Adaptando textos para a clase de inglés con software libre.

David González Gándara
Mestre no CEIP Emilia Pardo Bazán, de Leiro

Resumen

Programar software pode ser algo que meta medo ó principio, pero enfocado desde un punto de vista de resolución de problemas, e coa axuda adecuada, é algo ó alcance de todo o mundo.
KEYWORDS: linguaxes de programación, algoritmos, entornos de traballo.

Programming software con be something scary in the begining, but focusing from the point of view of problem solving, and with the right kind of help, is something available to anyone..
KEYWORDS: programming languages, algorythms, working environments.

Identificando problemas

No traballo diario dos profesores xorden todo tipo de problemas. Algúns deles poden resolverse por medio de programas informáticos. Desgraciadamente, non sempre existe o programa que resolve o noso problema, ou de existir non fai exactamente o que nos gustaría. Unha vez que identificamos un problema, e non existe un programa que o resolva por nós, ás veces podemos aforrar moitas horas de traballo escribindo nós mesmos ese programa. O exemplo que vou relatar aquí, que é o que da título a esta ponencia, é a adaptación de textos para o seu uso nas clases de inglés.

O meu problema era que, á hora de escoller textos para traballar na aula, nunca tiña claro como de difíciles ían resultar para o alumnado. Aínda que co tempo se desenvolve un especie de sexto sentido para isto, eu prefería unha forma automática de tomar decisións. O núcleo do problema era o número de palabras que aparecen nun texto que poden ser excesivamente difíciles para un grupo de alumnos en concreto. Despois dun período de investigación atopei unha boa idea: decidir unha lista de vocabulario *fácil* e contrastar os textos con esta lista. Un texto

sería difícil se ten moitas palabras que non se atopan na lista *fácil*, e do mesmo xeito un texto sería *fácil* se as palabras que ten están na súa maior parte na lista. Atopei outras experiencias previas de facer este tipo de listas, coma por exemplo a descrita por Ogden (1937), ou a lista de *Voice of America*, unha emisora de radio destinada a xente que quere aprender inglés. Tamén están dispoñibles online as listas de frecuencia de varios corpus británicos, así como as que utilizan os chineses para os seus sistemas de escribir con ordenadores. Experiencias máis mediáticas foron o famoso *inglés con 1000 palabras* do profesor Maurer, ou o máis reciente, para a lingua chinesa *8belts*, do fisterrán Anxo Pérez. Con respecto a estes dous últimos non tiven oportunidade de consultalos nin utilizalos na miña experiencia debido ó seu carácter comercial.

Unha vez consultados os que sí estaban á miña disposición, e despois dun estudo comparativo de todos eles, cheguei á miña propia lista, que denominei *MOLE*, como acrónimo das palabras *more* and *less*, transmitindo a idea de conseguir maior comunicación con menos palabras. Xa estaba listo para comezar o traballo de automatizar a idea nun programa de software.

Planificación. Entornos de traballo.

É o momento de tomar varias decisións importantes. A primeira delas, en que entorno de traballo queremos insertar o noso programa. Hai varias opcións que pensei: podía ser un programa de consola, consistente nun comando que hai que teclear; podía ser un programa que funcionase cun entorno gráfico propio, xa sexa no PC, nun móbil ou nunha tablet; podía ser un programa web, que funcionase nun navegador; ou ben funcionar como módulo ou macro doutro programa xa feito. Cada unha destas opcións implica facer o programa cun determinado linguaxe de programación. Por exemplo, para funcionar como app nun móbil ou tablet, é necesario utilizar a versión de Java destinada a estes dispositivos. Para funcionar nun navegador, hai que utilizar *html*, *PHP*, *python*, *Flash*, ou outras linguaxes que o navegador sexa quen de interpretar. Para funcionar como macro ou como módulo hai que utilizar a linguaxe que o programa utilizado interprete.

Eu tiña varias prioridades. A primeira delas, que o programa fose o máis multiplataforma e cómodo posible, é dicir, que se puidese utilizar desde calquera dispositivo. A segunda, utilizar unicamente software libre.

