**Diseño**

El programa B2 extenderá el programa B1 declarando una nueva clase, class\_based\_loc\_counter, heredando de simple\_loc\_counter. Rastreará el número de características en cada método contando puntos y comas en la declaración de clase (en caso de que las declaraciones de características fluyan sobre los límites de la línea) y contará LOC en métodos y clases rastreando los niveles de inicio / fin de bloque. Los niveles de inicio / fin de bloque serán rastreados por un contador de enteros, al igual que los niveles de inicio / fin de comentarios fueron rastreados en el programa B1.

Haré modificaciones menores al código original del programa B1. La clase simple\_loc\_counter tendrá una refactorización menor, dividiendo las comprobaciones de inicio / fin del bloque en una comprobación de inicio y finalización; Los niveles de anidación de bloques se agregarán a esa clase. El programa principal requerirá cambios muy pequeños, simplemente cambiando la clase del objeto loc\_counter.

**Código**

**simple\_input\_parser.h**

|  |
| --- |
| /\*  \*/  #ifndef SIMPLE\_INPUT\_PARSER\_H  #define SIMPLE\_INPUT\_PARSER\_H  #include <string>  #include <iostream>  //a standard "framework" for parsing a set of input lines  class simple\_input\_parser  {  public:  //sets input stream  void set\_input\_stream (istream \* new\_input);  //parse input until EOF  void parse\_until\_eof (void);  //reads single line from input, transforms it, stores it in last\_line  void read\_line (void);  //sets the last line read  void set\_last\_line (const std::string & new\_line);  //last line read from input  const std::string & last\_line (void) const;  //parses the last line read  virtual void parse\_last\_line (void);  //returns a "transformed" copy of the given line  virtual std::string transformed\_line (const std::string & line) const;  //constructor  simple\_input\_parser (void);  //virtual destructor  virtual ~ simple\_input\_parser (void);  //resets the parser  virtual void reset (void);  private:  //the input stream  istream \* m\_input\_stream;  //the last line of input  std::string m\_last\_line;  };  #endif  /\*  \*/ |

**simple\_input\_parser.cpp**

|  |
| --- |
| /\*  \*/  #include "simple\_input\_parser.h"  #ifndef CONTRACT\_H  #include "contract.h"  #endif  void  simple\_input\_parser::set\_input\_stream (istream \* new\_input)  {  m\_input\_stream = new\_input;  }  void  simple\_input\_parser::parse\_until\_eof (void)  {  REQUIRE (m\_input\_stream != NULL);  while (!(m\_input\_stream->eof ()))  {  read\_line ();  parse\_last\_line ();  }  }  void  simple\_input\_parser::read\_line (void)  {  REQUIRE (!(m\_input\_stream->eof ()));  const int  input\_buffer\_size = 255;  char  input\_buffer[input\_buffer\_size];  m\_input\_stream->getline (input\_buffer, input\_buffer\_size);  //for some reason, the G++ standard library needs me to do this or it  //doesn't register the EOF condition properly. This makes no sense  //to me...  char c = m\_input\_stream->get();  m\_input\_stream->putback( c );  std::string input\_line (input\_buffer);  //no source code line should be longer than 255!  CHECK (input\_line.size () < 255);  set\_last\_line (transformed\_line (input\_line));  }  void  simple\_input\_parser::set\_last\_line (const std::string & new\_line)  {  m\_last\_line = new\_line;  }  const  std::string & simple\_input\_parser::last\_line (void) const  {  return m\_last\_line;  }  void  simple\_input\_parser::parse\_last\_line (void)  {  //basic version does nothing  }  std::string simple\_input\_parser::transformed\_line (const std::string & line) const  {  return line;  }  simple\_input\_parser::simple\_input\_parser (void)  {  reset ();  }  simple\_input\_parser::~simple\_input\_parser (void)  {  }  void  simple\_input\_parser::reset (void)  {  m\_input\_stream = NULL;  m\_last\_line = "";  }  /\*  \*/ |

