Dalhousie University

Topics in Program Comprehension

CSCI 6306

Architecture Extraction

Group : Something
Bhupendra Rajawat
Bryan Thomas D'silva
Saurabh Singh

June 13, 2017

DALHOUSIE UNIVERSITY

Note: Our fourth group member, Anto Anthony Arulappan had an emergency in the family and had to leave to India for the same. He has dropped the course and hence will no longer be a part of our group.

Contents

Abstract

Like we already discussed in different presentation and even in class that to figure out the architecture of any application program comprehension is required.To do program comprehension either code must be properly written to give over all understanding or application documentation need to be done well to make other people understanding code a edge to grasp the logic easily. We went across different application which we have stated in next section and found Gimp to be most suitable in terms of code and documentation. Fulfilling the main aim of this assignment to get the better insight how to figure out high -level architecture we did used a mix approach of top-down and bottom-up approach along with taking help from well written documentation of Gimp present at http://wiki.gimp.org .

1 Finding a Software System

We looked at the following software systems and games.

1. **Kodi**: Kodi is an open source software media player and entertainment hub. This is available on atleast seven platforms. This was one of the first softwares we looked at and kept it as a contender

2. **Quake III**: The source code for Quake is challenging and interesting. However, there is almost no documentation for the same and hence we decided not to go ahead with this for now.

3. **GIMP**: GIMP is a cross platform image editor with an excellent community backing it up. We decided to extract the architecture for GIMP. We talk more about GIMP in the further sections.

We also looked at Doom3, SuperTuxKart, Wordpress, TuxRunner and Linux but found GIMP to be the most interesting one to use for this assignment

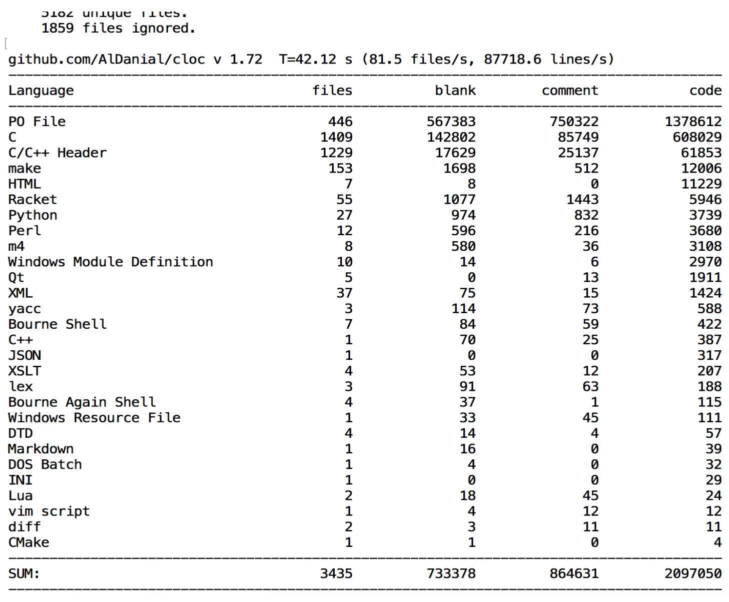
2 About GIMP

GIMP was designed initially as a General image manipulation program. This was later changed to GNU Image Manipulation Program. It is free and openly available to retouch and edit images, convert image formats and more specialized tasks targeted towards image manipulation. GIMP provides an interface to handle simple graphics needs without learning advanced methods for image manipulation. GIMP is more user friendly as compared to other photoshop techniques and can be modified to fit your needs. And the best part about it is that it is free.

3 Tools

3.1 Count Lines of Code

Count Lines of Code[2], abbreviated as CLOC is a software used to count comment lines, blank lines and physical lines of code. We used this software initially when we looked at what software system we need to pick. We can look at figure 1 for the output for GIMP. To sum it we have 3,695,059 LOC. However, we would assume that the chunk of our work would be for C which sums to about 608,029 LOC.

