Documentación y análisis de Windows XP

Índice

1. Métricas y su clasificación

Para este análisis se han tenido en cuenta distintas métricas que se pueden encontrar en la herramienta *Understand* y en *Analizo*. Se pueden clasificar de distintas maneras y son las siguientes:

- Métricas de líneas de código:
  - *Lines of Comments*, **LCOMM**: cuenta el número de líneas que son comentarios de un programa.
  - *Lines of Code*, **LOC**: cuenta el número de líneas de código fuente de un programa. Así se determina el tamaño de un programa. Esta métrica se puede utilizar para estimar la carga de trabajo para el desarrollo del programa, la productividad de la programación y el mantenimiento del software publicado.
  - *Blank Lines of Code*, **BLOC**: número de líneas en blanco.
  - *Comment Lines of Code*, **CLOC**: número de líneas que contienen un comentario.
  - *Maximum Method Lines of Code*, **MMLOC**: número máximo de líneas de código de un método.
  - *Average Method Lines of Code*, **AMLOC**: indica si el código está bien distribuido entre los métodos. Como de grandes son los métodos.
    - Es preferible tener operaciones pequeñas y comprensibles.
- Métricas de Complejidad:
  - *Cyclomatic Complexity*, **V(G)**: La complejidad ciclomática indica la complejidad de un programa. Es una medida cuantitativa del número de caminos linealmente independientes a través del código fuente de un programa. También puede aplicarse a funciones, módulos, métodos o clases individuales dentro de un programa.
  - *Weighted Method per Class*, **WMC**: Indica la complejidad de una clase.
  - *Essential complexity*: se refiere a la cantidad de caminos lógicos independientes en una función o método. Mide cuántos caminos diferentes de ejecución pueden existir en una función o método debido a las diferentes ramas condicionales y bucles.
  - *Average Cyclomatic Complexity per Method*, **ACCM**: media de complejidad de los métodos.
- Métricas de Acoplamiento:
  - *Coupling Between Objects*, **CBO:** Un objeto está acoplado a otro si ambos actúan el uno sobre el otro. Si una clase utiliza los métodos de otras clases, entonces ambas están acopladas.
    - Un aumento de **CBO** indica un aumento de las responsabilidades de una clase. El valor de **CBO** para las clases debe mantenerse lo más bajo posible.
  - *Count Class Base*, **IFANIM**: Número de las clases base inmediatas.
  - *Number of Children*, **NOC**: Indica cuantas subclases van a heredar los métodos de la clase padre.
    - El valor de NOC indica el nivel de reutilidad en una aplicación.
- Métricas de Clases:
  - *Lack of Cohesion in Methods*, **LCOM**: Se puede usar para medir el grado de cohesión. Refleja lo bien que un sistema está diseñado y cómo de compleja es una clase.
  - *Lack of Cohesion in Operations*, **LCOO**: Mide la cantidad de pares de métodos que no acceden a la misma instancia de variables. Mide la carencia de cohesión en componentes.