**simple\_loc\_counter.h**

|  |
| --- |
| /\*  \*/  #ifndef SIMPLE\_LOC\_COUNTER\_H  #define SIMPLE\_LOC\_COUNTER\_H  #ifndef SIMPLE\_INPUT\_PARSER\_H  #include "simple\_input\_parser.h"  #endif  #include <string>  #include <vector>  //subclass of simple\_input\_parser that stores countable lines of code in a buffer  //and can return their count.  class simple\_loc\_counter:public simple\_input\_parser  {  public:  //adds last line to the buffered lines if it is countable  void parse\_last\_line (void);  //the count of LOC  int loc\_count (void) const;  //whether last line was comment  bool last\_line\_is\_comment (void) const;  //whether last line was compiler directive  bool last\_line\_is\_compiler\_directive (void) const;  //whether we are in a block comment  bool is\_in\_block\_comment (void) const;  //whether last line was a block begin  bool last\_line\_is\_block\_begin (void) const;  //whether last line was a block end  bool last\_line\_is\_block\_end (void) const;  //whether last line was part of a begin/end pair  bool last\_line\_is\_block\_begin\_or\_end (void) const;  //whether the last line was countable  bool last\_line\_is\_countable (void) const;  //whether the last line was empty  bool last\_line\_is\_empty (void) const;  //updates the block comment count  void update\_block\_comment\_count (void);  //updates the block nesting level  void update\_block\_nesting\_level (void);  //are we in a block?  bool is\_in\_block (void) const;  //block nesting level  int block\_nesting\_level (void) const;  //whether the last line starts with the given string  bool last\_line\_starts\_with (const std::string & search\_string) const;  //whether a given string starts with a search string  static bool string\_starts\_with (const std::string & given\_string,  const std::string & search\_string);  //returns the input string stripped of leading/trailing whitespace    std::string string\_stripped\_of\_whitespace (const std::string & input\_string) const;  //returns the transformed line (here, the line stripped of whitespace)  virtual std::  string transformed\_line (const std::string & input\_string) const;  //constructor  simple\_loc\_counter (void);  //destructor  virtual ~ simple\_loc\_counter (void);  //resets the object  virtual void reset (void);  //writes the countable lines to the given output stream  void write\_countable\_lines (ostream & ostr) const;  protected:  //the buffered countable lines  std::vector < std::string > m\_countable\_lines;  //the "block comment" nesting level  int m\_block\_comment\_nesting\_level;  //the "block" nesting level  int m\_block\_nesting\_level;  //the beginning of a block comment  static const std::string & block\_comment\_begin;  //the end of a block comment  static const std::string & block\_comment\_end;  //the beginning of an inline comment  static const std::string & inline\_comment\_begin;  //the beginning of a compiler directive  static const std::string & compiler\_directive\_begin;  //the "begin block" string  static const std::string & block\_begin;  //the "end block" string  static const std::string & block\_end;  //whitespace characters  static const std::string & whitespace\_characters;  };  #endif  /\*  \*/ |