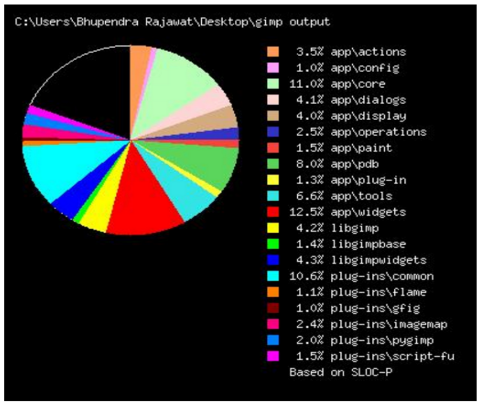


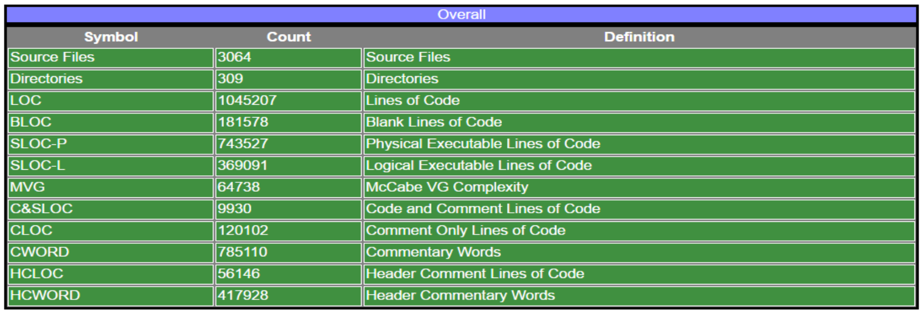
```
    5182 unique files.
    1859 files ignored.

github.com/AlDanial/cloc v 1.72  T=42.12 s (81.5 files/s, 87718.6 lines/s)
-------------------------------------------------------------------------------------
Language                             files          blank        comment           code
-------------------------------------------------------------------------------------
PO File                                446         567383         750322        1378612
C                                     1409         142802          85749         608029
C/C++ Header                          1229          17629          25137          61853
make                                   153           1698            512          12006
HTML                                     7              8              0          11229
Racket                                  55           1077           1443           5946
Python                                  27            974            832           3739
Perl                                    12            596            216           3680
m4                                       8            580             36           3108
Windows Module Definition               10             14              6           2970
Qt                                       5              0             13           1911
XML                                     37             75             15           1424
yacc                                     3            114             73            588
Bourne Shell                             7             84             59            422
C++                                      1             70             25            387
JSON                                     1              0              0            317
XSLT                                     4             53             12            207
lex                                      3             91             63            188
Bourne Again Shell                       4             37              1            115
Windows Resource File                    1             33             45            111
DTD                                      4             14              4             57
Markdown                                 1             16              0             39
DOS Batch                                1              4              0             32
INI                                      1              0              0             29
Lua                                      2             18             45             24
vim script                               1              4             12             12
diff                                     2              3             11             11
CMake                                    1              1              0              4
-------------------------------------------------------------------------------------
SUM:                                  3435         733378         864631        2097050
-------------------------------------------------------------------------------------
```

Figure 1: clocgimp output for GIMP

3.2 LocMetrics

LocMetrics[1] is very similar to what CLOC does. We did find this software interesting because of how it visualized the output.LocMetrics counts total lines of code (LOC), blank lines of code (BLOC), comment lines of code, lines with both code and comments (C&SLOC), logical source lines of code (SLOC-L), McCabe VG complexity (MVG), and number of comment words (CWORDS). Physical executable source lines of code (SLOC-P) is calculated as the total lines of source code minus blank lines and comment lines. Counts are calculated on a per file basis and accumulated for the entire project. LocMetrics also generates a comment word histogram. [1]
LocMetric has a very unique way to represent data which can be visualize in the figure 2





