Journal of Engineering Science and Technology
Vol. 14, No. 1 (2019) 430 - 447

SOFTWARE EVOLUTION ON AZUREUS BIT TORRENT SOFTWARE: A STUDY ON GROWTH AND CHANGE ANALYSIS

SINAN DIWAN ADNAN

Computer Science and Information Technology College
University of Wasit, Wasit, Iraq
E-mail: sinanadnandiwan@gmail.com

Abstract

This paper provides an observation and analysis of growth and change in Bittorent Client software previously known as Azureus. Azureus software, currently known as Vuze is a cross-platform bit torrent client used to transfer files via the Bit Torrent protocol. It is written in Java and uses the Azureus Engine. According to the official website, the latest version available online is version 5.7.5.0, however, research only includes observations of Azureus from version 2.0.3.0 to version 4.2.0.8. The paper shows the growth of Azureus in terms of its features, size of methods and classes by comparing these factors to the age (number of days) and Gini coefficient values. This paper also compares the results of the software's investigated by existing research works who has the same observation; popular classes tend to increase popularity over time. A detail discussion was made to compare both observations by age and by Gini values in this research. Although the available data was scarce, the paper contains sufficient information for the measure of Azureus growth. The absence of 'Fan-Out' percentage attributes hinders the see whether the growth has a skewed graph or not. Finally, based on the results on the software evolution, Azureus has been observed with very good skewed growth

Keywords: Azureus bit torrent, Change and gini coefficients, Growth, Software evolution.

1. Introduction

Software evolution helps to know what to expect in a software project, it will show the normal and abnormal patterns in the software, infer and understand architecture/design choices made in the software, it helps to compare and pick software systems, there evolving of software is very important when the changes occurred [1]. In existing research work on software evolution, researchers have investigated the growth and change pattern in open software applications, object - oriented software applications, which can show how software grows and changes happen over time, however, pattern and scope of growth and change are in generic, it is predictable compared to be erratic [2-4]. Therefore, it leads to asking more fundamental questions for the researchers in terms of discovering more interesting insights in which, the changes occur and to what level the changes are expected [5-8]. The researchers have calculated the distances metrics to indicate how much change the class or component have in the different version of the software, they have also collected more information's about changes at fine-grained level to actions of change into individual items and also, coarse grain to gain insights into system level [9, 10]. Therefore, there are few research questions raised in previous works [11, 12] and are investigated in the current study. These are as follows:

- How much is the frequency of modification spread over the class in which, changes are expected?
- What will be the probability of class modification after creation?
- What will be the proportion or ratio of the code in which, there is no change from the start of the creation
- What is the ratio of minor and major modification in the classes?

For the current studies, the software investigated is Azureus, currently known as Vuze; a cross-platform bit torrent client used to transfer files via the Bit Torrent protocol [13]. It is written in Java and uses the Azureus Engine. Azureus was first released in 2003 at SourceForge.net with the intent to experiment Eclipse's Standard Widget Toolkit. It later became a popular bit torrent client until now. The latest version investigated is Azureus 4.2.0.8 and as such, it may be very different from the latest version of Azureus (Vuze) since the specified version was released in 2009.

Azureus is a Bit torrent client, which allows the user to download torrents from multiple locations at once. Other than providing the functionality of downloading torrents, it also caters for features such as searching, tracking and hosting torrents. Other features include RSS feeds, chat and sharing files with friends, mobile devices support, a plug-in architecture and more. The features listed below are taken from the Wikipedia page on September 2009, which is proportional to the release version of Azureus 4.2.0.8 [13].

- Ability to share torrents between friends and receive "friend boosts".
- Browsing and downloading high-quality official and/or original content on the Azureus Network.
- Chatting between friends.
- Advanced comments and ratings.
- Content search.
- Publishing content.
- Exporting media directly to external devices.

- Specification of maximum upload and download speeds.
- Opening files within the program.
- DHT tracking
- Torrent creation.
- Encryption support.
- Peer exchange and magnet URI.
- Super-seeding.
- Comments and ratings.
- Proxy settings.
- Ability to use I2P and Tor.
- Multiple UI
- Detailed Statistics.
- Beginner, Intermediate and Advanced Modes.
- Detailed Settings.
- Selective downloading/download priority.

The reason to choose Azures compared other torrents is that it is supported by a plug-in architecture. As per initial observations, there are peeks of development and points at which, changes the system, which suggests that maintenance and refactoring tasks have taken place in regular fashion, however, overall, the general trend of development, excluding the initial release stages, there is a steady climb of less and less features being added, while longer durations of time are elapsing between the changes. By analysing the change logs, growth statistics and source code we are able to identify key areas of growth and evolution for this software.

The main objective of this research report is to address the evolution, growth, and changes of Azureus software systems. By analysing the change logs, growth statistics and source code, we are able to identify key areas of growth and evolution in the systems, and explain to some extent the reasons for any anomalies that are found. The paper is organized as follows: Section 2 explains the existing software evolution research studies; Section 3 explains the bit torrent feature evolution, in Section 4 changelog audit were discussed based on Chapin's model, growth analysis was explained in Section 5, change analysis of Azureus were explained in detail at Section 6. In Section 7, observations and findings of Azureus growth and change were briefed in detail. In Section 8, limitation of the research study was explained and finally concluded in Section 9.

2. Existing Research Studies on Software Evolution

There were limited research studies carried out on software evolution. Kermer and Slaughter [14] and his team have investigated the software maintenance profile based on the granularity of modules at five different type of information systems. As per their studies, it shows that there were few modules, which changes in frequent manner and it is considered to be significant. Researcher Girba have conducted experiments on software evolution metric data for java based applications and noted that classes changed in the past will have impact of changes in the future modules, however, changes occur will be a minority. There were several other researchers have investigated on existence of class detection from past versions, which is considered as

a clone detection. Methods applied for this research are string matching, syntax tree classifiers and metric-based categorization.

