ISSN:2184-044X ISBN:978-989-54312-6-7 © 2019
DOI: 10.36315/2019v2end025

LOW-COST CODE TO CHECK SOME OF THE 20 RULES FOR EFFICIENT WEB WRITING: A PROBLEM-BASED LEARNING SITUATION

Lina García-Cabrera
Computer Science Department, University of Jaén (Spain)

Abstract

People do not read information on the Web in the same way as they read printed material. Studies have shown that reading on screen is around 25% slower than reading from paper. Instead of carefully reading information, users typically scan it (Morkes & Nielsen, 1997). Users will have time to read 28% of the words if they devote all of their time to reading. More realistically, users will read about 20% of the text on the average page (Nielsen, 2008).
Therefore writing for the web is different from traditional print writing; it has its own rules. Nowadays everyone may be a web writer so everyone should learn the basic rules about how to write effective web content. Users should not only know how to write for the web but also how to present information to facilitate scanning and enhance comprehension.
In short, the text for the web must be easy to understand, concise, scannable and leave no uncertainty in the mind of the reader at first and faster reading.
In order to reach these goals, a previous contribution (García-Cabrera, 2018) offers a brief methodology based on two basic principles (Make text short and Make text scannable) and 20 good practices structured in a series of microcontents (Loranger & Nielsen, 2017). Following this brief guide you can teach/learn how to apply the principles of ease of use or usability of the Web to write digital text. The results show that this minimalist methodology, structured in microcontents, can be taught and learned in just a few minutes.
With the aim of supporting this brief method automatically, this paper shows how to code some of these 20 good practices in an easy and low-cost way. This challenge will be a flipped classroom in whatever subject where student learn programming. In particular, the students of operating systems can learn about bash programming but at the same time about web information system. In addition, the experience of solving this open-ended problem teaches general computer science engineering competencies such as "Solve problems through initiative, determination, independence and creativity" and traversal skills such as "Manage time and available resources. Work in an organised manner."
The final objective of this paper is to share this problem-situation case to encourage the design and implementation of problem-based learning that allows students to apply content from more than one Computer Science Engineering subject and generate useful, open, low-cost software products.

Keywords: *Flipped classroom, problem-based learning, microlearning, web writing.*

1. Introduction

This contribution is based on a reality that generates a need, which involves a complex problem that requires a solution.

The reality is that content is the key to the success of a good website. But the digital medium has some limitations that detract from the ability of readers to understand the Web:

- Reading speed decreases by 25% (Morkes & Nielsen, 1997) and content is poorly understood.
- Users are unable to focus their attention on long pieces of text. If the text is too long, it is simply not read, it is preferred to print it if we are really interested in that information.
- The sight jumps from one side to the other without focusing on the details. Studies have shown that nearly 80 percentage of users scan Web pages for visual clues so they can find content quickly. They look for things that stand out: bold text, bullets, links and descriptive titles.
- Attention is dispersed. Users read only 28% of the information if they are reading. But in practice they only read about 20% of the text on each page (Nielsen, 2008).

Therefore, it is imperative to adapt our way of writing to the digital medium, to the Web. It is necessary to write texts and present this writing on Web pages in an effective way. In other words, texts that help users to read less yet remember and understand more information with less effort. So, the complex problem to be solved is to coach current and future Web writers (who will be all of us) on how to write efficiently on the Web. In order to solve this, a previous contribution (García-Cabrera, 2018) offers a minimalist methodology based on two basic principles (Make text short and Make text scannable) and 20 good practices structured in a series of microcontents (Loranger & Nielsen, 2017). It therefore applies a microlearning pedagogical approach (Job & Ogalo, 2012). Following this brief guide you can teach/learn how to apply the principles of ease of use or usability of the Web to write digital text.

But this is not enough; it would be better to have some automatic assistance to help web writers. As a result, the challenge is to solve an open-ended problem: to provide a useful, low-cost and open software product that supports as far as possible the edition of web texts.