Overall

Symbol	Count	Definition
Source Files	3064	Source Files
Directories	309	Directories
LOC	1045207	Lines of Code
BLOC	181578	Blank Lines of Code
SLOC-P	743527	Physical Executable Lines of Code
SLOC-L	369091	Logical Executable Lines of Code
MVG	64738	McCabe VG Complexity
C&SLOC	9930	Code and Comment Lines of Code
CLOC	120102	Comment Only Lines of Code
CWORD	785110	Commentary Words
HCLOC	56146	Header Comment Lines of Code
HCWORD	417928	Header Commentary Words

Figure 2: LocMetrics representaion of GIMP

3.3 Eclipse:

For the analysis part of this application IDE used is Eclipse as it provide features that are interesting for C/C++ developers like Some of the features that are interesting for C/C++ developers are: **cross-referencing:** Ctrl click an identifier and it will take you to its definition, call hierarchy, comprehensive search capabilities, built in Git support, Autotools integration, graphical debugger, explore remote systems from within Eclipse, focus on the current task, Bugzilla integration, static code analysis. and more. Eclipse runs on every platform that can run Java and has a graphical interface. Because of the above features Eclipse is good to use with the dependencies required for GIMP.

4 Directory Structure

We generated a butterfly diagram in figure 3 for the directory structure. Using this and the documentation we did want to identify the main folder we would want to have a look at. We concluded that the app directory is the main directory that we are concerned about. It contains the source code for GIMP without any external tools or plugins. We would ideally ignore the modules, plugins and libraries for now and focus on the app directory. So we did the same for the app folder in figure 4. It was easy to comprehend since GIMP is very well organized with a good community support and documentation. The app/core directory is the guts of the GIMP software. Looking at the documentation provided for GIMP, major changes that need to be made start here. With the goal of refactoring in mind, this is where we would start, keeping in mind that changing files here will have an avalanche effect, so we need to be prepared for big changes in other files depending on these.

5 Entry Point

While trying to build the application, we looked at MakeFile.am. The following line in MakeFile.am which gave us an idea about the entry point into the program.
gimp_@GIMP_APP_VERSION@_SOURCES = $(libapp_sources) main.c
Understand C++ has an option to generate a callby graph. Running this on main.c, instantly we can guess that main is not called by anything which

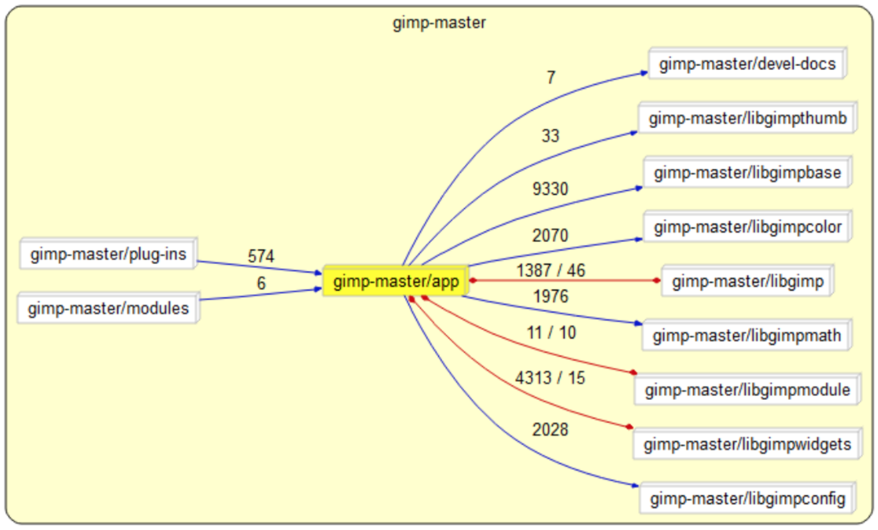


Figure 3: Directory Structure

further reinforces the idea that main is the entry point into the program. The call by graph for main.c is shown in figure 5

So looking into the main.c to further reinforce our idea.

The main.c has a main function which points to functions that talk about the installation directory. We did gain a lot of information from the Understand C++ software.