Quería que fose multiplataforma porque moitas veces sería moi práctico poder utilizar o programa desde ordenadores con sistemas operativos diferentes, ou desde o móvil. Con respecto á comodidade sería ideal, por exemplo, que as tarefas de escribir un texto ou copialo para despois comparalo coa lista de vocabulario se fixese dentro do mesmo entorno, sen necesidade de usar varios programas distintos

Tamén quería utilizar unicamente ferramentas de software libre. A filosofía difundida pola Fundación do Software Libre encaixa perfectamente co mundo educativo. Software Libre non significa gratuíto. Significa que o usuario é libre de acceder ó seu código fonte e modificalo segundo as súas necesidades. Igualmente, este modo de facer as cousas propicia que os coñecementos das distintas persoas que participan no deseño dun programa, e os de todas as que o modifican únense para crear un programa mellor. Habitualmente a xente tende a pensar que os programas comerciais van funcionar mellor, porque o Software Libre non é profesional. Isto non ten por que ser certo. Algunhas veces, os programas de Software Libre son escritos por empresas, como sucede con *OpenOffice*, por exemplo; outras veces, aínda non sendo un traballo profesional (en canto a comercial), funciona mellor que as alternativas comerciais. Traballando con Software Libre, e escribindo Software Libre estamos promocionando valores positivos. O software que esconde as fontes en certo modo, unicamente pode contribuír a valores negativos. No entorno educativo está moi claro, pois se o noso alumnado necesita utilizar un software privativo que nós temos, só existen dúas opcións, igualmente malas: ou ben llo negamos, fallando así ás nosas obrigas coas persoas achegadas, ou lle dicimos que sí, incumplindo así a licenza do programa. Aínda que existe unha terceira opción, que é pagar nós mesmos as licenzas para a xente que nos pide o programa, resulta bastante caro e innecesario.

Logo de estudar todas as posibilidades que tiña, decidín probar varias das alternativas. Escribiría unha app para móbil e un programa web. Despois, no propio proceso de escribir o código, empecei a coñecer a fondo un programa que non valorara demasiado inicialmente, *emacs*. A priori, este programa parece un simple editor de textos, pero estudándoo máis a fondo, descubrín que o que realmente o fai interesante, é a súa tremenda potencia de programación modular. Esta se realiza mediante a linguaxe *LISP*, que eu non coñecía, pero que resulta tremendamente sinxela. Programar un módulo de emacs para comparar textos cuna lista de palabras cumplía todas as condicións iniciais que me propuxera. *Emacs* funciona tanto en PC como en móbiles e tablets, tanto en modo consola como gráfico, é unha ferramenta libre, e ademais, foi programada polo propio Richard Stallman, fundador do Software Libre e de *GNU*, co cal, fomentando o uso deste programa difúndese toda a filosofía do Software Libre coa que me identifico totalmente.

Polo tanto, finalmente escribín o programa en *Java*, e en *PHP*. Escribín tamén un módulo de *emacs*, en *LISP*, que simplemente chama ó programa en *PHP*. A continuación pódense ver os tres programas para comparar as linguaxes de programación.

Nos listados pódese ver unicamente parte do ficheiro principal de *Java* para *Android*, xa que un programa de *Android* ten que ter unha estrutura concreta e bastante complexa de ficheiros. Ademáis, como xa mencionei, os programas de *Java* adoitan ter unha lonxitude, na miña opinión, absurdamente lon-

ga. Apréciase claramente que só unha parte do programa en *Java* é máis longo que o programa completo en *PHP*. O código completo do programa en *Java* atópase dispoñible no enderezo *https://github.com/mrrookes/MOLEanalysis*. Tamén se mostra o código do módulo *LISP* que permite lanzar o programa *PHP* desde *emacs*. O programa en *PHP* , funcionando en entorno web, pódese probar en *http://www.laretlernejo.org.es/mole_analysis.php*.