Barry et al. [15] explained about volatility of software based on three dimensions, which includes change size, change frequency and change consistency. By applying a sequence analysis approach, volatile patterns were detected. The experiments were tested with 24 different type of systems, which includes maintenance logs. Kaner and Bond [16] have used a precise method of direct and indirect measurements in the software property measurement; direct measurement is computed for a function having single variable and in-direct measurement is computed for a function, which has n-variables. In this research, Line of Code (LoC) and total number of defects are observed as direct measurement; Number of defects per Line of Code are considered as indirect measurement.

By having these metrics, author have analysed the software evolution behaviours. Mendel [17] carried out research to analyse the evolving metric data distribution using Gini-coefficient, Decision frames and Lorenz curves. These research study measurements are burrowed from wealth distribution and economics study, these techniques are applied to allocate specific attributes within the population and observe how there is change over time on wealth distribution in detail by economists, therefore, in this research study same technique is applied to measure the degree of functionality within the system. Current proposed research study aims to close the research gaps in software evolution based on Lehman law of evolution.

3. Feature Evolution: Torrent Functionalities

To identify the evolution of the features of the software systems, their change logs were analysed to identify major changes to the application. Features are defined as components that enhance the applications characteristics. Observation was made from the Azureus changelog from version 2.0.3.2 to version 2.1.0.0 [13]. Functionalities related to downloading, uploading, publishing, importing, deleting and creating a torrent is observed by inputting the keyword 'torrent' in the search bar. The findings are as follows:

- Azureus 2.0.3.2: Torrents can now be stored in a user-specified directory and can be deleted from the client directly. It will also not restart download for stopped half-downloaded files.
- Azureus 2.0.4.0: Torrent can now be exported to/from XML and can be published to a tracker. Auto priority, start and stop seeding can now be disabled. Torrent size limit (1MB) is removed.
- Azureus 2.0.4.2: Torrent default encoding can set from Config. Moving completed torrent is optional. Added version, stats and comment fields when creating torrents. Torrent encoding can be rewritten. The client can download. tor not just .torrent.
- Azureus 2.0.6.0: 'Stop All Torrents' and 'Drag and Drop. torrents' added in System Tray. Added keyboard shortcuts to navigate and move torrents in the client. Added option to backup torrent files. The client can play a sound when the download is finished.
- Azureus 2.0.8.0: Added 'Queued' status. Queued torrents are stopped, however, available for an automatic start. Can Force-Start a torrent, ignoring download limits or seeding rules. Auto position the torrents. Better UI options and able to

group the torrents. When a torrent is complete, it will no longer start downloading after removed.

- Azureus 2.1.0.0: When making torrents, automatically exclude some files. Torrents can now be uploaded with xml/http interface. When torrent data is missing, the directory can be changed via the context menu. Added 'Auto-imported' functionalities. Global peer connection limit can be set from the config.

From the observation, the trends as of the stated versions were, new features were released a lot. Changes and improvements followed in the next releases (probably after receiving user feedbacks). Bug fixes happen in small amount shows that the code base is of good quality. This shows that even the main features are developed over a few releases before the best quality is achieved through refining based on user feedback. Taking this example, the other features are assumed to have the same pattern of growth.

4. Change Log Audit Analysis

Based on studies by Lumpe et al. [18], the Chapin Mode argues that the model for classification of software maintenance activities proposed by Swanson is too coarse-grained and would need to be further broken down into sub-categories to be effective. Additionally, as Chapin et al. [19] commented, it takes documentation into account, whereas, the previous model focused exclusively on software. To this end, the Chapin Model proposes twelve types of software maintenance, broken down into four categories.

- Support Interface
  - Training
  - Consultive
  - Evaluative
- Documentation
  - Reformative
  - Updative
- Software Properties
  - Groomative
  - Preventive
  - Performance
  - Adaptive
- Business Rules
  - Reductive
  - Corrective
  - Enhansive

In theory, applying this approach to the generation of change logs should result in them being more descriptive and aid in transparency throughout the team. As of Azureus version, released on 20th May, 2018, the changelog starts to label, which category does a commit belongs to are compared with Chapin' [13]. Some examples of the category are Core, Dev, UI, WebUI and Plug. Although this seems good at first impressions, it can be better by implementing Chapins model to increase the readability of the changelog.

Table 1. Chapin's model compared with other model for this study [13].

Evidence-based (Chapin et al.)		Intention-based definitions			Activity-based definitions	
Cluster	**Type**	**Swanson [5]**	**IEEE [14, 19]**	**ISO/IEC 14764 [19]**	**Kitchenham et al. [20]**	**ESF/EPSOM [19]**
Support interface	Training	NI	NI	(All)	NI	User support
	Consultative	NI	NI	(All)	NI	User support
	Evaluative	NI	NI	(All)	(All)	(All)
Documentation	Reformative	Perfective	Perfective	Perfective	(All)	Perfective
	Updative	Perfective	Perfective	Perfective	(All)	Perfective
Software properties	Groomative	Perfective	Perfective	Perfective enhancement	Enhancement	Perfective
	Preventive	Perfective	Perfective or preventive	Perfective, or perfective enhancement	Preventive	Anticipative or perfective
	Adaptive	Adaptive	Adaptive	Perfective enhancement	Corrections, or implementation change	Anticipative or adaptive
Business rules	Reductive	Perfective	Perfective	Perfective enhancement	Changed existing requirements	Evolutive
	Corrective	Corrective	Corrective	Corrective	Corrective	Corrective
	Enhancive	Perfective	Perfective	Perfective enhancement	New requirements	Evolutive