The purpose of this work is to convert this realistic problem that requires a solution in the problem-situation design based on the problem-based learning (PBL) approach (Barrows, 1996) as part of flipped class activities. The main objectives are to make bash programming more attractive to students as well as helping them to develop professional skills, that is, general computer science engineering competencies, for instance, first as "Solve problems through initiative, determination, independence and creativity" and traversal skills as "Manage time and available resources. Work in an organised manner."

2. Previous work

The first step in solving a problem is to describe it as clearly, specifically, simply and systematically as possible. This effort was made in a previous contribution by means of a brief methodology, based on two basic principles and only 20 good practices structured in a series of microcontents. The Methodology for Writing Effective Texts for the Web applies two important pieces of advises: a) **Make text short** and, b) **Make text scannable**. I shall try to explain what I mean.

First of all, your text should be brief to the point it should take minimal effort to understand. Text must be concise, clear and direct. It must be as long as necessary but as short as possible. (a1) Connect with the audience and keep their attention. (a2) Inverted pyramid. Story is structured from most to least important. (a3) Answering the basic 5 W's and H Questions: WHAT, WHO, WHEN, WHERE, WHY and HOW. (a4) Facts and not empty words. Avoid subjective descriptions and remove instructions text. In contrast, use factual information. (a5) Use simple sentence structures (15-20pp). (a6) Use the most common words and those with the fewest letters. (a7) Avoid jargon, acronyms or obscure words. (a8) Use plain, active and clear language (English, Spanish and so on) (Boldyreff, 2001). (a9) Remove instruction text or reduce to a minimum. (a10) Use time expressions with day and time, avoid relative time expressions referring to the past or future because the information remains on the web.

Second, the text should be formatted to help users find content. Users go online with a specific aim or interest. They are going to browse the page to pick up the information they actually need. There are many ways to format your text to aid scannability: (b1) Break your text in short paragraphs with just a few lines, no more than 3 or 5 lines (40-70 words). (b2) Start each paragraph with the most relevant and usable information. (b3) Write each paragraph (or phrase) around one idea. (b4) Use headlines in order to break up the content and make it very easy to scan (no more than 3 levels). (b5) Highlight the most important sentences in order to draw attention to ideas that are crucial. You can make them bold but never italic or underlined. Italic it is difficult to read. Underlining can be confused with a link and it is less illegible. (b6) Use a numbered list for things in order, bulleted list for things in no particular order. (b7) No more than 7 items in a list, much better if there are 5 things. Lists create chunks of content that facilitate scanning, separating ideas and allow for counting. (b8) Use numerals rather written numbers (no more than 4 figures). (b9) With large numbers with several zeros, use first the number and after the word 'thousands', 'millions', 'thousands of millions' or 'billions'. (b10) Do NOT use Roman numerals, only for kings, popes and centuries.

Following this brief guide you can teach/learn how to apply the principles of usability of the Web to write digital text (Krug, 2013). The aim is to apply simple but efficient recommendations aimed at producing concise texts presented in such a way that they are easier to understand in a first and quick reading by the user. It seeks to develop in one of the basic skills, essential in the Knowledge Society, to be a Web writer who contributes to the development of collective intelligence (Woolley, 2010).

3. Considering the feasibility of the PBL

Before thinking about the design & implementation all aspects related this problem situation (Edström & Kolmos, 2014) it is necessary to check if it conforms to the characteristics of a PBL and at the same time if this learning process can achieve some learning outcomes relevant to the subjects involved and general competences related to computer science degree.

The first step is to analyse the problem in detail in order to have a clear definition and conception of the problem taking into account the restrictions: the solution must adopt an open-source model, encourages open sharing and collaboration, simple and by means of bash language. Therefore, the question is which of these 20 good practices (see section 2) can be automated in these low-cost conditions.