Listing 1 Extracto do programa en Java.

```
1  package laretlernejo.moleanalysis;
2
3  import android.app.*;
4  import android.os.*;
5  import android.view.*;
6  import android.widget.*;
7  import android.content.*;
8  import java.io.*;
9  import java.util.*;
10
11 public class MainActivity extends Activity
12 {
13     public String message;
14     @Override
15     public void onCreate(Bundle savedInstanceState)
16         {
17         super.onCreate(savedInstanceState);
18         setContentView(R.layout.main);
19     }
20     public void onClick(View view){
21         EditText text = (EditText)findViewById(R.id.edit_box);
22         message = text.getText().toString();
23         String resultado = MOLEanalysis(message, "MOLE.xml");
24         AlertDialog.Builder dialogBuilder = new AlertDialog.Builder(this
               );
25         dialogBuilder.setMessage(resultado);
26         dialogBuilder.setCancelable(true).setTitle("Analysis");
27         dialogBuilder.create().show();
28     }
29     private String MOLEanalysis(String texto, String filename){
30
31         List<Word> words=null;
32         String resultado = null;
33         List<Word>palabras_conocidas = new ArrayList<Word>();
34         List<Word>palabras_desconocidas= new ArrayList<Word>();
35         List<Word>words_para_analizar = new ArrayList<Word>();
36         List<Word>texto_analisis_arreglado= new ArrayList<Word>();
37
38         try{
39            DataHandler parser = new DataHandler();
40            words = parser.parse(getAssets().open(filename));
```

```
41
42          }catch (IOException e){
43                  e.printStackTrace();
44          }
45
46          String []texto_analisis = texto.split("\\s+");
47          for(int i = 0; i<texto_analisis.length; i++)
48          {
49             Word newWord = new Word();
50             newWord.setWord(texto_analisis[i]);
51             words_para_analizar.add(newWord);
52          }
53
54          for(int i=0; i< words_para_analizar.size(); i++)
55          {
56             for(int j=0; j<words.size(); j++)
57             {
58
59               for(int x = 0; x<cosas_para_quitar.length;x++){
60                  elemento1 = elemento1.replace(cosas_para_quitar[x], "");
61                  elemento2 = elemento2.replace(cosas_para_quitar[x], "");
62               }
63
64               Word elemento1_word = new Word();
65               Word elemento2_word = new Word();
66               elemento1_word.setWord(elemento1);
67               elemento2_word.setWord(elemento2);
68
69               if(!texto_analisis_arreglado.contains(elemento1_word))
70               {
71                  texto_analisis_arreglado.add(elemento1_word);
72               }
73          }
74      }
75  }
```

Listing 2 Programa completo en php.

```
1  #/usr/bin/php -q
2
3  <?php
4  $coinciden = 0;
5  $filename = "MOLE.xml";
6  $xml =file_get_contents("MOLE.xml");
7  $data_analisis = simplexml_load_string($xml);
8
9  array_shift($argv);
10
11 if(!is_null($argv[0]))
12 {
13    $coinciden = 0;
```

```
14      $texto = file_get_contents($argv[0]);
15      $texto_analisis = explode(' ',$texto);
16      $longitud_texto = count($texto_analisis);
17      $longitud_data = count($data_analisis);
18      $palabras_conocidas = array();
19      $texto_analisis_arreglado = array();
20      printf("Comezando analise....\n");
21      for($i=0; $i<=$longitud_texto-1; $i++)
22      {
23
24          for($j=0; $j<=$longitud_data-1; $j++)
25          {
26            $cosas_para_quitar = array(".",",","","?","!","\\","/","-");
27            $tmp_elemento1 = strtolower($texto_analisis[$i]);
28            $tmp_elemento2 = strtolower($data_analisis->word[$j]);
29            $elemento1 = $tmp_elemento1;
30            $elemento2 = $tmp_elemento2;
31            $tmp_elemento1 = str_replace($cosas_para_quitar, "",
                  $tmp_elemento1);;
32            $tmp_elemento2 = str_replace($cosas_para_quitar, "",
                  $tmp_elemento2);;
33            $elemento1 = $tmp_elemento1;
34            $elemento2 = $tmp_elemento2;
35            if(!in_array($elemento1, $texto_analisis_arreglado))
36            {
37                    $texto_analisis_arreglado[] = $elemento1;
38            }
39            $comparacion = strcmp($elemento1, $elemento2);
40            if($comparacion==0 && !in_array($elemento1, $palabras_conocidas)
                  )
41            {
42                    $palabras_conocidas[] = $elemento1;
43                    $coinciden++;
44            }
45          }
46      }
47
48      $palabras_desconocidas = array_diff($texto_analisis_arreglado,
            $palabras_conocidas);
49      $porcentaje = $coinciden*100/$longitud_texto;
50      printf("O texto ten %s palabras\n", $longitud_texto);
51      printf("%s foron atopadas na base de datos MOLE (%s por cento)\n",
            $coinciden, $porcentaje);
52      $desconocidas = implode(", ", $palabras_desconocidas);
53      printf("Non atopei as seguintes palabras: %s", $desconocidas);
54  }
55  ?>
```