As compared in the table above, the Chapins model covers all areas specifically where some areas covered by Swanson and other models are repeated. Another reason why Chapin model is chosen instead of Swanson model is because Chapins model covers both Maintenance and evolution (Enhancive, Corrective, Reductive, Adaptive, Performance, Preventive, Groomative, Updative, Reformative, Evaluative, Consultative and Training) whereby Swanson model focuses on maintenance (i.e., Adaptive, Perfective and Corrective). Below are the shortened original sample from Azureus 2.1.0.0 changelog and its improvement with Chapins model. Below investigation result carried out in software study are as follows:

New features:

- Core | Generic update mechanisms for core, updater and swt.
- Core | Support for loading user-specific plugins from user dir and shared ones from app dir.
- Dev | Column management for any of Azureus' table views.
- Dev | Easy to use "basic plugin view": see.
- Plugin Interface:getUIManager: getBasicPluginViewModel.
- UI | Added option to auto-update language file from web (Config -> Interface -> Language).
- UI | Added option to show transfer rates in bits/seconds.
- UI | In the details view, the peer's pieces that we already have are shown in a faded color.
- UI | Linux system tray support.
- UI | My Tracker row right-click support for copying torrent URL to clipboard.
- UI | OSX: About and Preferences items are listed under 'Azureus' menu.
- UI | Added an option not to use units bigger than MB.
- WebUI| Web Plugin support for uploading torrents.
- WebUI| Webui + xml/http "access" property support for IP range.
- WebUI| Webui + xml/http plugins have had basic plugin view added.

Changes:

- Core | Ignore Share Ratio can now be non-integer.
- Core | New SHA-1 hasher: up to 25% faster.
- Core | Unix user- pref/plugin dir moved from ~/Azureus/ to ~/.Azureus/ to meet unix.
- UI | All progress/piece bars re-done (again).
- UI | Azureus should work with SWT 2.12 until we break backwards compat again.
- UI | In the Donation Window, the OK button should be on top of other controls.
- UI | Added a "what's new" item in help menu, pointing to changelog for current version.

Corrected bugs:

- Core | Files incorrectly shared if contents not a torrent when opening.
- Core | First Priority rules based on time now work across sessions.
- Core | Fix for saving of .torrent file in wrong dir: Bug #916137.
- UI | Fix for the General View in a torrent details, not being layout correctly.
- UI | Fix for the toolbar on Linux/OS Xm the Combo on OS X.
- UI | Fix for the Freeze on exit under OSX.

From the input data, it can be determined that the majority of the commits were Enhansive, Groomative or Corrective. From this, we can determine that almost all of the commits in the sample were in the Software Properties or Business Rules categories. This can obviously be attributed to the timing of the commits, them being in a phase of the project that did not involve much documentation.

5. Azureus Software Growth Analysis

According to the two of Lehman's Laws of Software Evolution, most widely accepted law, the software will go through continuous change and continuous growth in order to stay useful [5, 6, 21].

Additionally, functionalities will increase along with the growth to maintain the user's satisfaction. As such, Azureus growth and evolution are mapped into Fig. 1.

As shown in Fig. 1, the number of classes increase over the years, which are marked with different colours for each version. Blue dots are classes; red dots are superclasses. Each initial versions (i.e., 2.0.0.0 and 2.1.0.0, etc.) are separated through an average of 200 days each version with the highest from version 3.0 to version 3.1 (459 days).

As shown in Fig. 2, the growth of Azureus from version 2.0 until version 4.2 is a sub-linear growth as it slows down after version 2.4. A line is better fitted as it grows nearer to one; in this case, the quadratic fit is closer to 1. This validates the conclusion of Azureus having a sub-linear growth. It shows that the classes radically slowdown in growth rate with the increase of complexity, confirming that Azureus is following the software evolutions law patterns created by Lehman. This leads to the question; what is the cause of this sub-linear growth? There could be a number of reasons for this behaviour; the reasons are as follows:

- No new functionality is being added, or is not being added in quick succession because of the increase of complexity. Lehman's' laws state that over time a system will become more complex, causing the effect that new functionality takes longer to be added.
- Developers have slowed down working on the projects.

After conducting a research study based on the figure observation, it is found that from version 2.0 to 2.1, there was a lot of new requirement implementation hence the sharp increase in the number of classes. While around version 2.5 to 3.0 where Azureus starts adding the Vuze part (basically adding more functionalities and UI enhancement) and a lot of users are complaining about the software now being bloated while in fact, it was not (as claimed by the developers). At this point, the name changed from Azureus to Azureus/Vuze. It is changed again permanently in 3.1 to Vuze to avoid confusions among users and to settle the issue of Vuze-haters. The software, however, still retains the use of Azureus engine in next versions.

Algorithmic computation and more observations: To analyse the growth of the Azureus, different levels of abstraction are used to accumulate and allow for observation of the available data. To determine a general trend either a linear or quadratic fit is applied to give a clear indication of the growth of each abstraction. The quadratic fit is able to show if the growth is super- or sub-linear, where a super-linear has a positive x^2 value and a sub-linear has a negative x^2 value (x^2 value provides an indication of the strength of the fit is very close to 1.0 providing support for the different type of growth trends). Creating relative values from the absolute values gives another perspective on the growth of Azureus.