To face this challenge, the student must considerer the strengths of bash programming language. Bash is an unix shell (a command interpreter) and a shell program (sometimes called a shell script) is a text file that contains standard UNIX and shell commands. It is far easier to write and debug a shell script than a C/C++ program. A shell script, by virtue of being a text file, can easily be viewed to check out what actions it is performing. A shell script can be transferred to other Unix and Unix-like operating systems and executed (if the shell itself is present). In bash you can use and combine easily all unix/linux commands and unix/linux provides a great many powerful commands to process texts and strings in different ways. These text processing commands are often implemented as filters as: *grep*, *sort*, *uniq*, *cat*, *more*, *cut*, *paste*, *head*, *tail*, *wc*, *tr*, and so on. In addition, bash programming supports list data structure called *array*. An *array* stores a list or collection of elements not bound to specific data type (strings, numbers and so on). So, in bash language it is very easy to make iterate loops over a list of words.

Taking into account these previous facts, the easiest rules to face are (**a5**, **b1**) because they imply a code that extracts phrases or paragraphs and count the number of words.

It must be assumed that raw text will be processed and therefore all these good practices (b4, b5, b6, b7) related to style and editing, in principle, require an off bash solution and a brainstorming session. Perhaps, for rule (**b7**) a rough solution is possible. For instance, by assuming that if paragraphs with very few words in a row appear in the text, they may be elements of a list. The solution would be to extend the previous code (b1) and check that the number of very short paragraphs in a row should be less than 7.

Initially you may think that the rest of the rules would be out because they require semantic interpretation and need a complex natural language processing. But a deeper analysis of these rules allows us to identify a set of rules (**a7**, **a10**, **b8**, **b9**, **b10**) for which a rough solution could be found because some pattern could be found.

For instance, for the good practices (a7, a10, b8) it is possible to have 3 glossaries files. One text file that stores all jargon, acronyms or obscure words (the most common), another that contains the most frequent relative temporal expressions and another with all written numbers from zero to nine hundred and ninety-nine. The code for the three is to check that these words do not appear in the text.

It is possible to code rule (b9) finding big numbers, that is, sequences of 5 or more figures. Finally, the (a10) rule involves checking for Roman numerals, that is, finding words formed only by the letters I, V, X, L, C, D, M, in upper or lower case. The program can warn the writer to make sure he is using it with kings, popes, and centuries. You can also try to check if the word “pope”, “king/queen” or “century” appears nearby.

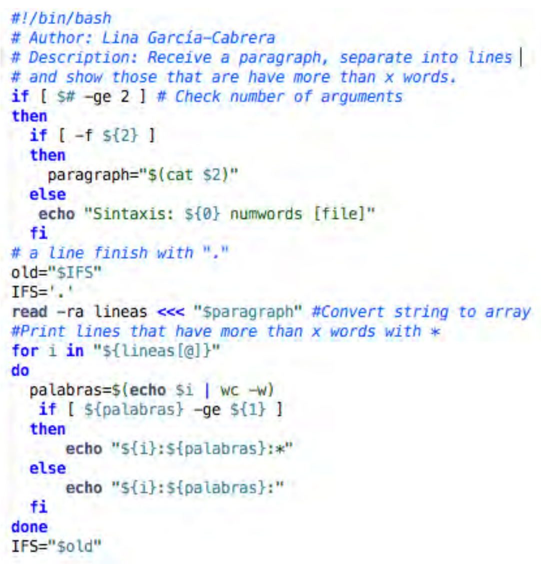
From the previous discussion, it follows that the problem is truly ill-structured in which there is no obvious right answer. In order to develop a robust solution, the students need: a) define the problem, b) generate possible solutions, c) evaluate the alternative solutions by constructing arguments and articulating personal beliefs, d) implement the most viable solution, and e) monitor the implementation (Jonassen & Hung, 2018) (Sinnott, 1989). Technical knowledge and reasoning must be applied as basic programming skills and the use of the engineering lifecycle process Conceive Design Implement Operate, CDIO (Crawley, 2007).

