

*Software Measurement. SOEN 6611 2014/4 D*

*Dr. Peter Rigby*

**Assignment 3**

**Understand API**

|  |  |
| --- | --- |
| **Team members information (in no particular order)** | |
| **Name** | **SID** |
| Ali Sangari | 6816304 |
| Hojabr Sattari | 6435807 |
| Bhaskar Baddam | 6761038 |

# Understand API

Understand API is very efficient at collecting metrics about the code it analyzes. These metrics can be extracted automatically via command line calls, exported to spreadsheets, viewed graphically, dynamically explored in the GUI. It is an easy way of analyzing the code with the diagram and navigational features for definition and declaration. They can also be reported at the project level, for files, classes, functions and methods.

We calculated LCOM and CBO metrics from Understand that have evolved over many years to accommodate common needs of software engineering/development projects. Understand is a static analysis tool for maintaining, analyzing and measuring a large code bases.

Understand provides APIs for four programming languages,

* C/C++
* Perl
* Python
* Java

C/C++ and Perl APIs have extensive support and were developed before the other two. Python has good support and there are example codes to help developers get started. The library for Java was developed at the same time as the team started to develop their eclipse plugin. Java API is very lean and provides only the most necessary metric extraction and calculations.

The following is a list of metrics that are available for C/C++ source files in OOP paradigm,

* **Base Classes:** Number of immediate base classes. [aka IFANIN]
* **Coupling Between Objects:** Number of other classes coupled to. [aka CBO (coupling between object classes)]
* **Number of Children:** Number of immediate subclasses. [aka NOC (number of children)]
* **Classes:** Number of classes.
* **Class Methods:** Number of class methods.
* **Class Variables:** Number of class variables.
* **Function:** Number of functions.
* **Instance Methods:** Number of instance methods. [aka NIM]
* **Instance Variables:** Number of instance variables. [aka NIV]
* **Private Instance Variables:** Number of private instance variables.
* **Protected Instance Variables:** Number of protected instance variables.
* **Public Instance Variables:** Number of public instance variables.
* **Local Methods:** Number of local methods.
* **Methods:** Number of methods, including inherited ones. [aka RFC (response for class)]
* **Local Const Methods:** Number of local const methods.
* **Friend Methods:** Number of local friend methods. [aka NFM]
* **Private methods**: Private Methods: Number of local private methods. [aka NPM]
* **Protected Methods:** Number of local protected methods.
* **Public Methods:** Number of local public methods. [aka NPRM]
* **Outputs:** Number of called subprograms plus global variables set. [aka FANOUT]
* **Depth of Inheritance Tree:** Maximum depth of class in inheritance tree. [aka DIT]
* **Lack of Cohesion in Methods:** 100% minus the average cohesion for package entities. [aka LCOM, LOCM]

For a full list of metrics please visit: <https://scitools.com/support/metrics_list/>

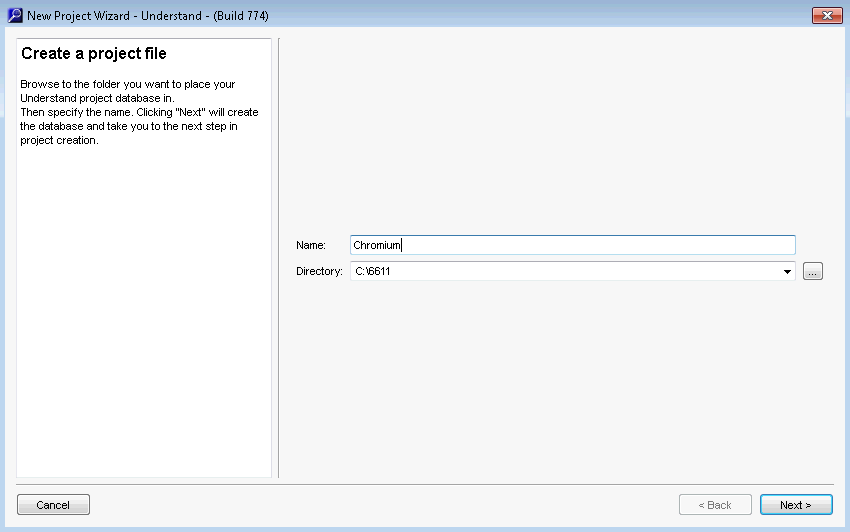
Appendix

# Step-by-step description of how we used Understand for this assignment

Understand can be used to perform analysis on source code, generate diagrams that visualize metrics, export metrics and reports for further analysis of the project.