$$RelativeGrowth = \left(\frac{CurrentValue - Basevalue}{BaseValue} \right)$$

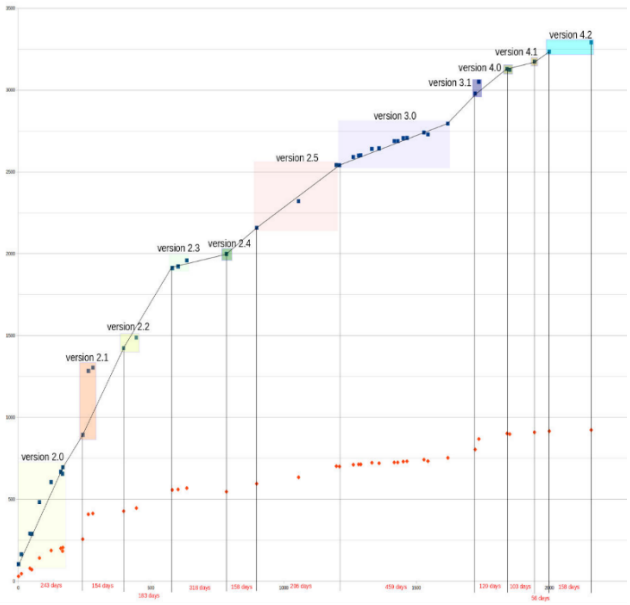


Fig. 1. Evolution of classes in software for different versions.

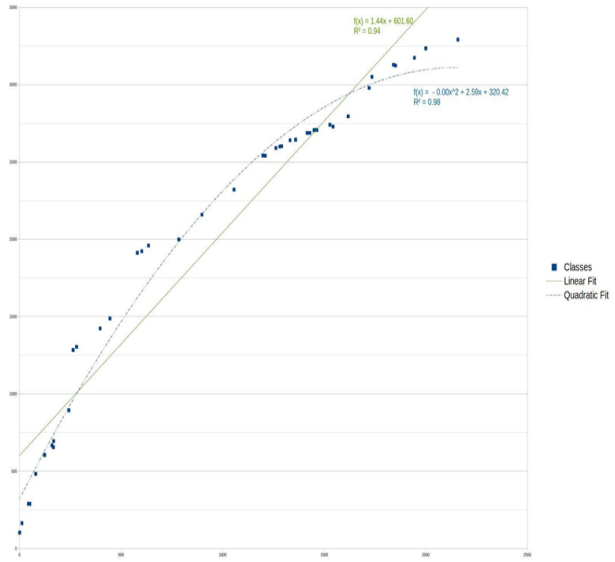


Fig. 2. Linear and quadratic fit for the class evolution.

The relative growth graph in Fig. 3 shows the classes are growing the fastest plus are of sub-linear growth trend. The methods, public methods and fields follow a linear trend. All 4 four abstractions follow a similar pattern that can be identified from version 2.1.0.0 or revisions 10 onwards to revision 18. To determine the distribution of growth we use the Gini coefficient. The Gini coefficient has indicated the distribution of a particular abstraction. More factors are taken as input to identify the growth. Figure 4 shows the distribution of Azureus' methods, public methods and fields from Revision Sequence Number 12 onwards. The values from RSN 1 to 11 were distorting the data and so we only look from RSN 12 further.

The graph shows that the methods including public methods are well distributed, whereas the fields are less well distributed. A possibility is that the "stubs" were created first, creating many fields that might not be needed in the first instance. As time progresses the growth of the number of fields actually slows down, while the method count steadily increases and are distributed more poorly compared to the fields.

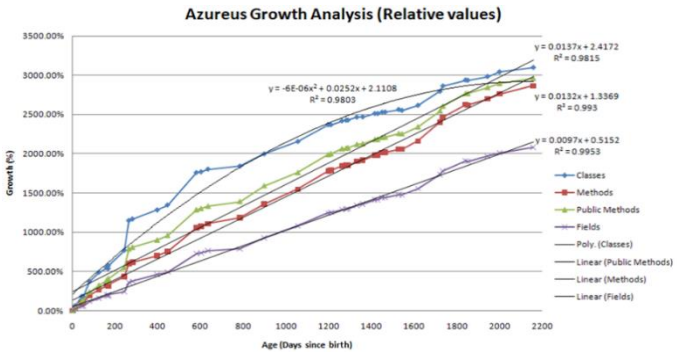


Fig. 3. Azureus growth analysis based on relative values.

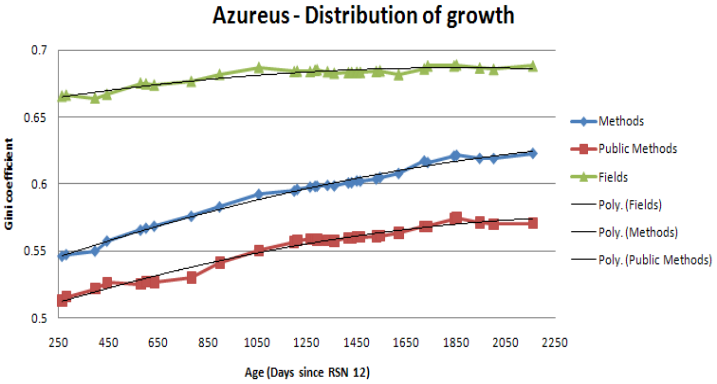


Fig. 4. Azureus growth distribution.

6. Azureus Software Growth Analysis

In this section, we try to identify if "popular classes tend to increase in popularity over time" by looking at the software systems Azureus and looking at their Gini coefficient values of the in-degree count. Using the Gini Coefficient" shows how well an abstraction is distributed in a software system. A limitation of the Gini coefficient is that it does not show, which classes are wealthy or poor, only how well an abstraction is distributed. The Gini coefficient of the in-degree count shows how well the popularity is distributed. If the Gini coefficient value is high it means that only a few classes are very popular, whereas a low value indicates that all classes are evenly popular or used equally.