4. Considering the knowledge and skills learned

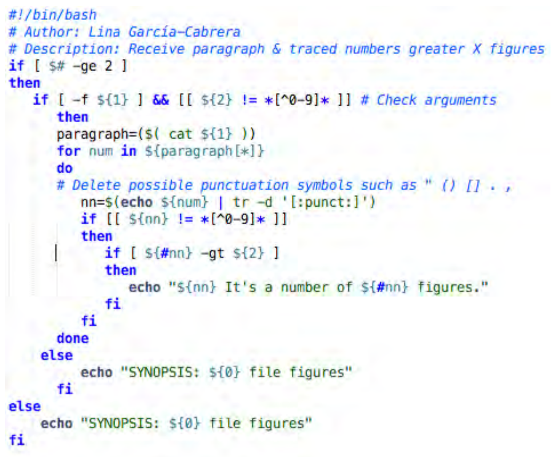
The second step is to check if this PBL produces deep knowledge and skills that encourage independent and lifelong learners and assist in development of skills relating to bash programming. From the previous sections, it is evident that students learn some technical knowledge/methods about web usability and writing web. Also the students develop personal and professional skills such as critical thinking, that is, general computer science engineering competencies as "Solve problems through initiative, determination, independence and creativity" and traversal skills such as "Manage time and available resources. Work in an organised manner."

The only way to check if the solutions generated develop the most relevant features of bash programming is to codify them. In addition, the implementation of this programming allows us to identify the information and resources that students need to understand and solve the problem. As can be seen in the examples (Figure 1), it is necessary to learn skills relating bash programming such as: arguments, variables, special variables, quotes, command substitution, input, redirection, pipes, while and for loops, conditionals, boolean operations, files and so on. Therefore, this problem situation gives students a solid foundation in how to write Bash scripts, that is, students will be well armed with the right knowledge and skills.