Failures: We tried to use Doxygen to generate call graphs for comparison and better understanding. However, the tool crashed on the first go. And on the second go generated quite a bunch of call graphs when run directly on the app directory so will leave this tool aside for now.

6 Build Process

Analysis of any application can not be completed without building it. Like every application provide documentation related to how one can build the application on any platform eg Linux, Mac etc. Building any application not only help to compile any application to create executable but end up to helping to figure out what are the entry point with in the code of application.

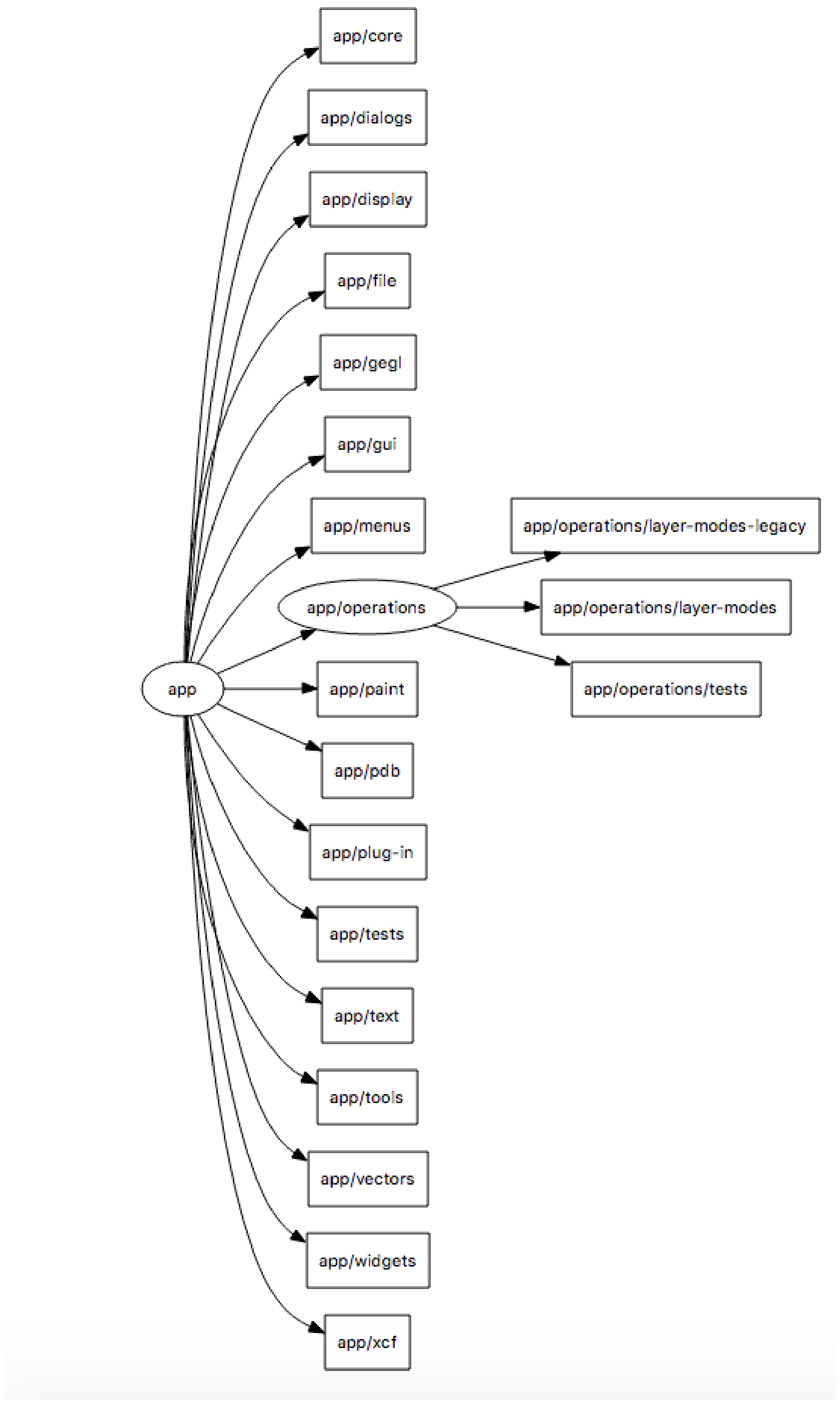


Figure 4: Directory Structure for app folder

Figure 5: Callby graph for main.c

Like in case of Gimp they have "MakeFile.am" which is a autogenerator file with in "master/app" directory to create "Makefile.in" . If you analyze Makefile.am you will get that "main.c" is the file from were compilation starts.

