Final Report on Exploratory Change Analysis Techniques of Java Programs

Group 3:

Hao Nguyen, Ahmad Masri, Eddie Kim, Jiho Kim, Valerie Kim

December 18, 2022
SENG 541 Fall 2022
University of Calgary
Dr. Robert Walker
TA: Mohammad Reza Kianifar

Project & Tool Selection

Projects Selection:

The way we picked our projects was based on the quality of documentation, number of contributors, familiar platform/framework/programming language (Java in our case), and having multiple revisions with clear change logs. We mainly used github to search for these projects, and voted on which would be best for our project analysis. Project star history as is plotted next would be a positive contributor to our choices.

In the following table, we are describing each project with the motive behind our choice of selection.

Project Name	**Description**	**Motive**
JMusicBot [1]	Chat-based Interface for Music Streaming	Familiar Platform Strong and independent documentation Very clear architectural changes with new customization features.
Processing 4.0 [2]	Art-focused Graphics Library + IDE	Changes were well documented on different releases
Archi [3]	Enterprise Planning for ArchiMate ML (like UML for Business)	Independent File Specifications in ArchiMate Modeling Language specifications Long Development History (>10 years!)

Analysis

Lexical Analysis:

In this section, we chose TF-IDF over LDA because we found it to be more practical, and the frequency of words/tokens with our choice of semantic analysis tool (CDA, shown later) is a good validator for our dependency graphs. Also, LDA didn't work unfortunately, and in any case getting "topics" from the documents or code wouldn't help us that much in our analysis.

Overview of TF-IDF: [Java code used will be submitted alongside the report]

Semantic Analysis:

For our semantic analysis, at first we wanted to use Jdeps, but saw that the output of the tool is projected as incoming/outgoing dependencies on the command line, which would be super hectic to choose even if we use the Webgraphviz tool to visualize the dependencies. In that regard, we found a class dependency analyzer (CDA) tool that has many advantages and a few limitations to what it is supposed to do. Though, it was the perfect tool to extract dependency graphs between classes to verify and further explain our manual analysis of semantic and conceptual changes.
Implementation: http://www.dependency-analyzer.org/

How do we find changes of dependencies between two classes?
First, we will read through the description of hierarchical and conceptual changes that we gathered from Manual Analysis results.
In hierarchical change, a well-documented change log will tell us what classes were extended from and replaced the old class. After that, we will find classes that used the old class to give us a start point of the dependency graph. Finally, we will choose the old class as the end point for the graph.
In conceptual change, a well-documented change log will tell us added classes and their functionalities. After that, we will find old classes that used those functionalities and classes that used those old classes. Those classes will be our starting points and endpoints for the dependency graph.

Manual Analysis

For our manual analysis and for each project, we picked 2 revisions that had a well-written and worthy change log, as well as a decent amount of commits that would relate to the respective change log. Furthermore, for each change, we would search for the most relevant modification in code by finding the related commit and by searching keywords throughout the code.

This helped us figure out and write down on a spreadsheet (which will be submitted with this report) the semantic and conceptual changes between version 1 and version 2 for each class or set of classes that were affected in relevance to the change log context. One example of that in the JMusicBot project would be this:

Class	Changes Explanation	Change Type
DJCommand	DJCommand replaced MusicCommand and it extends MusicCommand. This means that the DJCommand can do everything that MusicCommand can. By implementing the DJCommand, we removed the category from inside the bot and moved it to DJCommand for modularity. This means that we added an extra layer of hierarchy to our software.	Hierarchical Change

Results/Data

We can split this section into 3 parts for our results: Manual Analysis, Lexical Analysis, Semantic Analysis.

Manual Analysis:

Our in depth explanation is in the “Manual Analysis” excel sheet submitted along this report.

JMusicBot:

Class	Changes	Change Type
ownerCommand	Adding a new abstract class to replace Command	Hierarchical Change

DJCommand	DJCommand replaced MusicCommand and it extends MusicCommand.	Hierarchical Change
Listener	Listener replaced ListenerApdater extensions from the bot and functionalities associated with it	Hierarchical Change
NowplayingHandler	This class was added to handle functions that existed in different classes	Conceptual Change
PlayerManager	PlayerManager replaced DefaultAudioPlayerManagers and it extends this instead.	Hierarchical Change

Processing 4.0:

Class	Changes	Change Type
Util.java	An if statement was added that checks if the string contains '#' and then trims the string accordingly. (bug fix)	Inner Semantic Change
StatusDetail.java	The update function now checks whether the progress bar exists or not through added if-else statements.	Inner Semantic Change
PSurfaceJGOL.java	The order of the method calls was swapped, placing initDisplay() after initGL().	Inner Semantic Change
ShimAWT.java	The method displayDensityImpl is handling more branches/conditions for all display devices.	Inner Semantic Change
SketchName.java/ContributionManager.java	This new version offers more explanation to fix Processing when a user disallows access to the Documents folder for macOS users.	Inner Semantic Change
Base.java	In this version, the class is now using UUID as names for temp folders to introduce another layer of indirection.	Inner Semantic Change

Archi:

Class	Changes	Change Type
AbstractModelAction	This new abstract class was added and replaced AbstractModelSelectionAction	Architectural Change
AbstractPaletteRoot	New abstract class that acts as the format painter tool entry created by sub-classes	Behavioral Change
createToolsGroup / createControlGroup	The createControlsGroup function inside ArchimateDiagramEditorPalette class got moved to the AbstractPaletteRoot and got renamed to createToolsGroup	Architectural Change
getPaletteRoot	The getPaletteRoot function now returns a SketchEditorPalette object (previously PaletteRoot object).	Inner Semantic Change
clipboardImageTransfer	A new helper class was created to transfer images from the clipboard. Previously, this functionality was done directly from where the file needed to be transferred.	Architectural Change

Lexical Analysis:

JMusicBot:

Class	Lexical Analysis Result
OwnerCommand	• Could not extract output • No previous version to compare new class/file to
Listener	• ListenerAdapter removed from the output • Shows high chance of removal of dependency of listenerAdapter
NowplayingHandler	• Could not extract output • No previous version to compare new class/file to
DJCommand	• Could not extract output • No previous version to compare new class/file to
PlayerManager	• Could not extract output • No previous version to compare new class/file to

Processing 4.0:

Class	Lexical Analysis Result
Util.java	• Output revealed that keyword “AllowHex” was added in new version of file • Good indication that the change is valid
StatusDetail.java	• Looked for keywords that could verify change • Both files contained "contribprogressprogressbar" • Extracted words did not provide enough evidence to verify
PSurfaceJOGL.java	• Tool output did not provide enough clues • Both outputs were nearly identical • Only significant change is the added word “order”
ShimAWT.java	• The keyword "displaydensityimpli" was added in the new version of the file • Provides evidence for the change
SketchName.java/ContributionManager.java	• Output from the tool did not provide enough evidence to validate the change
Base.java	• New version added keywords “uuid” and "uuidrandomuuidtostring" • This provides supporting evidence to validate the change from manual analysis

Archi:

Class	Lexical Analysis Result
AbstractModelAction.java	• TF-IDF analysis does not help as there is no version to compare to
AbstractPaletteRoot	• New version of file added keywords: “selection tool”, “marquee”, “selection stack” and “painter tools” • Clear evidence for new tool support
createToolsGroup /	• Keyword “createcontrolsgroup” has been removed from previous version

createControlsGroup	• Output file of the AbstractPaletteRoot, the “createToolsGroup” was added • Good indication that function has been removed and moved to AbstractPaletteRoot
getPaletteRoot (SketchEditor)	• The keyword “PaletteRoot” was removed from new version • Strong indication of change
clipBoardImageTransfer	• TF-IDF analysis does not help as there is no version to compare to

Semantic Analysis:

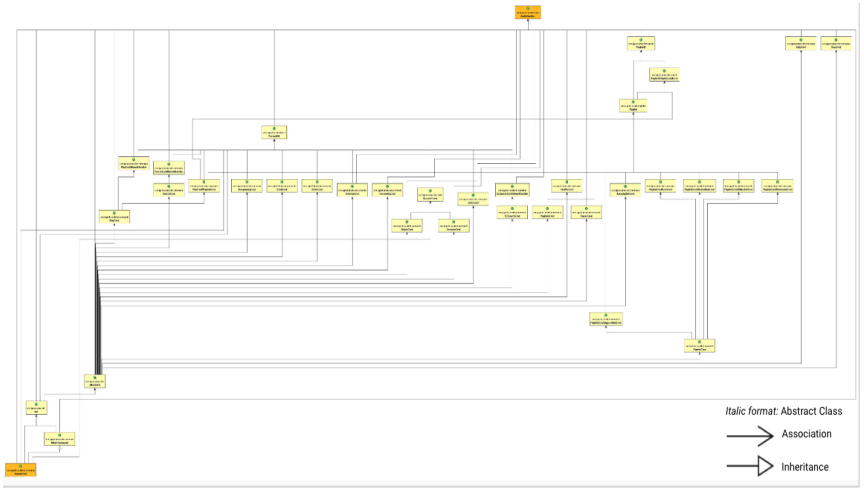
After doing our manual analysis and having pinpointed around 5 classes per project that have changed internally, been removed, or are being used by new classes, we then have our set of cases to run in the class dependency analyzer tool.