**simple\_loc\_counter.cpp**

|  |
| --- |
| /\*  \*/  #include "simple\_loc\_counter.h"  #ifndef YAK\_MIN\_MAX\_H  #include "yak\_min\_max.h"  #endif  #ifndef CONTRACT\_H  #include "contract.h"  #endif  void  simple\_loc\_counter::parse\_last\_line (void)  {  if (last\_line\_is\_countable ())  {  m\_countable\_lines.push\_back (last\_line ());  }  update\_block\_comment\_count ();  update\_block\_nesting\_level ();  }  int  simple\_loc\_counter::loc\_count (void) const  {  return m\_countable\_lines.size ();  }  bool  simple\_loc\_counter::last\_line\_is\_comment (void) const  {  bool Result = false;  if (last\_line\_starts\_with (block\_comment\_begin)  || last\_line\_starts\_with (inline\_comment\_begin)  || is\_in\_block\_comment ())  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::last\_line\_is\_compiler\_directive (void) const  {  bool Result = false;  if (last\_line\_starts\_with (compiler\_directive\_begin))  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::is\_in\_block\_comment (void) const  {  bool Result = false;  if (m\_block\_comment\_nesting\_level > 0)  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::last\_line\_is\_block\_begin (void) const  {  bool  Result = false;  if (last\_line\_starts\_with (block\_begin))  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::last\_line\_is\_block\_end (void) const  {  bool  Result = false;  if (last\_line\_starts\_with (block\_end))  {  Result = true;  }  return Result;  }  bool  simple\_loc\_counter::last\_line\_is\_block\_begin\_or\_end (void) const  {  return (last\_line\_is\_block\_begin () | last\_line\_is\_block\_end ());  }  bool  simple\_loc\_counter::last\_line\_is\_empty (void) const  {  return (last\_line ().length () == 0);  }  bool  simple\_loc\_counter::last\_line\_is\_countable (void) const  {  bool Result = true;  if ((last\_line\_is\_comment ())  || (last\_line\_is\_block\_begin\_or\_end ())  || (last\_line\_is\_compiler\_directive ()) || (last\_line\_is\_empty ()))  {  Result = false;  }  return Result;  }  void  simple\_loc\_counter::update\_block\_comment\_count (void)  {  //count through the string; add 1 to the block comment count if the begin  //string is encountered, subtract one if the end string is encountered.  for (unsigned int i = 0; i < last\_line ().length (); ++i)  {  std::string line\_remaining =  last\_line ().substr (i, last\_line ().length ());  if (string\_starts\_with (line\_remaining, block\_comment\_begin))  {  ++m\_block\_comment\_nesting\_level;  }  else if (string\_starts\_with (line\_remaining, block\_comment\_end))  {  --m\_block\_comment\_nesting\_level;  }  }  }  void  simple\_loc\_counter::update\_block\_nesting\_level (void)  {  if (last\_line\_is\_block\_begin ())  {  ++m\_block\_nesting\_level;  }  else if (last\_line\_is\_block\_end ())  {  --m\_block\_nesting\_level;  }  ENSURE (m\_block\_nesting\_level >= 0);  }  bool simple\_loc\_counter::is\_in\_block (void) const  {  return (m\_block\_nesting\_level > 0);  }  int  simple\_loc\_counter::block\_nesting\_level (void) const  {  return m\_block\_nesting\_level;  }  bool  simple\_loc\_counter::  last\_line\_starts\_with (const std::string & search\_string) const  {  return string\_starts\_with (last\_line (), search\_string);  }  bool  simple\_loc\_counter::string\_starts\_with (const std::string & given\_string,  const std::string & search\_string)  {  int  substring\_size =  yak\_min (given\_string.length (), search\_string.length ());  std::string substring = given\_string.substr (0, substring\_size);  bool Result = (substring == search\_string);  return Result;  }  std::string  simple\_loc\_counter::string\_stripped\_of\_whitespace (const std::  string & input\_string) const  {  std::string::size\_type start =  input\_string.find\_first\_not\_of (whitespace\_characters);  if (start == input\_string.npos)  {  start = 0;  }  std::string::size\_type end =  input\_string.find\_last\_not\_of (whitespace\_characters);  if (end == input\_string.npos)  {  end = input\_string.length ();  }  if (end < input\_string.length ())  {  ++end;  }  std::string Result = input\_string.substr (start, end);  return Result;  }  std::string  simple\_loc\_counter::transformed\_line (const std::string & input\_string) const  {  return string\_stripped\_of\_whitespace (input\_string);  }  simple\_loc\_counter::simple\_loc\_counter (void)  {  reset ();  }  simple\_loc\_counter::~simple\_loc\_counter (void)  {  }  void  simple\_loc\_counter::reset (void)  {  m\_countable\_lines.clear ();  m\_block\_comment\_nesting\_level = 0;  m\_block\_nesting\_level = 0;  }  void  simple\_loc\_counter::write\_countable\_lines (ostream & ostr) const  {  for (std::vector < std::string >::const\_iterator iter =  m\_countable\_lines.begin (); iter != m\_countable\_lines.end (); ++iter)  {  ostr << \*iter << "\n";  }  }  const  std::string & simple\_loc\_counter::block\_comment\_begin = std::string( "/" ) + std::string ("\*");  const  std::string & simple\_loc\_counter::block\_comment\_end = std::string( "\*" ) + std::string( "/" );  const  std::string & simple\_loc\_counter::inline\_comment\_begin = "//";  const  std::string & simple\_loc\_counter::compiler\_directive\_begin = "#";  const  std::string & simple\_loc\_counter::block\_begin = "{";  const  std::string & simple\_loc\_counter::block\_end = "}";  const  std::string & simple\_loc\_counter::whitespace\_characters = " \t\n\0x32";  /\*  \*/ |

**class\_metric.h**

|  |
| --- |
| /\*  \*/  #ifndef CLASS\_METRIC\_H  #define CLASS\_METRIC\_H  class class\_metric  {  public:  //LOC in the class  int loc;  //number of features  int feature\_count;  class\_metric (void);  };  #endif  /\*  \*/ |