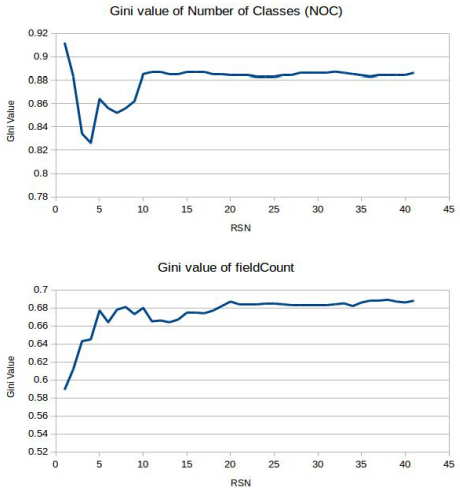
From Fig. 5, the changes are rapid from Release Sequence Number: RSN 1 to 10 (version 2.0 to 2.1) with the exception of Gini value of *methodCount*. As found out in Azureus Growth Analysis, between version 2.0 to 2.1, there was a lot of new implementation wherein the changelog, some of the version has new additions and changes amount to near 100 lines in just one version.According to Tamai and Nakatani [22], Gini coefficient shown a poor value below 0.45, the average value is between 0.45 and 0.65 and good value is 0.65 and 0.85.

It is possible to get values more than 0.85, however, the chances are the codes are machine generated [22]. From the figures below, within the first 10 RSN, Azureus has had big jumps due to the same reason as above. Again, only the Gini value of *methodCount* fluctuates a lot, however, same as the others, the quality steadily increases after RSN 23 (version 3.0).

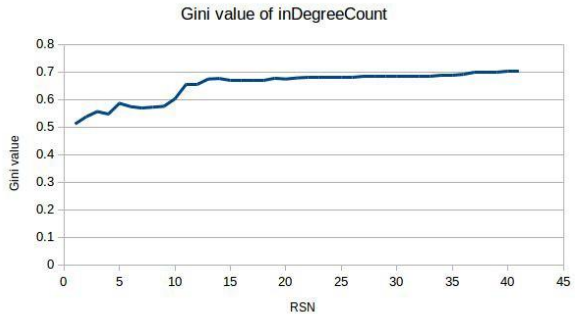
Although at this point the Azureus/Vuze issues were at its worst point, the quality remains stable. Overall, the Gini values support that Azureus has maintained a good quality over 41 RSN. The popularity of certain classes: According to previous research studies, popular classes tend to increase in popularity over time. In Azureus case, the popular classes can expand up to 3 times the original size and shown in below Table 2.

Table 2 has some examples of the popular classes that had continued growth in both sizes based on methodCount. The DownloadManagerImpl class had almost 3.2 times increase in size compared to the initial version and TRTTrackerServerImpl had increased 37 times its original size.

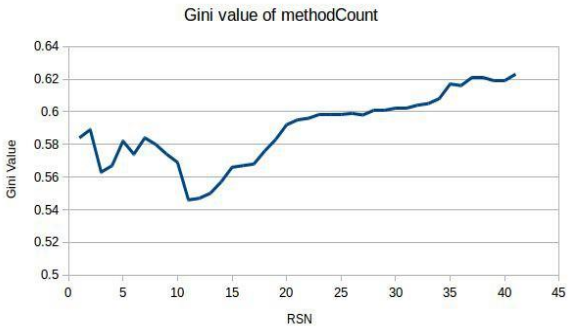
As observed the classes with the highest growth are the most popular ones (i.e., UI related, main functionality). Therefore, this confirms that, as popular classes tend to increase popularity over time.



(a) Values of number of classes and field count.



(b) Values of in Degree count.



(c) Values of methodCount.

Fig. 5. Comparison of Gini.

Table 2. Examples of popular classes with highest growth of method count.

Class	Initial	Final
DownloadManagerImpl	V3: 65 methodCount	V41: 212 methodCount
TableCellImpl	V10: 49 methodCount	V41: 132 methodCount
NetworkAdminImpl	V21: 54 methodCount	V41: 111 methodCount
IpRangeImpl	V3: 11 methodCount	V41: 30 methodCount
DownloadStats	V6: 15 methodCount	V41: 50 methodCount
SideBar	V37: 129 methodCount	V41: 155 methodCount
TRTrackerServerImpl	V2: 2 methodCount	V41: 74 methodCount
ConfigurationManager	V2: 22 methodCount	V41: 64 methodCount
TableViewSWTImpl	V23: 256 methodCount	V41: 330 methodCount
DHTControlImpl	V15: 142 methodCount	V41: 268 methodCount

More observations: A limitation of the Gini coefficient is that it does not show, which classes are wealthy or poor, only how well an abstraction is distributed. The Gini coefficient of the in-degree count shows how well the popularity is distributed. If the Gini coefficient value is high it means that only a few classes are very popular, whereas a low value indicates that all classes are evenly popular or used equally. We have tried to input more data and observe how each field changes and gaining popularity over time.

The Gini coefficient of the in-degree count shows how well the popularity is distributed. If the Gini coefficient value is high it means that only a few classes are very popular, whereas a low value indicates that all classes are evenly popular or used equally. Figure 6 shows a rapid increase for the in-degree count in the first revisions of Azureus, which is caused by the adding of many new classes in Revision 9, 10 and 11, which had many classes with a low in-degree count. For that reason, we will focus our intention on the data from RSN 11 onwards.

An example that was identified is the Debug class, as seen in Fig. 7, which is one of the most popular classes in the system. It over time increases in popularity. This is a trend that is prevalent in the data within popular classes in Azureus.