JMusicBot

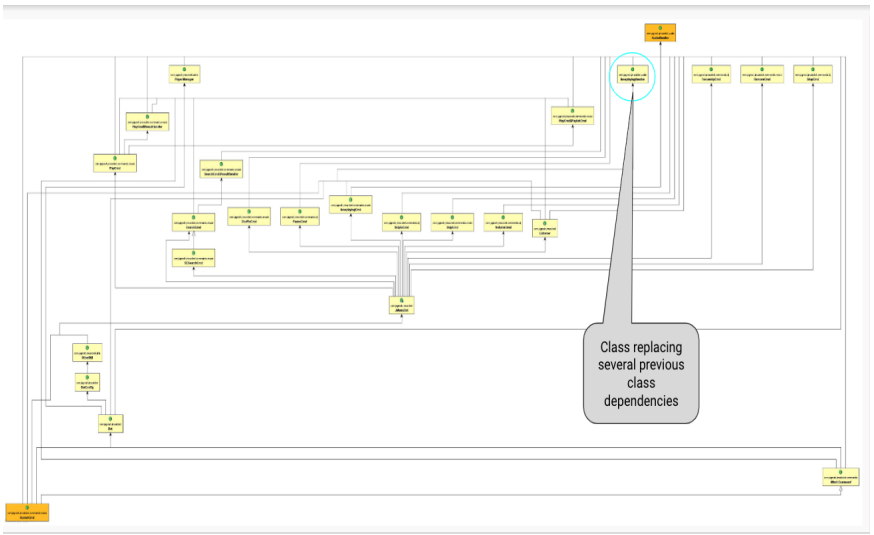
Class	**Version 1**	**Version 2**	**Change**
OwnerCommand	Command | | | {AutoPlayistCmd, EvalCmd, SetgameCmd}	Command | OwnerCommand | {AutoPlayistCmd, EvalCmd, SetgameCmd}	Hierarchical: Insert Class
DJCommand	MusicCommand | | | {PauseCmd, ...}	MusicCommand | DJCommand | {PauseCmd, ...}	Hierarchical: Insert Class
PlayerManager	DefaultAudioPlayerManagers | | | {.....}	DefaultAudioPlayerManagers | PlayerManager | {.....}	Hierarchical: Insert Class
NowPlayingHandler	{various classes} | QueueCmd	NowPlayingHandler | QueueCmd	Conceptual
Listener	net.dv8tion.core.ListenerAdapter | | {various classes}	ListenerAdapter | Listener | {various classes}	Hierarchical: Insert Class

The following 2 dependency graphs are the ones extracted for class “NowPlayingHandler”, where it is showing in V2 that this class NowPlayingHandler has been added to replace old functionality. As explained before, we used QueueCmd as our starting point since it depends on NowPlayingHandler and AudioHandler as our endpoint since NowPlayingHandler depends on it.

V1:



V2:



Processing 4.0:

Class	Version 1	Version 2	Change
Util.java	Util	Util	CDA was unable to detect changes
StatusDetail.java	StatusDetail	StatusDetail	CDA was unable to detect changes
PSurfaceJGOL.java	PSurfaceJGOL	PSurfaceJGOL	CDA was unable to detect changes
ShimAWT.java	ShimAWT	ShimAWT	CDA was unable to detect changes

SketchName.java	SketchName	SketchName	CDA was unable to detect changes
Base.java	Base	Base	CDA was unable to detect changes

Archimate:

Class / Function	**Version 1**	**Version 2**	**Change**
AbstractModelAction	AbstractModelSelectionAction | ExportModelAction	AbstractModelAction | ExportModelAction	Architectural change
AbstractPeletteRoot	AbstractPaletteRoot | ArchimateDiagramEditorPalette	AbstractPaletteRoot | ArchimateDiagramEditorPalette	CDA was unable to detect changes
createToolsControl / createControlsGroup (function)	AbstractPaletteRoot | ArchimateDiagramEditorPalette	AbstractPaletteRoot | ArchimateDiagramEditorPalette	CDA was unable to detect changes
getPaletteRoot (function)	SketchEditor	SketchEditor	CDA was unable to detect changes
clipboardImageTransfer	ExportAsImageToClipboardAction | | | PlatformUtils	ExportAsImageToClipboardAction | clipboardImageTransfer | PlatformUtils	Architectural change