        - Un valor de LCOO elevado indica disparidad en la funcionalidad proporcionada por el aspecto.
    - *Count Decl Class Variable*, **NV**: Número de las variables de clase.
    - *Number of Instance Methods*, **NIM**: Métodos definidos en una clase que solo son accesibles a través de un objeto de una clase.
    - *Number of Instance Variables*, **NIV**: variables definidas en una clase que solo son accesibles a través de un objeto de esa clase.
    - *Number of Private Methods*, **NPRM**: Número de métodos privados locales no heredados.
    - *Number of Public Methods*, **PM** o **NPM**: Número de métodos públicos locales no heredados.
    - *Average Number of Parameters*, **ANPM**: Calcula la media de parámetros de los métodos de una clase.
        - Un número alto de parámetros puede indicar que el método tiene más de una responsabilidad, es decir, más de una función.
    - *Afferent Connections per Class*, **ACC**: Mide la conectividad de una clase.
    - *Number of Atributtes*, **NOA**: Calcula el número de atributos de una clase.
        - Una clase con muchos atributos puede indicar que tiene muchas responsabilidades y presenta baja cohesión.
    - *Number of Public Attributes*, **NPA**: Mide la encapsulación. Las buenas prácticas de programación recomiendan que los atributos de una clase deben de ser privados.
        - El número ideal de esta métrica es 0.
    - *Response for a Class*, **RFC**: Número de los distintos métodos y constructores invocados por una clase.
    - *Number of Methods*, **NOM:** Calcula el número de métodos por clase, para medir el tamaño de las clases.
- Métricas de ficheros:
    - *CountDeclFile*: número total de ficheros.
    - *CountDeclFileCode*: número total de ficheros de código.
    - *CountDeclFileHeader:* número total de ficheros de encabezado.
- Métricas de Herencia:
    - *Depth of Inheritance Tree*, **DIT**: Mide el camino máximo/más largo desde un nodo hasta la raíz del árbol. Indica cómo de lejos se ha declarado una clase en la jerarquía de herencia.
    - *Count Class Base*, **IFANIM**: Número de las clases base inmediatas.
    - *Number of Children*, **NOC**: Indica cuantas subclases van a heredar los métodos de la clase padre.
        - El valor de NOC indica el nivel de reutilidad en una aplicación.
- *Number of Refined Constants*, **NCR**: Es el número de constantes que se han ajustado para adaptarse a contextos específicos.
    - Su valor sugiere lo complejo que puede ser el mantenimiento.
- *Number of Constants*,**NOCT**: Mide el número de constantes (clases, interfaces) usados para realizar una característica.
- *Number of Features*, **NOF**: Indica el número de características de un sistema.

2. Understand, Analizo y SCALe

2.1. Understand y Analizo

Understand se puede descargar desde la página principal, es una herramienta que cuenta con una interfaz gráfica y además se puede utilizar mediante línea de comandos.
En este caso se ha instalado en un Windows 11 con la versión *Build 1144*, y se puede usar o la aplicación o mediante **cmd** con el comando *und*, si ejecutamos solamente el comando entramos en la consola interactiva o se pueden añadir otros argumentos.
Para crear el proyecto del código de Windows XP y analizarlo se ha hecho lo siguiente, teniendo como referencia *Understand Command Line*:

```
und -db XPSP1.und create                //esto crea una base de datos vacia llamada XPSP1
und add XPSP1 XPSP1.und                 //añadimos todos los ficheros de la carpeta XPSP1
und settings -metrics all XPSP1.und     //especificamos que queremos todas las métricas
und analyze XPSP1.und                   //lo analizamos
und report XPSP1.und
und metrics XPSP1.und
```

Una vez creado el proyecto, como el comando *und metrics XPSP1.und* generaba un fichero csv de métricas vacío se ha hecho uso de la API de Python con la que cuenta la herramienta para generar todas las métricas disponibles. Para ello primero hemos realizado estos pasos para poder utilizar la API. Y se ha realizado el siguiente script, que podéis encontrar aquí.

Analizo se puede instalar de varias maneras, en este caso al tener *Windows Subsystem for Linux* hemos descargado Debian, y para instalar la herramienta se ha ejecutado *apt install analizo* con la versión *1.25.4*. Con esta herramienta se puede obtener el grafo de dependencias y la evolución del código fuente, además de obtener métricas mediante el comando *analizo metrics XPSP1*.

2.2. SCALe

SCALe web app es una aplicación que permite combinar resultados de análisis estáticos de múltiples herramientas en una interfaz, además de que proporciona asignaciones para diagnósticos desde las herramientas a los estándares de codificación segura SEI CERT.
Para instalarlo se han seguido los manuales que se pueden obtener al descargar la aplicación en la ruta *SCALe-main/scale.app/public/doc/SCALe2/215846575.html*, aun así se va a detallar los pasos seguidos para su correcta instalación.