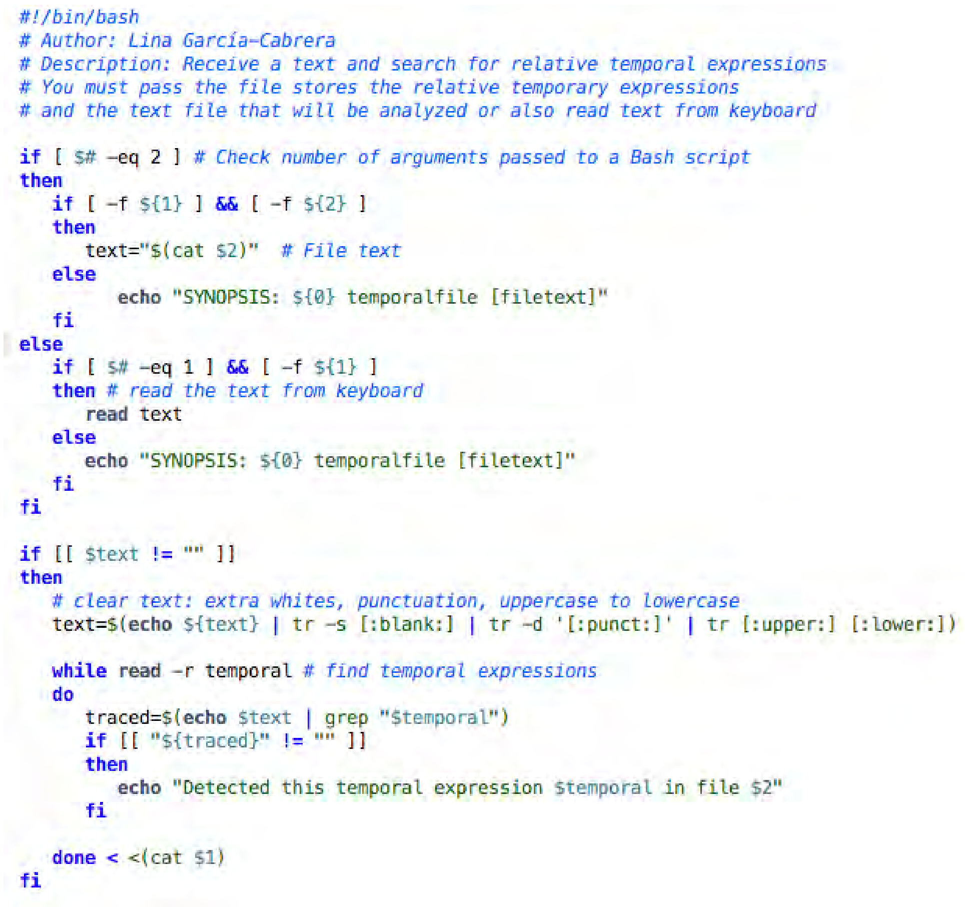
Figure 1. Bash code of rules a5, a10 and b9.



```bash
#!/bin/bash
# Author: Lina García-Cabrera
# Description: Receive a paragraph, separate into lines
# and show those that are have more than x words.
if [ $# -ge 2 ] # Check number of arguments
then
  if [ -f ${2} ]
  then
    paragraph="$(cat $2)"
  else
   echo "Sintaxis: ${0} numwords [file]"
  fi
# a line finish with "."
old="$IFS"
IFS='.'
read -ra lineas <<< "$paragraph" #Convert string to array
#Print lines that have more than x words with *
for i in "${lineas[@]}"
do
  palabras=$(echo $i | wc -w)
   if [ ${palabras} -ge ${1} ]
  then
      echo "${i}:${palabras}:*"
  else
      echo "${i}:${palabras}:"
  fi
done
IFS="$old"
```



```bash
#!/bin/bash
# Author: Lina García-Cabrera
# Description: Receive paragraph & traced numbers greater X figures
if [ $# -ge 2 ]
then
   if [ -f ${1} ] && [[ ${2} != *[^0-9]* ]] # Check arguments
      then
      paragraph=($( cat ${1} ))
      for num in ${paragraph[*]}
      do
      # Delete possible punctuation symbols such as " () [] . ,
         nn=$(echo ${num} | tr -d '[:punct:]')
         if [[ ${nn} != *[^0-9]* ]]
         then
            if [ ${#nn} -gt ${2} ]
            then
               echo "${nn} It's a number of ${#nn} figures."
            fi
         fi
      done
    else
         echo "SYNOPSIS: ${0} file figures"
      fi
else
    echo "SYNOPSIS: ${0} file figures"
fi
```



```bash
#!/bin/bash
# Author: Lina García-Cabrera
# Description: Receive a text and search for relative temporal expressions
# You must pass the file stores the relative temporary expressions
# and the text file that will be analyzed or also read text from keyboard

if [ $# -eq 2 ] # Check number of arguments passed to a Bash script
then
   if [ -f ${1} ] && [ -f ${2} ]
   then
      text="$(cat $2)"  # File text
   else
         echo "SYNOPSIS: ${0} temporalfile [filetext]"
   fi
else
   if [ $# -eq 1 ] && [ -f ${1} ]
   then # read the text from keyboard
      read text
   else
      echo "SYNOPSIS: ${0} temporalfile [filetext]"
   fi
fi

if [[ $text != "" ]]
then
   # clear text: extra whites, punctuation, uppercase to lowercase
   text=$(echo ${text} | tr -s [:blank:] | tr -d '[:punct:]' | tr [:upper:] [:lower:])

   while read -r temporal # find temporal expressions
   do
      traced=$(echo $text | grep "$temporal")
      if [[ "${traced}" != "" ]]
      then
         echo "Detected this temporal expression $temporal in file $2"
      fi

   done < <(cat $1)
fi
```