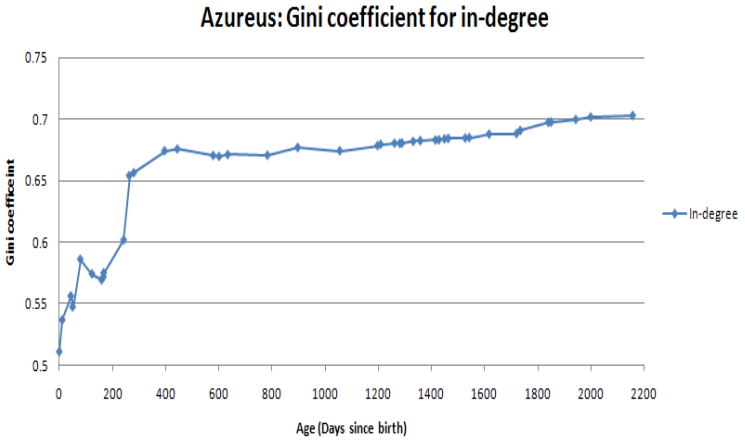


Fig. 6. In-degree count Gini coefficient for Azureus.

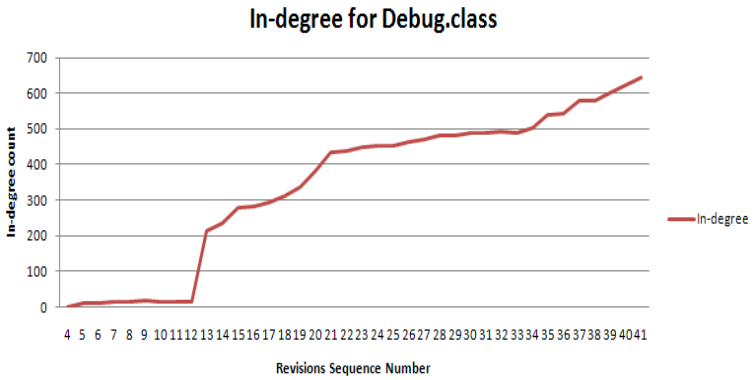


Fig. 7. Gini coefficient of debug class.

7. Discussion

In this research study, software complexity over time has been measured effectively based on investigating various metrics for growth and change analysis, complexity measures have shown a lot of richness. Software complexity is considered as a multi-dimensional construction, it depends on the magnitude, control structure, modularity and data flows. From analysing the change logs and growth distribution of Azureus, we are able to infer some of the development and architectural decisions made by the teams of programmers. Azureus expresses sub-linear growth after the initial releases of the product. We eliminate the radical changes for the first 300 days, as the product is still in its infancy and changes during this period skew the true future development of the product.

After this initial period, Azureus still displays sub-linear growth, which indicates that the complexity of the system is increasing over time and limiting the amount of growth and extra features being added. There are lots of poor, un-complex classes being added, which we can infer from the class growth increasing by 300% from the initial release and the public method count being just under 270 % increase. An example class is the AbstractIView class, which has a heavy in-degree count that is increasing over time, which supports the proposition that many low-complex classes are being added to the system.

The growth of the system also complies with Lehman's Law #5(i.e., #5. conservation of familiarity) in which, popular classes are becoming increasingly popular over time, which proves that few classes provide the most complex computations, are providing the most functionality to other classes and because of these interactions the developers are more likely to extend classes with new functionality then try to distribute the complexity over many classes, or a bias for adding complexity over changing and expanding abstractions, which would result in needing to remember more.

Although there is little increase in features and this being evident in the Azureus change logs and growth of the source code itself, it is evident that work is being done on the code base, namely in the matter of refactoring, especially around the 1400-day mark of the life span for the system. Although the system is increasing in complexity over time, this is not the sole reason for the development of new functionality decreasing, however, is a factor. The refactoring tasks that are slowing down development are also trying to mitigate the level of complexity, so are having

an effect on the entire system of reducing new feature introduction. Classes that were added in the software did not change the overall Gini value indicating that they do not have any dependencies, however, there is a massive drop in a lot of the popular classes in-degree count at specific time frames, notably at around version revision. This implies that there were points of large refactoring in these classes that either removed dependencies or farmed them out to smaller classes. Most classes have a very low number of dependencies. Even the most popular ones are still decoupled

By analysing the change logs, growth statistics and source code, we are able to identify key areas of growth and evolution in the systems, and explain to some extent the reasons for any anomalies that found. We have observed the growth of Azureus using two different sets of data; growth by versions across age and growth by Gini coefficient values and it is shown in Fig. 8. The first set is measuring growth by age; we can see over time, how many versions have been released to infer the growth pattern whether a linear, sub-linear or super-linear. The other sets are measuring the quality of growth by the released sequence numbers; we can see if the quality actually increase or dropped significantly and infer the reasons behind it.

Like most software's, initial version had a linear growth if not super-linear (version 2.0 to version 2.1) where the increase was rapid. As shown in the figure above, the quadratic fit shows the number closer to 1 and therefore, chosen instead of the linear fit. Hence, Azureus has a sub-linear growth as mentioned above. The slow down happens in between version 2.2 to version 3.0, which totals up to 857 days (2.3 years), which has a high chance of the software being stabilized.

Azureus had completed most of its main functions at this point. From version 3.0 onwards, development was rapid again due to adding Vuze; a social part of the application for the users to chat and interact with friends. From this point onwards, the codebase almost split into two; Azureus and Vuze, which in turns, explains the rapid development again. There is a possibility of more manpower at that point because being part of the open source community, more contributors have joined in the development team. However, the growth is not as fast as version 2.0 to version 2.1, which is probably due to less adding new functionality and more on to making changes and correcting bugs.