We tried to use Doxygen to generate call graphs for comparison and better understanding. However, the tool crashed on the first go. And on the second go generated quite a bunch of call graphs when run directly on the master/app directory so will leave this tool aside for now.

Talking about Dependencies required for the Gimp to get build you can check with in GIMP master folder you will find the INSTAL.in file which will tell how to build the application on particular platform with dependencies required. We are using Linux environment to build the application.

To build this process we have two different type of build system provided one through a Tarball and other building from latest source (Git).We used the latter one to build the application.

Being a product of GNOME ,gimp use many of their pre existing libraries which they utilize for their other products as well like GLib , GObject ,GTK+, Cairo, Pango.

The two major libraries which gimp uses are GEGL and babl which one need to install before building this application. And both can be installed either using tarball or git on system before building. Links for both the libraries are present on wiki page of the application.[3].

Once this is done we are require to just run autogen.sh which is a auto maker for configuration file. Once configuration file is generated we can run it to auto generate makefile. Also this is where you get to know if you have installed all libraries or not. This is the place where we got stuck as I already had a previous gimp installation and did not change the path for the new installation and end up having conflict between both of the installation which got resolved by just changing the default path. Once make file is generated you can run the “make” command and then “make install” to complete the build process.

So if we see the major flow of the build after installing all libraries is as follows:
autogen.sh—config—make—make install

Gimp can be build for three different platform Linux , Mac and Windows. Apart from few problem which I mentioned in the earlier section we also faced following issues.

- **We got error for relative path in the prefix.** If you google you get lot of solution for this and we figured out that the path we are using is not the absolute one so we were able to fix this issue after searching for two three time.
- **Also we got old GTK+ version error.** As I already had one installation of gimp on my machine so when for the first time I ran configuration file it gave me an GTK+ old version error which I was able to fix by just making An update for the library.

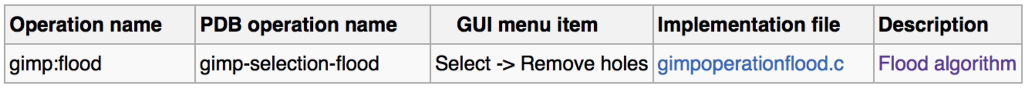
To Summaries the build process we need to have to install all libraries which are mentioned in the wiki page

7 Control Flow

8 Dependencies

Gimp has a dependency on many libraries some important ones are as follows:

1. **GLib:**
   A common library which is basically used in most of the gnome applications, it contains the various utilities and data structures needed by the program in c.
2. **GEGL:**
   It is a graph based image processing library which is used by GIMP in order to support high bit depth images.
3. **babl:**
   It is a support library for GEGL. It is helpful in color-space conversions.



Operation name	PDB operation name	GUI menu item	Implementation file	Description
gimp:flood	gimp-selection-flood	Select -> Remove holes	gimpoperationflood.c	Flood algorithm

Figure 6: Documentation on operation algorithms

4. **PDBTypes.h:**
   It is helpful in defining enums for various fields contained in PDB.

5. **GDK:**
   It helps by acting as a wrapper around low level functions.

6. **GObject:**
   It is a library which is used to implement C objects.

9 Algorithms

To find the underlying algorithms behind the operations performed on GIMP, the wiki for the same had a link to the algorithms used. We expected to find all the underlying algorithms here but encountered a problem. There was only one operation described here shown in figure 6.

Since this is implemented in the gimpoperationflood.c, the best way to find the algorithms for the other operations performed would be to have a look at where this file is located.