**class\_metric.cpp**

|  |
| --- |
| /\*  \*/  #include "class\_metric.h"  class\_metric::class\_metric (void):  loc (0), feature\_count (0)  {  }  /\*  \*/ |

**class\_based\_loc\_counter.h**

|  |
| --- |
| /\*  \*/  #ifndef CLASS\_BASED\_LOC\_COUNTER\_H  #define CLASS\_BASED\_LOC\_COUNTER\_H  #ifndef SIMPLE\_LOC\_COUNTER\_H  #include "simple\_loc\_counter.h"  #endif  #ifndef CLASS\_METRIC\_H  #include "class\_metric.h"  #endif  #include <map>  class class\_based\_loc\_counter:public simple\_loc\_counter  {  protected:  //mapping of class names to metrics  std::map < std::string, class\_metric > m\_class\_map;  //are we in a class declaration?  bool m\_in\_class\_declaration;  //are we in a class feature implementation?  bool m\_in\_class\_feature;  //name of the class we're in  std::string m\_current\_class\_name;  static const std::string class\_begin;  static const std::string scope\_operator;  std::string m\_previous\_line;  public:  //adds the last line to method/class counts as appropriate  virtual void parse\_last\_line (void);  std::string class\_name\_from\_class\_begin (const std::string & str) const;  std::string class\_name\_from\_external\_feature (const std::string & str) const;  bool last\_line\_is\_class\_begin (void) const;  bool last\_line\_is\_class\_end (void) const;  bool last\_line\_is\_feature\_begin (void) const;  bool last\_line\_is\_feature\_end (void) const;  bool last\_line\_is\_data\_feature (void) const;  bool last\_line\_declares\_feature (void) const;  bool last\_line\_contains\_scope\_operator (void) const;  bool last\_line\_contains\_known\_class\_before\_scope\_operator (void) const;  void adjust\_class\_feature\_count (const std::string & class\_name,  int adjustment);  void adjust\_class\_loc\_count (const std::string & class\_name,  int adjustment);  bool has\_entry\_for\_class\_name (const std::string & class\_name) const;  const std::map < std::string, class\_metric > &class\_map (void) const;  class\_based\_loc\_counter (void);  virtual void reset (void);  };  #endif  /\*  \*/ |