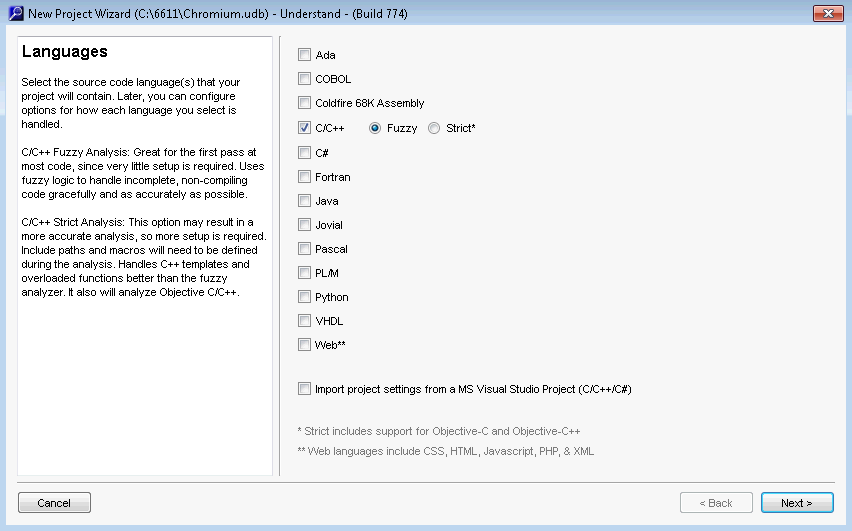
## 1. Creating a project in Understand, exporting metrics and generating reports

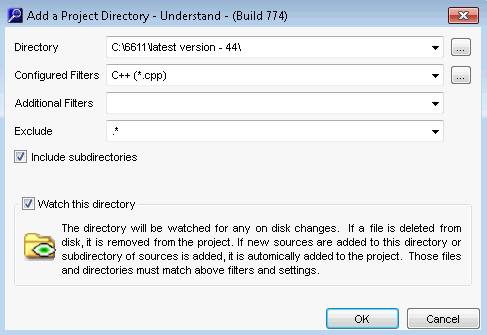
First, we created a new project in Understand to import and analyze each version of Chromium project.



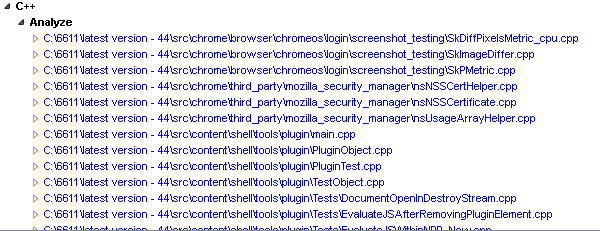
During project creation, we selected the file types that were going to be analyzed using

Understand. Based the assignment description, CPP files were selected.

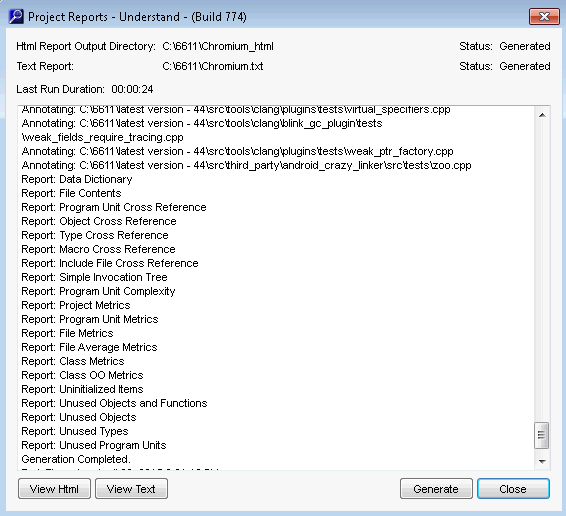




After completing the wizard for project creation, Understand begins to analyze the source code.

