Performing Commit Analysis to define the lifecycle stages of apps

Sourish Roy
Dept. of Computer Science
University of Calgary
Calgary, Canada
sourish.roy@ucalgary.ca
ELISA Networks Lab

Suchina Parihar
Dept. of Electrical and Computer Engineering
University of Calgary
Calgary, Canada
suchina.parihar2@ucalgary.ca

Abstract— Researchers have carried out multiple techniques to extract underlying topics that relate to software development artifacts. This paper focuses on an exploratory analysis of figuring out the most common lifecycle models followed by several android apps. We have performed a comparative study using multiple machine learning techniques to perform topic modeling on the commit messages of multiple apps. This comparative analysis also shows the best way for selecting the number of topics while performing topic modeling. The modeled topics are then compared with the commit message documents to find the similarity between them. After labeling the topics manually, this similarity measure would be used to check for alikeness between topics and models. Based on all of our findings an attempt would be made to figure out the lifecycle models followed by various apps.

Keywords—topic modelling; commit; LDA; machine learning

I. INTRODUCTION

The approach to software development plays a significant role in deciding the quality of software being delivered. The prime objective of software development industry is to deliver high quality software. Non-planned and non-systematic approach to software development, if applied for complex software requirements, will certainly result in the development of low quality and high-cost software products. This leads software professionals and practitioners to develop and deploy various software development lifecycles that can be successfully applied[24,25].

SOFTWARE DEVELOPMENT LIFE CYCLE

Software development lifecycle process is a type of structure or framework used in the development of any software product. There are many different lifecycle models defined. Waterfall model, spiral model, prototyping model are a few such models. Each model is described by a sequence of activities. The development steps or the activities may vary in each and every model but all the models will include planning, requirement, analysis, design etc. The waterfall model emphasizes more on the step-by step process. The spiral model emphasizes on risk assessment while the prototyping model takes an incremental approach in each and every phase of the development process. [26,27].

The software development life cycle framework provides a sequence of activities for system designers and developers to follow. It consists of a set of steps or phases in which each phase of the SDLC uses the results of the previous one.

Stages of software development lifecycle model:

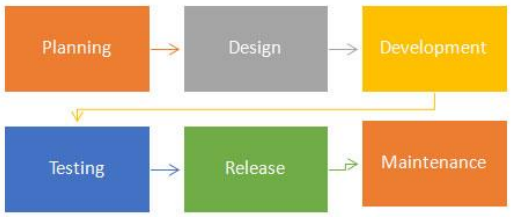

Figure 1: Stages of Software Development Lifecycle Model

COMMIT ANALYSIS

In applications, there are huge amounts of data related to commits which include commit ids, commit dates, number of commits and commit messages. Commit messages are important because when someone is reviewing a developer's work they set the stage for what the change set contains. Thus these messages can be analyzed and some logical information can be inferred from these. We could apply topic modelling or clustering techniques on these messages. Topic modeling is a machine learning technique which creates multinomial distributions of words extracted from a text corpus. This technique infers the hidden structure of a corpus using posterior inference: the probability of the hidden structure given the data. Topic models are useful in software maintenance because they summarize the key concepts in a corpus, such as source code, commit comments, or mailing-list messages, by identifying statistically co-occurring words. Among other uses, it can quickly give developers an overview of where significant activity has occurred, and gives managers or maintainers a sense of project history.

Though machine learning techniques can automatically

identify clumps of commonly recurring terms, devising an appropriate summary label for each clump/topic is harder.
A given topic extracted from a set of commit logs might consist of the following terms: “host listener change remove add multiply”. This topic might reasonably be labelled as “event handling” by a developer who understands the domain well, despite the fact that this label does not appear in the word list itself. In this paper, the approaches to label topics rely on manual intervention and are project-specific topic labels. We would like to see if that approach is feasible and easily computable.
In our approach we use a dataset of around 1400 applicaions with over 5000 commit messages. A large dataset like this gives us an abundane of data and good validation points for the assumptions we make. We also conduct certain analytical studies to visualize our results using various data representation techniques like bar graphs. pie charts, line graphs, histograms, box plots, dendrograms, scatter plots, network graphs and various other machine learning techniques to consolidate and further strengthen the understanding of anlayzing and managing software projects by making them as visual as possible and thus helping readers gain a valuable insight in this domain irrespective of their first hand knowledge in this field.