- Primero se ha obtenido la imagen iso del sistema operativo Ubuntu 14.04 y se ha creado una máquina virtual en *VirtualBox*.
- Una vez con la máquina creada y la carpeta de la aplicación descargada de GitHub y extraída, se procede a la instalación de *SCALe*:

1. Primero se instalarán los siguientes paquetes y sus versiones con *sudo apt-get install <package_name>=<version_string>* .

build-essential	11.6ubuntu6
sqlite3	3.8.2-1ubuntu2.2
sqlite3-pcre	0~git20070120091816+4229ecc-0ubuntu1
rubygems-integration	1.5
python	2.7.5-5ubuntu3
libncurses5-dev	5.9+20140118-1ubuntu1
libssl-dev	1.0.1f-1ubuntu2.27
ruby-dev	1:2.3.0+1
libsqlite3-dev	3.8.2-1ubuntu2.2

2. Luego se instalará el gestor de versiones de ruby, rvm, con estos pasos que son los siguientes:

    sudo apt-get install software-properties-common
    sudo apt-add-repository -y ppa:rael-gc/rvm
    sudo apt-get update
    sudo apt-get install rvm
    sudo usermod -a -G rvm $USER
    echo 'source "/etc/profile.d/rvm.sh"' >> ~/.bashrc

3. Ahora se instalará GNU Global en el directorio *scale.app*:

    cd $SCALE_HOME/scale.app
    wget http://tamacom.com/global/global-6.5.1.tar.gz
    tar -xzf global-6.5.1.tar.gz
    cd global-6.5.1
    ./configure
    make
    sudo make install

4. Después se instalarán dos gemas con los siguientes comandos:

    sudo gem install json_pure -v 1.8.3
    sudo gem install bundler -v 1.8.3

5. Al lanzar ahora la aplicación nos saldrá el siguiente error *execJs: 'Could not find a JavaScript runtime'*, para solucionarlo se ha instalado *Node.js* con el siguiente comando *sudo apt-get install nodejs* .

6. Por último se instalan las dependencias:
   cd $SCALE_HOME/scale.app
   bundle install --path vendor/bundle/
   bundle exec rake db:migrate

7. Y se lanza el servidor:
   cd $SCALE_HOME/scale.app
   bundle exec thin start --port 8080

- Tras esto, en un navegador en la dirección *http://localhost:8080* estará *SCALe* funcionando. Para usarlo solo hay que crear un proyecto, subir el código fuente comprimido del código que has analizado y los ficheros csv con las métricas.

3. Análisis de las métricas obtenidas

En general la media de la Complejidad ciclomática de las funciones/métodos es baja, está en un 4, lo que implica que en general no son complejas al igual que la media de la complejidad esencial, sin embargo si observamos los máximos cuyos valores son 1.591 y 514 respectivamente, se puede observar que hay funciones/ficheros de extremada complejidad.

También tanto por el número de clases (196.249), métodos (49.188) y variables (60.883), además del número de ficheros (120.811) y el número de líneas (66513977) es un proyecto bastante grande.

El número de archivos de código (70.678) y el número de archivos de encabezado (50.133) sugiere una estructura de interfaz clara y separada del código de implementación, lo que es una buena práctica en el desarrollo de software para mejorar la legibilidad y la modularidad.

Por las métricas respectivas a las variables podemos observar una división entre las variables de instancia dependiendo de su visibilidad (privado, protegido, público) al igual que los métodos, lo que implica que se está utilizando la encapsulación en el código, lo que puede ayudar a controlar la exposición de datos y garantizar la integridad de los objetos.

Luego si tenemos en cuenta la distribución de líneas en blanco (10.187.612) y de código (35.176.292) hay cierto equilibrio entre los espacios y la lógica implementada lo que es relevante para evaluar la organización y la legibilidad del código.

El número de sentencias (20.525.331) implica que el código es bastante extenso en términos de funcionalidad implementada, de estas sentencias hay un gran número (6.431.269) que son declaraciones por lo que hay una cantidad considerable de definiciones y estructuras de alto nivel.