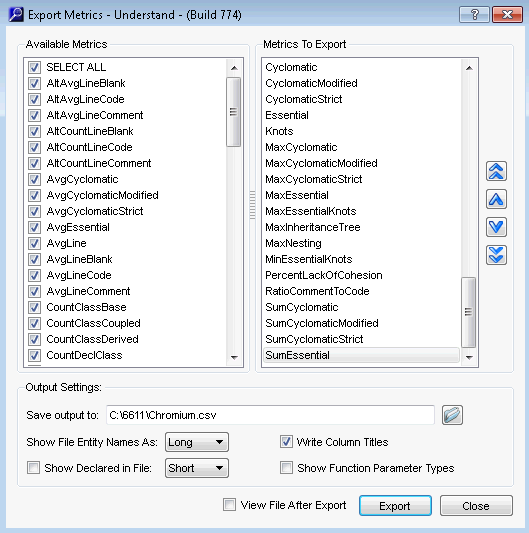


Understand generates reports in HTML format for easy navigation.



To view generated reports from all versions of Chromium used in this assignment, please navigate to “*Reports generated by Understand*” folder relative to this file’s location.

Generating metrics is another way of exporting information about the source code that could help in analysis process.



To view exported metrics from all versions of Chromium used in this assignment, please navigate to “*Metrics generated by Understand*” folder relative to this file’s location.

## 2. Using the generated project file and Python API

After creating a project and running the initial code analysis, a .UDB gets created on disk (project location). This file can then be fed into our Python script for metric extraction and analysis.

UDB files for all versions of Chromium used in this assignment, please navigate to “*UnderstandProjects*” folder relative to this file’s location.

We have analyzed versions [25-34], and the latest version (version 44) of Chromium for this assignment.

In assignment it is mentioned that we should calculate LCOM and CBO “for each cpp class file in the system”, and that an output similar to the following should be produced,

|  |  |  |  |
| --- | --- | --- | --- |
| release | class\_path | cbo | lcom |
| 30 | /home/pcr/MyClass.cpp | 3 | 1 |

One if the issues that we faced during this assignment was that CPP class files, unlike java class files that have one dominant class that also has the same name as its file, have multiple classes inside them and the classes do not necessarily have the same name as their file.

Because of the above, we wrote two slightly different scripts to generate two sets of output,

1. A script that extracts only classes that have a file with the same name. this way we could also have *class\_path* as one of the columns. But this approach blocked us from providing analytical information on many classes.
2. In the second approach, we export all CPP classes in the project along with their analytical information. In this approach we replaced *class\_path* with *class name (class type)*.

To start coding, we made sure the same architecture edition of Python and Understand is installed (32-bits / 64-bits).

Next we made sure that Understand has an entry in PATH environment variable to provide system access to the Python API.

After that it was pretty straight forward,

*// To have access to the API from Python script***import understand**

// To load a UDB file into memory  
**db = understand.open(dbfile)**

// To access info on .CPP files  
**for file in (db.lookup(".cpp","File")):**

// To make sure only classes are being picked up  
**if((ent.kindname() == "Class") or (ent.kindname() == "Public Class") or (ent.kindname() == "Unknown Class") or (ent.kindname() == "Private Class")):**

// To get CBO/LCOM for a class. -1 is used as error code.  
**metric = file.metric(("CountClassCoupled",))**

**if metric["CountClassCoupled"] is not None:**

**return metric["CountClassCoupled"]**

**if metric["CountClassCoupled"] is None:**

**return "-1"**