**class\_based\_loc\_counter.cpp**

|  |
| --- |
| /\*  \*/  #include "class\_based\_loc\_counter.h"  #ifndef CONTRACT\_H  #include "contract.h"  #endif  const  std::string class\_based\_loc\_counter::class\_begin ("class ");  const  std::string class\_based\_loc\_counter::scope\_operator ("::");  class\_based\_loc\_counter::class\_based\_loc\_counter (void):  simple\_loc\_counter ()  {  reset ();  }  void  class\_based\_loc\_counter::reset (void)  {  simple\_loc\_counter::reset();  m\_class\_map.clear ();  m\_in\_class\_declaration = false;  m\_in\_class\_feature = false;  m\_current\_class\_name = "UNSET";  m\_previous\_line = "";  }  void  class\_based\_loc\_counter::parse\_last\_line (void)  {  simple\_loc\_counter::parse\_last\_line ();  //cout << m\_class\_map[ "class\_based\_loc\_counter" ].loc << ":" << m\_countable\_lines.size() << ":" << last\_line() << "\n";  if (last\_line\_is\_countable ())  {  if (last\_line\_is\_class\_begin ())  {  CHECK (m\_in\_class\_declaration == false);  m\_in\_class\_declaration = true;  m\_current\_class\_name = class\_name\_from\_class\_begin (last\_line ());  }  if (m\_in\_class\_declaration)  {  adjust\_class\_loc\_count (m\_current\_class\_name, 1);  if (last\_line\_declares\_feature ())  {  adjust\_class\_feature\_count (m\_current\_class\_name, 1);  }  }  if (last\_line\_is\_feature\_begin ())  {  //cout << "<<begin feature; previous line = " << m\_previous\_line << ">>";  CHECK (m\_in\_class\_feature == false);  m\_in\_class\_feature = true;  m\_current\_class\_name =  class\_name\_from\_external\_feature (last\_line ());  //add one for the return type declaration, as indent usu puts it on the previous line  if ((m\_previous\_line.length () > 0))  {  //cout << "\*" << m\_previous\_line;  adjust\_class\_loc\_count (m\_current\_class\_name, 1);  }  }  if (m\_in\_class\_feature)  {  adjust\_class\_loc\_count (m\_current\_class\_name, 1);  }  if (last\_line\_is\_data\_feature ())  {  adjust\_class\_loc\_count (class\_name\_from\_external\_feature  (last\_line ()), 1);  if (m\_previous\_line.length () > 0)  {  //cout << "&" << m\_previous\_line;  adjust\_class\_loc\_count (m\_current\_class\_name, 1);  }  }  }  if (last\_line\_is\_class\_end ())  {  CHECK (block\_nesting\_level () == 0);  CHECK (m\_in\_class\_declaration == true);  CHECK (m\_in\_class\_feature == false);  m\_in\_class\_declaration = false;  }  if (last\_line\_is\_feature\_end ())  {  //cout << "<<end feature, nesting:" << block\_nesting\_level() << ">>";  CHECK (block\_nesting\_level () == 0);  CHECK (m\_in\_class\_declaration == false);  CHECK (m\_in\_class\_feature == true);  m\_in\_class\_feature = false;  }  m\_previous\_line = last\_line ();  }  std::string  class\_based\_loc\_counter::  class\_name\_from\_class\_begin (const std::string & str) const  {  REQUIRE (last\_line\_is\_class\_begin ());  std::string::size\_type class\_begin\_position =  last\_line ().find (class\_begin);  CHECK (class\_begin\_position != last\_line ().npos);  std::string::size\_type next\_white\_space\_position =  last\_line ().find\_first\_of (whitespace\_characters, class\_begin\_position);  CHECK (class\_begin\_position != last\_line ().npos);  std::string::size\_type next\_word\_position =  last\_line ().find\_first\_not\_of (whitespace\_characters,  next\_white\_space\_position);  CHECK (next\_word\_position != last\_line ().npos);  std::string whitespace\_characters\_and\_colon = whitespace\_characters;  whitespace\_characters\_and\_colon.append (":");  std::string::size\_type end\_of\_class\_name\_position =  last\_line ().find\_first\_of (whitespace\_characters\_and\_colon,  next\_word\_position);  if (end\_of\_class\_name\_position == last\_line ().npos)  {  end\_of\_class\_name\_position = last\_line ().length ();  }  std::string Result =  last\_line ().substr (next\_word\_position,  end\_of\_class\_name\_position - next\_word\_position);  //cout << Result;  return Result;  }  std::string  class\_based\_loc\_counter::  class\_name\_from\_external\_feature (const std::string & str) const  {  //this is used by last\_line\_is\_feature\_begin, so we can't use it here as  //a requirement... be cautious!  std::string::size\_type scope\_operator\_position = last\_line ().length ();  while (scope\_operator\_position != last\_line ().npos)  {  scope\_operator\_position =  last\_line ().rfind (scope\_operator, scope\_operator\_position - 1);  if (scope\_operator\_position != last\_line ().npos);  {  std::string beginning\_of\_string =  last\_line ().substr (0, scope\_operator\_position);  //cout << "<<" << beginning\_of\_string << ">>";  std::string::size\_type begin =  beginning\_of\_string.find\_last\_of (whitespace\_characters);  if (begin == beginning\_of\_string.npos)  {  begin = 0;  }  else  {  begin = begin + 1;  }  std::string possible\_result =  beginning\_of\_string.substr (begin, scope\_operator\_position);  //cout << "POSS:" << possible\_result;  if (has\_entry\_for\_class\_name (possible\_result))  {  return possible\_result;  }  }  }  return "";  }  bool  class\_based\_loc\_counter::last\_line\_is\_class\_begin (void) const  {  bool Result = false;  if (last\_line\_starts\_with (class\_begin))  {  Result = true;  }  return Result;  }  bool  class\_based\_loc\_counter::last\_line\_is\_class\_end (void) const  {  bool Result = false;  if (last\_line\_is\_block\_end () && m\_in\_class\_declaration)  {  Result = true;  }  return Result;  }  bool  class\_based\_loc\_counter::last\_line\_contains\_scope\_operator (void) const  {  return (last\_line ().find (scope\_operator) != last\_line ().npos);  }  bool  class\_based\_loc\_counter::last\_line\_contains\_known\_class\_before\_scope\_operator  (void) const  {  bool Result = false;  if (last\_line\_contains\_scope\_operator ())  {  std::string class\_name =  class\_name\_from\_external\_feature (last\_line ());  if (has\_entry\_for\_class\_name (class\_name))  {  Result = true;  }  }  return Result;  }  bool  class\_based\_loc\_counter::last\_line\_is\_feature\_begin (void) const  {  bool Result = false;  if (last\_line\_contains\_known\_class\_before\_scope\_operator ())  {  if (!last\_line\_is\_data\_feature ())  {  Result = true;  }  }  return Result;  }  bool  class\_based\_loc\_counter::last\_line\_is\_feature\_end (void) const  {  bool Result = false;  if (last\_line\_is\_block\_end ()  && m\_in\_class\_feature && (block\_nesting\_level () == 0))  {  Result = true;  }  return Result;  }  bool  class\_based\_loc\_counter::last\_line\_is\_data\_feature (void) const  {  bool Result = false;  if (last\_line\_contains\_known\_class\_before\_scope\_operator ())  {  if (last\_line ().find (";") != last\_line ().npos)  {  Result = true;  }  }  return Result;  }  bool  class\_based\_loc\_counter::last\_line\_declares\_feature (void) const  {  bool Result = false;  if (m\_in\_class\_declaration  && (last\_line ().find (";") != last\_line ().npos))  {  Result = true;  }  return Result;  }  bool  class\_based\_loc\_counter::has\_entry\_for\_class\_name (const yak\_string &  class\_name) const  {  bool Result = false;  if (class\_map ().find (class\_name) != class\_map ().end ())  {  Result = true;  }  return Result;  }  void  class\_based\_loc\_counter::  adjust\_class\_feature\_count (const std::string & class\_name, int adjustment)  {  if (!has\_entry\_for\_class\_name (class\_name))  {  m\_class\_map[class\_name] = class\_metric ();  }  m\_class\_map[class\_name].feature\_count += adjustment;  }  void  class\_based\_loc\_counter::  adjust\_class\_loc\_count (const std::string & class\_name, int adjustment)  {  if (!has\_entry\_for\_class\_name (class\_name))  {  m\_class\_map[class\_name] = class\_metric ();  }  m\_class\_map[class\_name].loc += adjustment;  }  const  std::map < std::string,  class\_metric > & class\_based\_loc\_counter::class\_map (void) const  {  return m\_class\_map;  }  /\*  \*/ |