En cuanto a las sentencias de ejecución (13.513.110) y las sentencias vacías (905.954) se puede deducir que se hacen una gran cantidad de operaciones, sin embargo habría que revisar las sentencias vacías.

Además aproximadamente el 45% del código contiene comentarios lo que puede significar que el código está documentado, algo importante para el mantenimiento, comprensión y legibilidad del código.

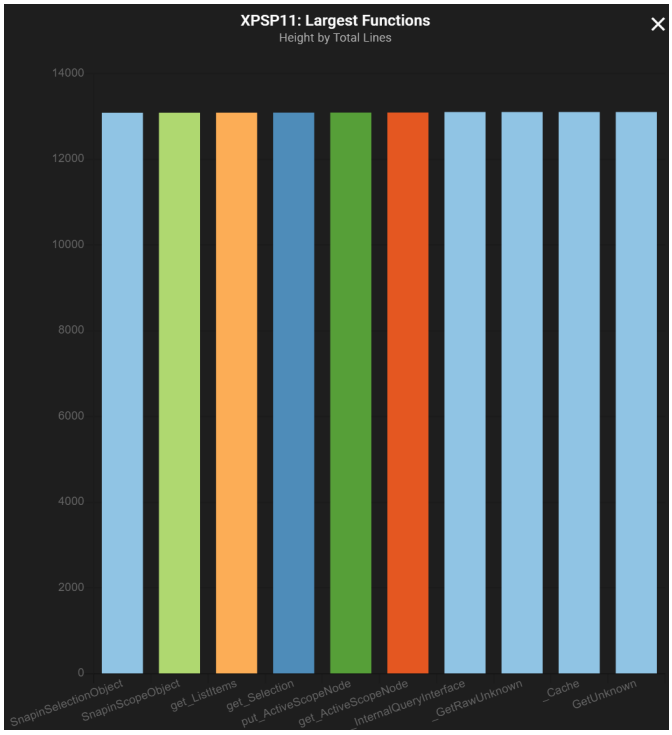
Se pueden encontrar las métricas obtenidas aquí.

3.1. Project Overview

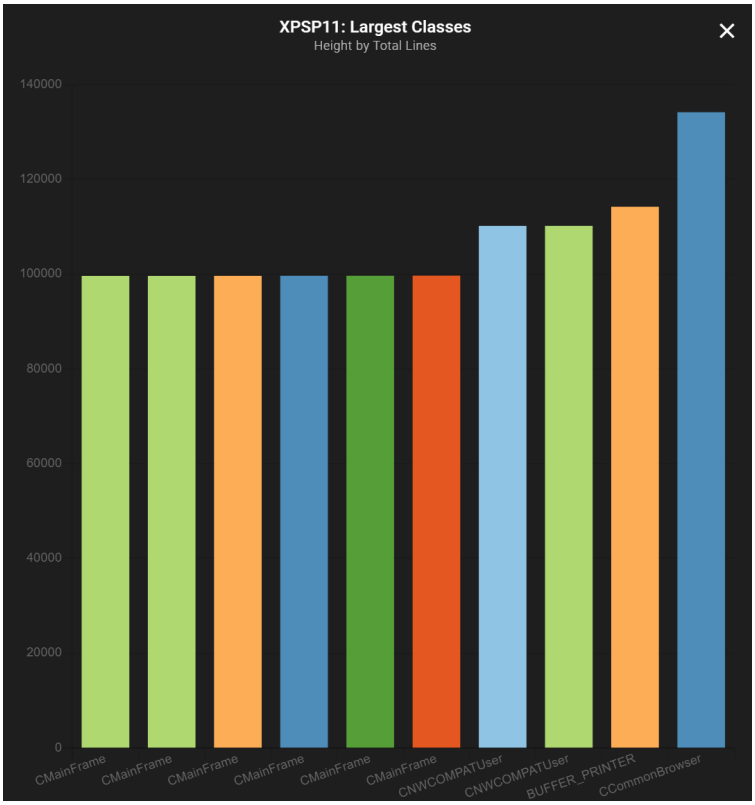
Aquí se presentan algunas de las funciones más complejas y de mayor tamaño además de las clases y los ficheros.