5. Discussions and conclusions

Web content is crucial to the success of a good website. People go to the Web to quickly find very specific information in which they are really interested. But the reader's attention tends to be dispersed and the efficiency of reading and understanding is very poor. This work presents a minimalist methodology based on two basic principles (Make text short & Make text scannable) and 20 good practices structured in microcontents. Bearing in mind that the average speed of comprehensive reading is between 200 and 300 words per minute, the reading of this minimalist methodology with a length of about 400 words involves around 2 minutes. Following this brief guide you can teach/learn how to write efficiently on the Web in just a few minutes. Hence, it adopts a microlearning as a strategic process for creating, harvesting, acquiring, retaining and applying knowledge learning (Job & Ogalo, 2012). Also, the paper shows how to convert this realistic world problem into a PBL that provide a useful, low-cost and open software product that supports as far as possible the edition of web texts. The analysis of the feasibility of the problem-situation shows that 8/20 rules can be implemented, 40%. This figure could be higher if the text was in HTML5 format. In this case the rules related to style and editing can be coded and 55% of the rules would be observed. This challenge will be a flipped classroom in whatever subject where students learn programming.

The ultimate goal of this paper is to share this problem-situation case to encourage the design and implementation of problem-based learning that allows students to:

- Apply content from more than one Computer Science Engineering subject,
- Stimulate interest in learning programming,
- Develop general Computer Science Engineering competencies to face professional challenges,
- Generate useful, open, low-cost software products.

References

Barrows, H.S. (1996). Problem-based learning in medicine and beyond: A brief overview. In L. Wilkerson, & W. H. Gijselaers (Eds.), New directions for teaching and learning, No. 68 (pp. 3-11). San Francisco: Jossey-Bass.

Boldyreff, C., Burd, E., Donkin, J., & Marshall, S. (2001). The case for the use of plain English to increase web accessibility. In Web Site Evolution. Proceedings of 3rd International Workshop on IEEE, 42-48.

Crawley, E., Malmqvist, J., Ostlund, S., & Brodeur, D. (2007). Rethinking engineering education. The CDIO Approach, 302, 60-62.

Edström, K., & Kolmos, A. (2014). PBL and CDIO: complementary models for engineering education development. European Journal of Engineering Education, 39(5), 539-555.

García-Cabrera, L. (2018). Metodología Remediavagos para Escribir Textos Eficientes para la Web: 20 buenas prácticas. En En R. Marfil-Carmona, S. Osuna-Acedo y P. González-Aldea (eds.), Innovación y esfuerzo investigador en la Educación Mediática contemporánea (pp. 25-38). Sevilla: Egregius y Grupo de Investigación GICID de la Universidad de Zaragoza. ISBN 978-84-17270-33-9.

Job, M. A., & Ogalo, H. S. (2012). Micro learning as innovative process of knowledge strategy. *International journal of scientific & technology research, 1*(11), 92-96.

Jonassen, D. H., & Hung, W. (2008). All problems are not equal: Implications for problem-based learning. Interdisciplinary Journal of Problem-Based Learning, 2(2), 4.

Krug, S. (2013). Don't make me think!: a common sense approach to Web usability. New Riders; Edición: 3rd revised edition.

Loranger, H. & Nielsen, J. (2017). Microcontent: A Few Small Words Have a Mega Impact on Business. Retrieved March 01, 2019, from: https://www.nngroup.com/articles/microcontent-how-to-write-headlines-page-titles-and-subject-lines/

Morkes, J., & Nielsen, J. (1997). Concise, scannable, and objective: How to write for the Web. Useit. com. Retrieved March 01, 2019, from: https://www.nngroup.com/articles/concise-scannable-and-objective-how-to-write-for-the-web/

Nielsen, J. (2008). How little do users read? Retrieved March 01, 2019, from: http://www.nngroup.com/articles/how-little-do-users-read/.

Sinnott, J. D. (1989). A model for solution of ill-structured problems: Implications for everyday and abstract problem solving. In J. D. Sinnott (Ed.), Everyday problem solving: Theory and applications (pp. 72-99). New York: Praeger.

Woolley, A. W., Chabris, C. F., Pentland, A., Hashmi, N., & Malone, T. W. (2010). Evidence for a collective intelligence factor in the performance of human groups. *science, 330*(6004), 686-688.