The app/operations folder is where this file located. Looking at the other files in the folder, we got an idea of where to look incase we had to change or add any further operations. Since these operations are triggered using the GUI, changing something here would cause us to look at the UI elements to be updated as well.

10 Problem and Solution

We already discussed few problem which we faced during previous section here again we are going to put few specific issues which we faced.
Make command stops with the error 'No rule to make target all': So looking at different solution on line we figured out that with in the configure.ac we missed to update our directory path because of which this error occurred. We can also provide prefix when running the autogen.sh file.

aclocal: error: non-option arguments are not accepted: Tried editing the project environments setting to check if ACLOCAL_FLAGS variable doesn't appear here or is not printed boldly. Also remove if any quotes are present from the ACLOCAL_FLAGS

11 Graph

Using LocMatrics we were able to visualize application in more detailed manner by creating different graph like below stating Complexity to biggest function in the application.

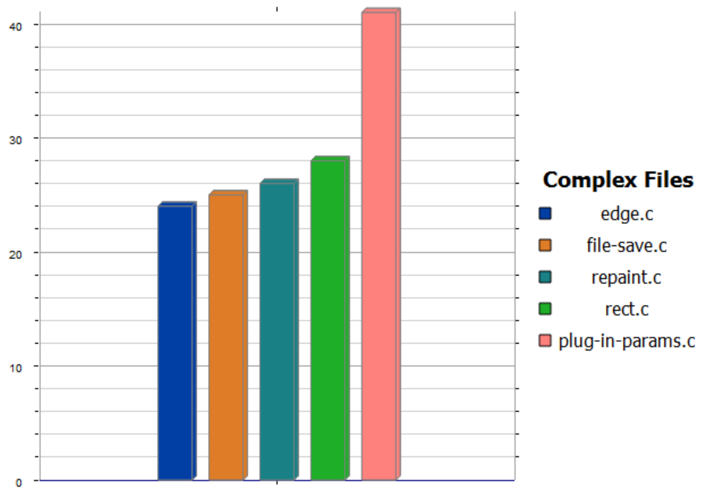


Figure 7:

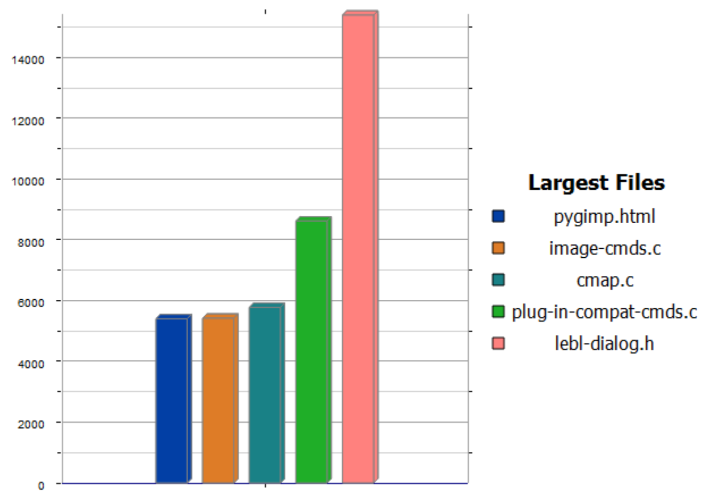


Figure 8:

12 Conclusion

After doing the analysis of the GIMP code base things which we learned are that if the code is well written with proper documentation its not tough to do the comprehension. In case of Gimp like we already stated in abstract that it has well written code base with proper documentation which give you easy understanding of the code. And when you use different visualization tool it becomes more easy to get proper call graphs which help to locate entry point in code when you build your code. Other thing what we concluded is that, one should not start doing analysis without building the application, building application first ends up letting you know not only entry point in the code but also what all different libraries are required to build the application which in turn sometime are meaning full enough to determine the purpose of application.

References

[1] http://www.locmetrics.com/

[2] https://github.com/AlDanial/cloc

[3] https://wiki.gimp.org/