3,256	72%	1,243,535	8,219
Missing Includes	Parse Accuracy	Errors	Warnings

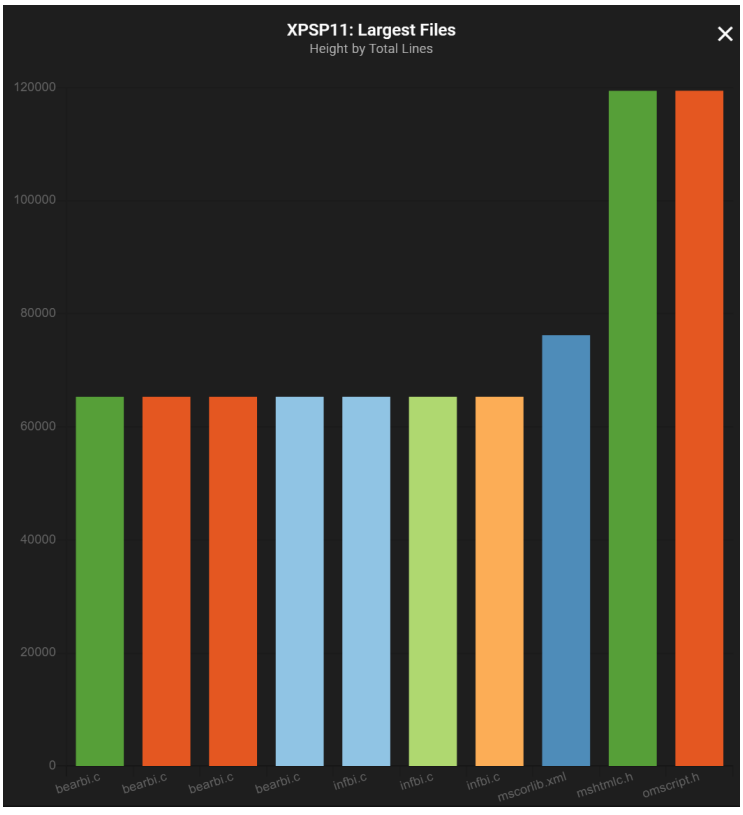
68,750,846	128,979	196,407	1,070,906	130
Lines	Files	Classes	Functions	Subprograms