From Fig. 9, there are many sharp jumps from RSN 1 to RSN 23. A normal change in Gini value is between 2% and 4%. After that, the quality seems to be increasing steadily on the remaining RSN. This indicates that there were a lot of changes that affect the quality in the first 20 RSN. According to Appendix C, Azureus Gini Detailed Metrics, the first 10 RSN was referring to Azureus version 2.0 to version 2.1 and RSN 11 to RSN 23 was referring to Azureus version 2.1 to version 3.0. This is tally with the mentioned reason in Growth (By Age), wherein the first 10 RSN, growth was rapid just like most initial versions of softwares. In the following RSN, however, the Gini value drops significantly wherein the Growth (By Age) shows that the growth was slowing down. The possible reason for this is that although Azureus was being stabilized, some recently joined programmers had lower quality codes that affected - this is reasonable because anyone can become a contributor in the open source community. From the versions onwards, the code quality was fixed and steadily improved.

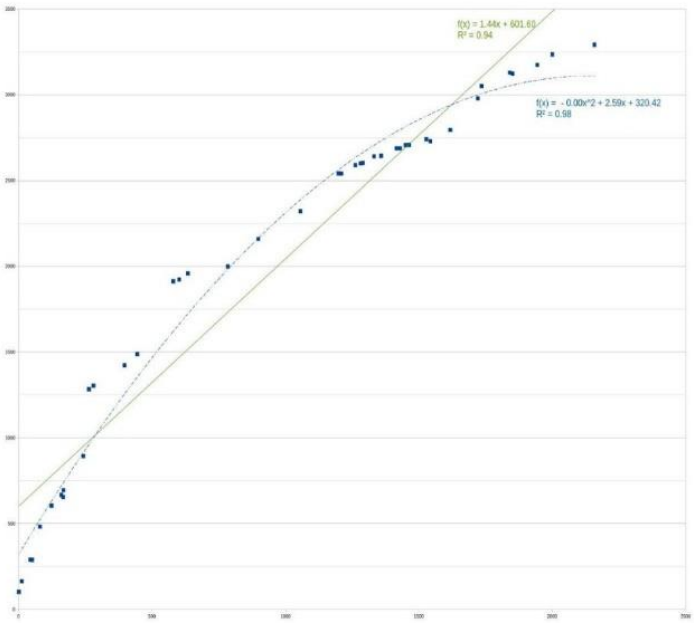


Fig. 8. Number of methods increase over the years where a linear fit (R^2 = 0.94) and quadratic (R^2 = 0.98) fit.

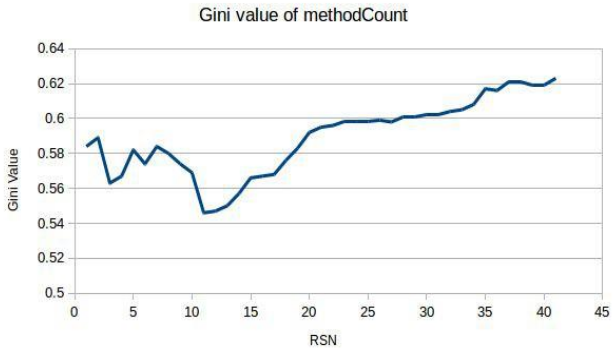


Fig. 9. Gini coefficient value of methodCount over RSN.

8. Limitations

Although the growth was able to be measured and inferred upon, there is still a limitation in terms of available data. Firstly, the only history available on the internet was from the Wikipedia, Vuze forums and some blogs containing the updates and news of Azureus development, which made it hard to exactly study the case. The most apparent news was the updates from version 2.5 to 3.0 to 3.1. Although the time for the research was ample, the research data was very limited. Secondly, the provided Gini data was insufficient. From the available data, only the growth by age and growth by Gini coefficient value was charted. The required data

to create a skewed graph was missing; percentage of classes (available) and percentage of fan-out (not available). Therefore, the third graph cannot be plotted.

Apart from the above limitations, the impact of the study is in the context of the above observations and summary are heavily rooted in the way the data was provided and not sourced by the team. Although the source of the data collection is considered reliable, because we were unable to observe nor be involved in its collection we are working on data and values pre-calculated and sourced before their use was decided and therefore, this could have impact on our interpretation of the data and it been 'made to fit' our study. Also to mention, the change logs and other generated data about the system was not observed upon creation, as the development teams of each system produced these change logs. With this in mind, it is must be known that their format and disruptions of changes made to each system were very different and at times expressed data in completely different way, i.e., Azureus explicitly indicated when a new feature was released and its functional status, or known bugs. This inconsistency makes it difficult to compare the systems change logs to each other as they express similar data, however, at different levels of intention, quality and detail. Even with these limitations, the analysis of the system expressed in this paper takes into consideration these potential issues and provides as accurate as possible observations with the data.

9. Conclusion

In this research study of software evolution, although the available data was limited, Azureus growth was properly observed and analysed. As most commercial software's, Azureus too experience a sub-linear growth despite being in an open source community. That does not mean it is of poor quality because the Gini values of the four selected factors show that the Gini values sits between 0.55 to 0.70, which is considered good. Although the skewed graph cannot be proven with the absence of an attribute, the other two plotted charts show that Azureus has a high chance of having a skewed growth. Finally, throughout the investigation, Lehman's laws have predicted many of the outcomes especially Lehman's first law of software evolution and observed that growth at all level of abstraction for Azureus are sublinear and Gini coefficient is normal on average for a developing software's.

There are long term studies were carried out for decades on impacts of software evolution, however, this little research is to show the growth and change how it is distributed over the software system. The main contribution of the research article is summarized as follows:

- In this research, how effectively skew data distribution is measured using Gini coefficient is summarized, which is a new measure applied in software, by applying this method, we have shown that how larger and complex classes grow during the evolution cycle. The other factors such as, which classes have gained complexity and increases in volume have been analysed, therefore, in general observation, popular classes will have more popularity over the period of evolution.
- Research has shown how code resists to changes and identified common pattern such as frequency of class modification, major and minor changes in the class modification, changes in complex classes, heavy changes in popular classes.