**main.cpp**

|  |
| --- |
| /\*  \*/  #ifndef CLASS\_BASED\_LOC\_COUNTER\_H  #include "class\_based\_loc\_counter.h"  #endif  #ifndef YAK\_EXCEPTION\_H  #include "yak\_exception.h"  #endif  istream \*  input\_stream\_from\_args (int arg\_count, const char \*\*arg\_vector)  {  istream \*Result = NULL;  if (arg\_count == 1)  {  Result = &cin;  }  else  {  const char \*help\_text = "PSP exercise B2: Count class and total LOC.\n Usage:\n\tpsp\_B2\n\n ";  cout << help\_text;  }  return Result;  }  int  main (int arg\_count, const char \*\*arg\_vector)  {  //get the input stream, or print the help text as appropriate  istream \*input\_stream = input\_stream\_from\_args (arg\_count, arg\_vector);  if (input\_stream != NULL)  {  class\_based\_loc\_counter counter;  try  {  counter.set\_input\_stream (input\_stream);  counter.parse\_until\_eof ();  //output the counted lines  //counter.write\_countable\_lines( cout );  //output the loc  cout << "Total LOC: " << counter.loc\_count () << "\n";  cout << "classname:feature count:LOC by class:\n";  for (std::map < std::string, class\_metric >::const\_iterator iter =  counter.class\_map ().begin ();  iter != counter.class\_map ().end (); ++iter)  {  cout << iter->first << ":"  << iter->second.feature\_count << ":"  << iter->second.loc << "\n";  }  }  catch (exception & e)  {  cout << "Aborted with exception: " << e.what () << "\n";  cout << "last line parsed: \n" << counter.last\_line () << "\n";  }  }  }  /\*  \*/ |