Listing 3 Código completo do módulo en LISP.

```
1  (defun MOLE-analyze ()
```

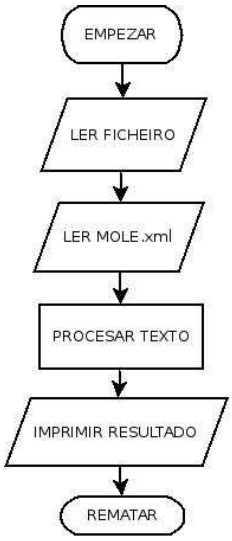

Figura 1. Funcionamento xeral do programa

```
2     (shell-command (concat (expand-file-name "MOLEanalysis.php")
          buffer-file-name)))
3
4  (global-set-key (kbd "C-x_C-M") 'MOLE-analyze)
```

Algoritmos

Unha vez finalizada a parte filosófica, hai que poñerse ó traballo. Primeiro hai que pensar o algoritmo, é dicir, que vai a facer exactamente o noso programa. O que eu recomendo é escribir un diagrama de fluxo. Para elo atopei moi cómodo o programa *Dia*, que permite de maneira moi simple facer o noso diagrama. Non me estenderei en explicar o que é un diagrama de fluxo, nas figuras 1 e 2 pódese ver a idea. Un diagrama permítenos ir escribindo o código dunha maneira coherente, sin dar tumbos.

Escribindo o código. Fontes de axuda.

Dependendo da linguaxe de programación que escollemos no primeiro paso, utilizaremos un programa distinto para escribir o código. Para escribir *Java* para dispositivos móbiles, penso que *AIDE* é a mellor opción, proporcionándonos automáticamente a estrutura de ficheiros requerida, etc. Para *PHP*, calquera editor de textos vale, aínda que un que recoñeza as palabras clave e as resalte pode resultar conveniente. *Emacs* pode facer esta función, ou ben *Gedit*.

O propósito principal desta ponencia é mostrar que calquera persoa pode escribir os seus programas, así que explicarei como fixen eu para escribir o código