- We observed that during the stage of maintenance, there was more time spend on newly created classes. The major implication of the finding is class, which undergoes continuous changes will progressively least useful in the system.

By having the above little contribution to the study of software evolution, we have created a software evolution tool to extract software metric evolution data and generate visualization report for the software research community. Therefore, this research has made us understand the software construction, how the software looks like and how its internal structure changes throughout different versions.

Nomenclatures

BaseValue	Previous growth value of the attributes
CurrentValue	Current growth value of attributes
x2	Strength of the fit in growth

Abbreviations

LOC	Line of Code
RSN	Release Sequence Numbers

References

1. Baxter, G.; Frean, M.; Noble, J.; Rickerby, M.; Smith, H.; Visser, M.; Melton, H.; and Tempero, E. (2006).Understanding the shape of java software. *Proceedings of the 21st Annual ACM SIGPLAN Conference on Object-Oriented Programming Languages, Systems, and Applications*. Portland, United States of America, 397- 412.
2. Lehman, M.M. (1980). Programs, life cycles, and laws of software evolution. *Proceedings of the IEEE*, 68(9),1060-1076.
3. Lehman, M.M.; Perry, D.E.; and Ramil, J.F. (1998). On evidence supporting the feast hypothesis and the laws of software evolution. *Proceedings of 5th International Software Metrics Symposium*. Bethesda, Maryland, United States of America, 84-88.
4. Lehman; Ramil, J.; and Wernick, P. (1997). Metrics and laws of software evolution - the nineties view. *Proceedings of the 4th International Software Metrics Symposium*. Albuquerque, New Mexico, 20-32.
5. Swanson, E.B. (1976). The dimensions of maintenance. *Proceedings of the International Conference on Computer Engineering.* San Francisco, California, United States of America, 492-497.
6. Vasa, R.; Schneider, J.-G.; and Nierstrasz, O. (2017). The inevitable stability of software change. *Proceedings of IEEE Conference on Software Maintenance.* Paris, France, 4-13.
7. Zimmermann, T.; Nagappan, N.; and Zeller, A. (2008). Predicting bugs from history. *Software Evolution*, 69-88.
8. Lanza, M.; and Marinescu, R.(2006). *Object-oriented metrics in practice.* Using software metrics to characterize, evaluate, and improve the design of object-oriented systems. Heidelberg, Germany: Springer-Verlag Berlin Heidelberg.

9. Succi, G.; Pedrycz, W.; Djokic, S.; Zuliani.; P.; and Russo, B. (2005) An empirical exploration of the distributions of the chidamber and kemerer object-oriented metrics suite. *Empirical Software Engineering,* 10(1), 81-104.
10. Ubuntu wiki. (2013). Release naming scheme. *Development Code Names*.
11. Madhavji, N.H.; Fernandez-Ramil, J.; and Perry, D.E. (2006). *Software evolution and feedback: Theory and practice (1st ed.).* West Sussex, England: John Wiley & Sons Ltd.
12. Al-Msie'deen, R.; and Blasi, A.H. (2018). The impact of the object-oriented software evolution on software metrics: The iris approach. *Indian Journal of Science and Technology*. 11(8), 8 pages.
13. Vuze. (2018). Azureus 2/3 and Vuze. Retrieved on May 20, 2018. from https://wiki.vuze.com/w/Azureus_2_/_3_and_Vuze.
14. Kemerer, C.F.; and Slaughter, S.A. (1997). Determinants of software maintenance profiles: An empirical investigation. *Software Maintenance: Research and Practice,* 9(4), 235-251.
15. Barry, E.J.; Kemerer, C.F.; and Slaughter, S.A. (2003). On the uniformity of software evolution patterns. *Proceedings of the 25th International Conference on Software Engineering*. Portland, Oregon, 106-113.
16. Kaner, C.; and Bond, W.P. (2004). Software engineering metrics: What do they measure and how do we know? *Proceedings of the 10th International Software Metrics Symposium (Metrics 2004)*. Chicago, Illinois, 1-12.
17. Mendel, J.M. (1991). Tutorial on higher-order statistics (spectra) in signal processing and system theory: Theoretical results and some applications. *Proceedings of the IEEE.* 79(3), 278-305.
18. Lumpe, M.; Mahmud, S.; and Vasa, R. (2010). On the use of properties in Java applications. *Proceedings of 21st Australian Software Engineering Conference.* Auckland, New Zealand, 235-244.
19. Chapin, N.; Hale, J.E.; Khan, K.M.; Ramil, J.F.; Tan, W.-G. (2001). Types of software evolution and software maintenance. *Journal of Software Maintenance and Evolution: Research and Practice,* 13(1), 3-30.
20. Kitchenham, B.A.; Travassos, G.H.; Mayrhauser, A.V.; Neissink, F.; Schneidewind, N.F.; Singer, J.; Takada, S.; Vehvilainen, R.; and Yang, H. (1999). Towards an ontology of software maintenance. *Journal of Software Maintenance,* 11(6),365-389.
21. Vasa, R.; Lumpe, M.; and Schneider, J.-G. (2007). Patterns of component evolution. *Proceedings of the 6th International Conference on Software Composition.* Braga, Portugal, 235-251.
22. Tamai., T.; and Nakatani, T. (2002). Analysis of software evolution processes using statistical distribution models. *Proceedings of the International Workshop on Principles of Software Evolution*. Orlando, Florida, United States of America, 120-123.