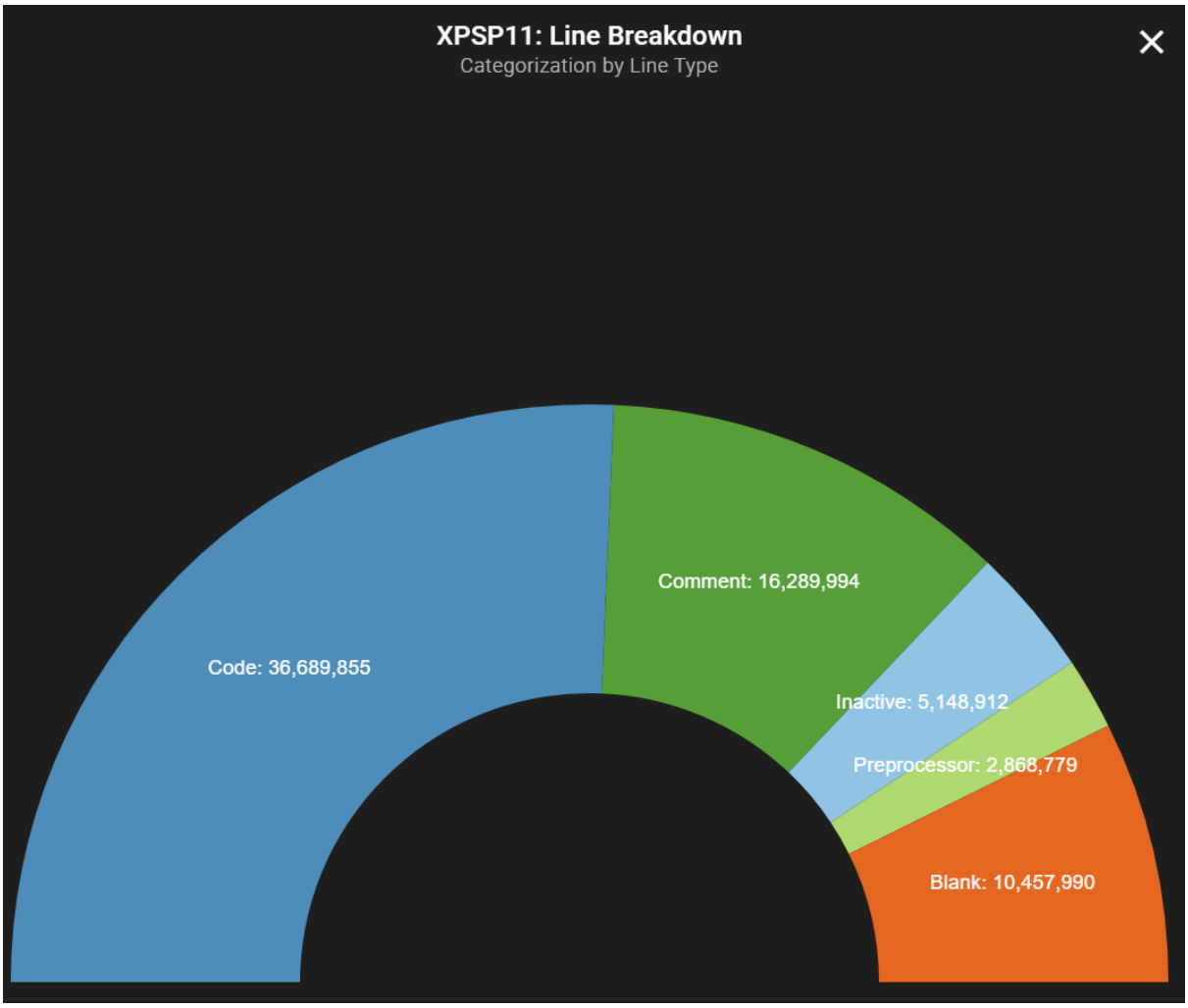



XPSP11: Most Complex Classes
Complexity by the Average McCabe Cyclomatic Metric
97
CMyWorkerArray
55
55
84
MSSQLParser
55
52
53
55
55
55
45
52
55
54
RDPShader
52
52
67
CQualifierRowFetchObj
79
CDllRedir
55
55
92
CMyWorkerScalar


XPSP11: Largest Classes
Height by Total Lines
140000
120000
100000
80000
60000
40000
20000
0
CMainFrame
CMainFrame
CMainFrame
CMainFrame
CMainFrame
CMainFrame
CNWCOMPATUser
CNWCOMPATUser
BUFFER_PRINTER
CCommonBrowser


XPSP11: Most Complex Files
Complexity by the Average McCabe Cyclomatic Metric
1127
atlbase.h
784
dbgtags.c
723
atlcom.h
1005
atlctris.h
653
atlcom.h
616
atlbase.h
643
atlbase.h
817
arity.h
694
atlcom.h
557
dbcmdtre.hxx
571
atldbcli.h
686
atlbase.h
872
atldbcli.h
653
atlcom.h
615
atlbase.h
617
dx8sddifw.h
794
pxrxpoly.c
778
atlsecurity.h
697
stdprop.h
1083
atlctris.h


XPSP11: Largest Files
Height by Total Lines
120000
100000
80000
60000
40000
20000
0
bearbi.c
bearbi.c
bearbi.c
bearbi.c
infbi.c
infbi.c
infbi.c
mscorlib.xml
mshtmlc.h
omscript.h


XPSP11: Line Breakdown
Categorization by Line Type
Code: 36,689,855
Comment: 16,289,994
Inactive: 5,148,912
Preprocessor: 2,868,779
Blank: 10,457,990