair, J. (2022). Development and use of domain-specific learning theories, models and instruments in computing education. *ACM Trans. Comput. Educ.* 23:48. doi: 10.1145/3530221
- McKinney, J. A. (2022). *Formatting and editing vocabulary*. Available online at: https://igcseicthelp.weebly.com/uploads/1/8/4/5/18452919/ict_igcse_paper_2_revision_wordprocessing.docx (accessed June 17, 2022).
- Microsoft Language Portal (2021). Available online at: <https://www.microsoft.com/en-us/language> (accessed April 25, 2021).
- National Center for Computing Education (2019). *Plenary: quiz*. Available online at: <https://hollis.horizonstrust.org.uk/file/hollis/ap-solutions-plenary-quiz-questions-answers-29306.docx> (accessed June 17, 2022).
- Panko, R. R. (1998). What we know about spreadsheet errors. *J. Organ. End User Comput.* 10, 15–21. doi: 10.4018/joeuc.1998040102
- Panko, R. R. (2013). "The cognitive science of spreadsheet errors: why thinking is bad," in *Proceedings of the 46th Hawaii international conference on system sciences, January 7-10, 2013, Maui*.
- Rigdon, J. C. (2016). *Dictionary of computer and internet terms "entries from the microsoft language portal. © 2016 Microsoft corporation. All rights reserved*, Vol. 1. Cartersville, GA: Eastern Digital Resources.
- Sebestyén, K., Csapó, G., Csernoch, M., and Aradi, B. (2022). Error recognition model: high-mathability end-user text management. 19, 151–170. doi: 10.12700/APH.19.1.2022.19.10
- tabaks (2022). *ICT IGCSE practical paper 2 revision for word processing*. Available online at: https://igcseicthelp.weebly.com/uploads/1/8/4/5/18452919/ict_igcse_paper_2_revision_wordprocessing.docx (accessed June 17, 2022).
- TechTerms (2021). *End user*. Available online at: <https://techterms.com/definition/enduser> (accessed April 25, 2021).
- The Skills Factory (2022). *Microsoft word - tutorial for beginners in 13 MINUTES!*. Available online at: <https://www.youtube.com/watch?v=GBHUBEOTdCA> (accessed June 17, 2022).
- Wing, J. (2006). Computational thinking. *Commun. ACM* 49, 33–35. doi: 10.1145/1118178.1118215
- Wood, M. (2007). *Word processing*. Available online at: <https://www.uen.org/lessonplan/download/28213?lessonId=454&segmentTypeId=3> (accessed June 17, 2022).