Quantitative and Qualitative Analysis (with Precision & Recall):

$$Precision = \frac{TruePositives}{TruePositives + FalsePositives} \qquad Recall = \frac{TruePositives}{TruePositives + FalseNegatives}$$

1. **TF-IDF:**
   MusicBot:
   Precision: 1/(1+3) = ¼ = 0.25
   Recall: 1/(1+4) = ⅕ = 0.2
   Processing:
   Precision: 4/(4+3) = 4/7 =
   Recall: 4/(4+2) = 4/6 = 0.667
   Archi:
   Precision: 3/(3+1) = ¾ = 0.75
   Recall: 3/(3+2) = ⅗ = 0.6

We used the manual analysis as our 'actual' results. As such, the true positive values we used for the TF-IDF analysis was when the prediction (TF-IDF analysis) matched with our manual analysis.
Some examples of what we found as true positives are as follows:

- Detecting changes in inheritance

- Detecting changes in other packages and classes

When calculating for recall, we obtained the values for false negatives by finding changes detected by the manual analysis. Once we found a list of potential changes, we performed the TF-IDF analysis to check if it also detected the changes in dependencies that we found using the manual analysis.
Some examples of what we found as false negatives are as follows:

- When a new class is created. The TF-IDF analysis cannot compare new files.
- When dependencies inside an inner class is moved to the outer class (also the same for the other way around)

When calculating for precision, we obtained the values for false positives by first figuring out what the TF-IDF flags as potential changes. Once we found a list of potential changes, we performed a manual analysis to check if there was indeed a change in dependencies.
Some examples of what we found as false positives are as follows:

- Renaming of a variable or a class
- Changes (both addition or removal) of comments

2. **Dependency graphs (CDA):**

We used the manual analysis as our 'actual' results. As such, the true positive values we used for semantic analysis are the number of changes in class dependencies matched with our annual analysis.

When calculating for precision, we obtained the value of false positives by finding changes in class dependencies that are not reflected on changes in the changelog. However, since our semantic analysis is not automated, this value is impossible to obtain.

When calculating for recalls, we obtained the value of false negatives by finding all changes during manual analysis that are not reflected on changes in class dependencies, for example: inner semantic changes (bug fixes, added parameters, etc).

MusicBot:
Recall: 5/(5+4) = 5/9 = 0.5556

Processing:
Recall: 1/(1+5) = 1/6 = 0.16667

Archi:
Recall: 2/(2+3) = 2/ 5 = 0.4

Limitations/Problems

There are many **limitations** to semantic and conceptual analysis that come from the tools and methodologies we chose, which are:

Manual Analysis:

1. Extensive knowledge in the programming language and framework is needed to understand the exact semantic changes in the code we are manually scanning.
2. Tedious and time-consuming process.

TF-IDF:

1. Difficult to find changes to names for functions and variables.
2. Analysis is heavily based on understanding the meaning of a name.
3. Identified items are mostly gibberish.
4. Hierarchical changes -> no file / new file.

CDA (Class Dependency Analyzer):

1. Only shows changes on a macro level (classes, packages, containers), and not functionality or internal semantic changes.
2. Dependency graphs are often too complex.
3. Strongly dependent on the design of Class relations.

Subjectivity in:

- Project Choice - Perception of Change Log Quality
- Selection Criteria for Repositories
- Transition Version Choice - Versions Numbers are unreliable even with SemVer

Arbitrary Methodological Approaches:

- TF-IDF over LDA because LDA didn't work
- CDA over JDeps, IntelliJ for Graphic Output

Analysis Limitations:

- CDA: eyeballing graphs to choose which to analyze further
- We did not explore CDA's package and container analysis.
- TF-IDF: focusing on files with >20 changes

Conclusions

We got some positive takeaways from these tools, which are that the TF-IDF can be useful for detecting internal semantic changes in stable software, and that it can be used for validating results from different analysis tools like CDA itself where the latter would indicate a removed dependency. This class dependency analyzer is also quite useful and consistent at showing the dependencies between classes (incoming/outgoing), but as a tool alone, providing class dependencies isn't enough for most situations. Yet, our main conclusion is that these tools take too much time and manual checking for results of questionable significance.

References

[1] https://github.com/jagrosh/MusicBot

[2] https://github.com/processing/processing4

[3] https://github.com/archimatetool/archi

[4] https://star-history.com/

[5] https://github.com/adityamdk/TFIDF/blob/master/TfidfCalculation.java

[6] http://www.dependency-analyzer.org/

[7] https://en.wikipedia.org/wiki/Bridge_(graph_theory)