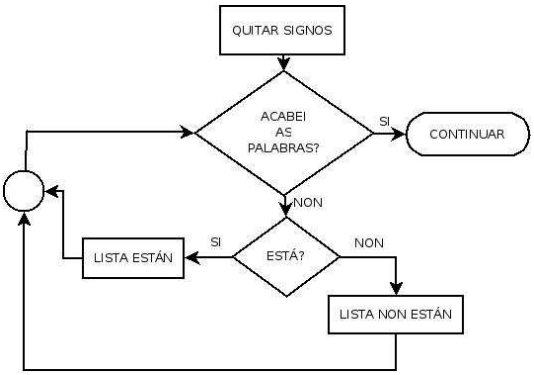

Figura 2. Análise das palabras do texto

sen dominar as linguaxes de programación. A única que coñecía algo era php. Para isto, o noso axudante será o buscador de internet. Aínda que a *Fundación do Software Libre* non recomenda *Google*, teño que recoñecer que eu o uso case todo o tempo. Como primeiro paso, podemos buscar un programa de exemplo o máis sinxelo posible, que nos permita comprender a groso modo como funciona esa linguaxe. Descubriremos que algunhas linguaxes, como python, son realmente fáciles de comprender sen saber nada de programación. Outras non tanto, coma *Java*, que na miña opinión é absurdamente complicado para os non programadores. A linguaxe utilizada por *Emacs*, *LISP*, é un pouco liosa, utilizando moitos parénteses que parecen innecesarios, pero unha vez que se consultan varios exemplos resulta sorprendentemente sinxela.

Despois de mirar os exemplos que baixamos escolleremos algún programa base que se pareza en esencia ó que nós queremos facer, e faremos as modificación oportunas para que faga o que nós queremos. Probablemente, co tempo, deixaremos de utilizar tantos exemplos e faremos as cousas desde cero. Hai numerosos foros nos que se poden consultar dúbidas que se nos presentan ó escribirmos o noso código. A xente da comunidade habitualmente estará encantadas de axudarnos. Aínda que escribir e ler en inglés facilita atopar recursos, hai suficientes comunidades en español como para que non necesitemos o inglés para nada. De todos modos, a maior parte das veces alguén fixo xa as cousas que imos facer nós, co cal non teremos máis que copiar partes do código dos seus programas. A filosofía do Software Libre a pleno rendemento. Precisamente por iso, o lóxico é que tamén os nosos programas sexan libres para así facer cada vez máis grande esta comunidade.

Probando. Debug.

Chegou o momento do proceso máis ingrato de todos, pero tamén o máis reconfortante. Cando xa temos o primeiro prototipo do programa escrito, hai que poñelo

a funcionar. Rara vez funciona á primeira, os erros sintácticos son tremendamente frecuentes. Os programas que recomendaba máis arriba, que recoñecen as palabras clave de cada linguaxe, axúdannos moito, porque se non escribimos algunha orde de maneira correcta, non sae na cor que debería, axudándonos a non cometer eses erros. Ademais de correxir os erros de sintaxe por escribir mal algunha orde, tamén existen aqueles nos que lle pedimos algo a unha función de maneira incorrecta ou lle pedimos algo que non pode facer. Algúns programas recoñecen estes erros, pero darannos unha mensaxe que ó mellor non entendemos, do tipo: "WARNING: variable type mismatch", ou cousas parecidas. Nestes casos, podemos recurrir novamente á comunidade, entrando nun foro e preguntando.

Esta fase do proceso é a máis longa e penosa, pero os momentos nos que logramos que as partes do programa vaian funcionando, son as que nos dan realmente a enerxía necesaria para seguir traballando. O que non se sinta ben cando logrou facer funcionar un programa, seguramente non sexa quen de sobrevivir ó proceso, e fará mellor en utilizar programas xa feitos.

Conclusión.

Por medio desta experiencia quería transmitir varias ideas importantes. A primeira delas, é que programar está ó alcance de calquera, sempre que atope as ferramentas e a axuda adecuada. En moitas ocasións vainos axudar a resolver problemas que doutra forma nos fai adaptarnos a unha solución pouco satisfactoria. Exemplos de problemas que me teñen xurdido a min ou a compañeiros: facer unha base de datos dos nosos alumnos que faga unhas funcións moi concretas; reducir todas as fotos dunha carpeta a un tamaño concreto (problema que calquera que xestione a páxina web dun colexio da Xunta terá atopado); escribir en grego antiguo; facer automáticamente follas de exercicios para o alumnado adaptadas ás nosas necesidades; etc.

A segunda, que o Software Libre é moito máis que conseguir programas gratuítos sen necesidade de piratealos. É unha maneira de vida, que probablemente temos a obriga de contaxiar a toda a xente que podamos.

Referencias

Ogden, C. (1937). *Basic english, a general introduction with rules and grammar*. London: K. Paul,Trench, Trubner and Co., Ltd.