**Test**

144 minutos para el código C ++, casi totalmente gastada en caprichos del lenguaje C ++ (discernir un nombre de clase de una característica implementada fuera de la declaración de clase, mientras que los argumentos o el tipo de retorno contienen espacios de nombres, tratando con declaraciones de clase para listar código, etc.). Todavía hay algunos problemas menores, pero, francamente, estoy dispuesto a mantener el código, ya que los problemas generan solo una diferencia de 1-2 LOC (aplicada al LOC total pero no a la clase LOC). El programa Eiffel tardó un poco más en probarse de lo que había imaginado gracias a las construcciones de bloques asimétricos de Eiffel (a lo que me refiero es a que "fin" termina un bloque, pero muchas, muchas cosas pueden comenzar uno, a diferencia del simétrico de C ++ " {"y"} ". También me preocupa que mi grabación defectuosa para el programa Eiffel estuviera incompleta ya que me distraía aprendiendo a hacer las cosas y dejaba que mi atención se desvaneciera de la PSP.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Program Number | [Class] Name | Number of Methods | [Class] LOC | Total Program LOC |
| 1A | number\_list | 7 | 54 |  |
|  |  |  |  | 82 |
| B1 | simple\_input\_parser | 12 | 52 |  |
|  | simple\_loc\_counter | 26 | 138 |  |
|  |  |  |  | 212 |
| B2 | class\_based\_loc\_counter | 24 | 200 |  |
|  | class\_metric | 3 | 7 |  |
|  | simple\_input\_parser | 12 | 52 |  |
|  | simple\_loc\_counter | 32 | 177 |  |
|  |  |  |  | 463 |

### PSP0.1 Project Plan Summary

**Table 3-4. Project Plan Summary**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Roman Huerta Bernabe | Date: | 000106 |
| Program: | Class/Feature/LOC counter | Program# | B2 |
| Instructor: | Wells | Language: | C++ |

|  |  |  |  |
| --- | --- | --- | --- |
| Program Size | Plan | Actual | To date |
| Base |  | 212 |  |
| Deleted |  | 1 |  |
| Modified |  | 11 |  |
| Added |  | 252 |  |
| Reused |  | 0 | 0 |
| Total New and Changed | 64 | 263 | 557 |
| Total LOC |  | 463 | 730 |
| Total new/reused |  |  |  |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Time in Phase (min): | Plan | Actual | To Date | To Date% |
| Planning | 8 | 13 | 37 | 6 |
| Design | 15 | 25 | 69 | 11 |
| Code | 32 | 86 | 185 | 29 |
| Compile | 10 | 12 | 43 | 7 |
| Test | 36 | 144 | 254 | 39 |
| Postmortem | 6 | 30 | 50 | 8 |
| Total | 107 | 310 | 638 | 100 |
| *Defects Injected* |  | Actual | To Date | To Date % |
| Plan |  | 0 | 0 | 0 |
| Design |  | 4 | 15 | 23 |
| Code |  | 19 | 46 | 70 |
| Compile |  | 0 | 2 | 3 |
| Test |  | 1 | 3 | 5 |
| Total development |  | 24 | 67 | 100 |
| *Defects Removed* |  | Actual | To Date | To Date % |
| Planning |  | 0 | 0 | 0 |
| Design |  | 0 | 0 | 0 |
| Code |  | 4 | 15 | 22 |
| Compile |  | 8 | 26 | 39 |
| Test |  | 12 | 26 | 39 |
| Total development |  | 24 | 67 | 100 |
| After Development |  | 0 | 0 |  |

|  |
| --- |
| Eiffel code/compile/test |

|  |  |  |  |
| --- | --- | --- | --- |
| Time in Phase (min) | Actual | To Date | To Date % |
| Code | 41 | 87 | 40 |
| Compile | 28 | 56 | 26 |
| Test | 61 | 74 | 34 |
| Total | 130 | 217 | 100 |
| Defects Injected | Actual | To Date | To Date % |
| Design | 1 | 4 | 10 |
| Code | 18 | 38 | 90 |
| Compile | 0 | 0 | 0 |
| Test | 0 | 0 | 0 |
| Total | 19 | 42 | 100 |
| Defects Removed | Actual | To Date | To Date % |
| Code | 0 | 1 | 2 |
| Compile | 10 | 24 | 57 |
| Test | 9 | 17 | 41 |
| Total | 19 | 42 | 100 |

### Time Recording Log

**Table 3-5. Time Recording Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Roman Huerta Bernabe | Date: | 170621 |
|  |  | Program: | B2 |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Start | Stop | Interruption Time | Delta time | Phase | Comments |
| 170621 13:12:10 | 170621 13:38:06 | 12 | 13 | plan |  |
| 170621 13:39:55 | 170621 14:05:45 | 0 | 25 | design |  |
| 170621 14:13:06 | 170621 15:39:21 | 0 | 86 | code |  |
| 170621 15:50:37 | 170621 16:03:28 | 0 | 12 | compile |  |
| 170621 16:03:33 | 170621 18:29:20 | 1 | 144 | test |  |
| 170621 19:15:59 | 170621 19:46:29 | 0 | 30 | postmortem |  |
|  |  |  |  |  |  |

### Defect Reporting Logs

**Table 3-7. Defect Recording Log**

|  |  |  |  |
| --- | --- | --- | --- |
| Student: | Roman Huerta Bernabe | Date: | 170621 |
|  |  | Program: | B2 |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Defect found | Type | Reason | Phase Injected | Phase Removed | Fix time | Comments |
| 170621 14:40:22 | md | om | design | code | 9 | forgot to add a feature to extract the class name from the class description |
| 170621 14:55:13 | md | om | design | code | 7 | forgot to add a feature to extract the class name from an external feature |
| 170621 15:21:33 | ic | ig | design | code | 3 | Added some refactorings to find scope operators, etc. |
| 170621 15:29:26 | ic | ig | design | code | 2 | Added "has\_entry\_for\_class\_name" feature |
| 170621 15:52:18 | ch | om | code | compile | 0 | Forgot to include required .h file for ENSURE macro |
| 170621 15:55:19 | sy | cm | code | compile | 0 | accidentally included "static" in member implementation for data members |
| 170621 15:56:20 | wn | cm | code | compile | 0 | Used "adjust\_class\_method\_count" instead of "adjust\_class\_feature\_count" |
| 170621 15:57:25 | wn | cm | code | compile | 0 | Wrong name: m\_class\_name instead of m\_current\_class\_name |
| 170621 15:58:26 | wn | cm | code | compile | 0 | Wrong name: used "method" in name instead of "feature" |
| 170621 15:59:18 | wn | cm | code | compile | 0 | Requisito de espacio de nombres perdido, es decir, std :: string :: size\_type en lugar de solo size\_type |
| 170621 16:00:29 | ic | om | code | compile | 0 | olvidé declarar una característica como constante |
| 170621 16:01:56 | ic | om | code | compile | 1 | olvidé agregar la implementación de la función de acceso class\_map |
| 170621 16:05:06 | me | ex | code | test | 20 | Se está procesando una excepción de contrato errónea: nada que ver con el programa B2 |
| 170621 16:26:07 | md | ig | code | test | 2 | Se agregaron más talones de diagnóstico |
| 170621 16:31:15 | mi | om | code | test | 0 | Olvidé reiniciar el contador durante la construcción. Tonto, eso. |
| 170621 16:32:21 | wn | ig | code | test | 0 | Estaba buscando "clase" al principio de las declaraciones de clase en lugar de "clase". |
| 170621 16:34:14 | wn | cm | code | test | 1 | Se usó la constante de manifiesto "clase" en lugar de la constante de clase para el comienzo de la clase. |
| 170621 16:36:26 | we | kn | code | test | 13 | La lógica de eliminar los espacios en blanco era una locura, ignorando las líneas de un carácter |
| 170621 16:54:59 | wa | kn | code | test | 4 | Debido a la sincronización incorrecta del cálculo del nivel de anidación de comentarios de bloque, los comentarios de final de bloque eran "contables" |
| 170621 17:00:01 | wa | kn | code | test | 3 | Se agregó una verificación para asegurarse de que la línea anterior tuviera algo útil antes de agregar una línea para la declaración de características externas |
| 170621 17:04:34 | wa | kn | test | test | 38 | Muchos problemas con el manejo de espacios de nombres en el recuento de códigos. Desafortunadamente, se corrigieron algunos errores sin iniciar sesión aquí. |
| 170621 17:46:36 | wa | kn | code | test | 8 | Error al manejar la extracción del nombre de la declaración de clase cuando se usa la herencia. |
| 170621 18:00:05 | ma | om | code | test | 20 | Estaba tratando las marcas "/ \*" entre comillas como comentarios anidados cuando eran datos de cadena. Solución económica: reformateó los datos de la cadena. |
| 170621 18:22:59 | wa | om | code | test | 2 | grr ... buscando el nombre de clase en una definición de función externa de C ++ mal etiquetado nombre de clase cuando está precedido por &. Modificará el estilo para que funcione. |
|  |  |  |  |  |  |  |