2. RELATED WORK

Marcus et al. [7] use Latent Semantic Indexing (LSI) to identify commonly occurring concerns for software maintenance. The concerns are given by the user, and LSI is used to retrieve them from a corpus. Topic modelling generates topics that are independent of a user query, and relate only to word frequencies in the corpus. In some cases, topics are named manually: human experts read the highest-frequency members of a topic and assign a label accordingly. As discussed earlier, given the topic “listener change remove add fire”, a label would be assigned like event-handling.
The labels are reasonable enough, but still require an expert in the field to determine them.
Furthermore, these labels are project-specific, because they are generated from the data of that project. A topic analysis tool such as LDA will try to find N independent word distributions within the word distributions of all input messages. Latent Dirichlet Allocation (LDA) is thus a generative probabilistic model for collections of discrete data such as text corpora. LDA is basically a three-level hierarchical Bayesian model, in which each item of a collection is modeled as a finite mixture over an underlying set of topics. Each topic is, in turn, modeled as an infinite mixture over an underlying set of topic probabilities. In the context of text modeling, the topic probabilities provide an explicit representation of a document. Principal component analysis (PCA) latent semantic indexing, and independent component analysis (ICA) are key methods in the statistical engineering toolbox. They have a long history, are used in many different ways, and under different names. They were primarily developed in the engineering community where the notion of a filter is common, and maximum likelihood methods less so. They are usually applied to measurements and real valued data. Text clustering is the key technology for topic detection, and topic detection is essentially similar to the unsupervised clustering. However, general clustering is based on global information, and clustering in the topic detection is based on incremental ways. So we should study topic detection according to clustering algorithm, and it is necessary for clustering algorithm to be in-depth and extensive research. And K-means[3] is a well-known and widely used partitional clustering method. Thus, we develop a topic detection prototype system to study how K in K-means affects topic detection [4,6,8]. In this study, we also focus on one aspect of project success by measuring the chances that a project will be alive[9] in the future, will receive updates, new features, and will not be ignored. We would also attempt to do lifecycle modelling of the topics derived but unfortunately there is no literature available in this particular domain.

3. ANALYTICS DESIGN SHEET

The initial plan of the project can be shown with the help of the analysis design sheet. It consists of 4 parts.
The first part describes the problem and its solution. There were so many questions in our minds when we started writing this paper like which techniques to be used and how to figure out the data on which the experiments need to be performed. So we stated them in the first part of our Analysis design sheet. The second part is about the data and the relationship between the data. In the third part, some attributes of the data are provided in the table which consists of commit information like commit messages, release time of apps etc. The fourth part is about the basic steps to be followed to achieve the goal of the paper. It is like a black box view of the process to be followed.

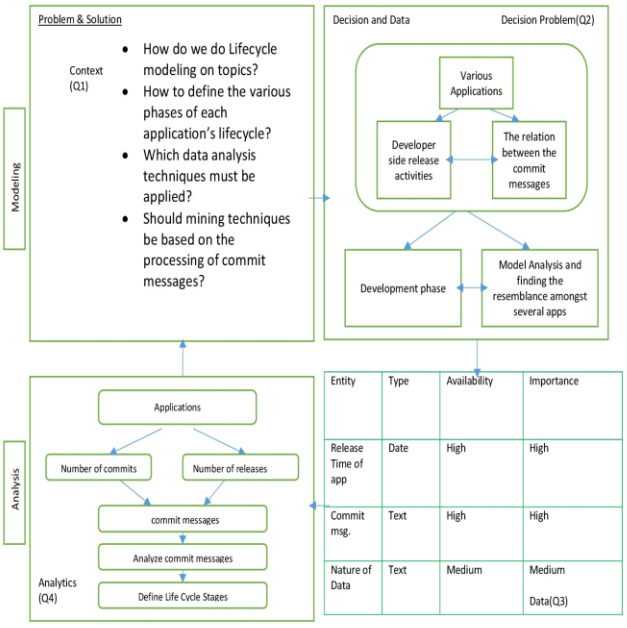

Figure 2: Analytics design sheet(Generated in MS word)

4. STUDY DESIGN AND EXECUTION

4.1 RESEARCH QUESTIONS

To formally define the directions of this study, we describe our goals in three research questions, shown below:

RQ1: How can we relate the lifecycle model of an app based on the pattern of releases?(Descriptive analysis)

RQ1.1: What is the frequency distribution of the number of releases across various apps?

RQ1.2: What is the frequency distribution of the number of days, weeks or months of app releases on Github?

RQ2: How do we acquire the topics (for life cycle modelling) based on the commit messages found from the release data of each app?

A fundamental question that arises while performing topic modelling is to select the number of clusters or topics. Several machine learning techniques have been explored to perform topic modelling so that we can get an overview of which technique is better and how are the topics related. The topics generated from this RQ could potentially help us in getting an idea of the Software Developmet Lifecycle process followed by each app.

RQ3: Can we define lifecycle stages of an app based on the topics after the topic modelling process?

Would help us get a sense of the Software Development Lifecycle Process followed by the apps. This field of research is relatively new and manual labelling will be performed to generate our results. We can also look into the commit dates to find the date since last commit to judge how alive a project is.

4.2 DATA COLLECTION

We collected a data set of approximately 1400 android applications. Each of these applications had a varied number of commit messages a total of approximately 5000. These were entirely raw csv files and in order to make meaning out of it we had to create a text corpus. Our dataset looks like the the table 2. Every app consists of the following Commit information as presented in Table 1.

COMMIT INFORMATION	DESCRIPTION
ID	An unique identifier for the commit
Commit Date	The date on which this commit happened
Commit time	The time this commit happened
Commit message	The comment written when the commit is made

Table 1: Commit information extracted from the dataset(Generated in MS Word)

	Our dataset sample		
	Commit id	***Commit Date***	***Commit message***
	95901b8b60cb29c6c9d3cffdb7150cb65e34389e	2012-03-14 18:09:51-07:00	initial commit
	fbae248ff3262162204cfdba784f242d9e2c68bc	2012-03-15 20:47:21-07:00	clipping bugfix
	f8fce7d26fbdab6d52db4cf6082c4ed9ab424a50	2012-03-15 21:43:31-07:00	condense train display by destination
	d9188b723c557e61f6a4410bef27a5c5eaa85bd5	2012-03-25 21:57:53-07:00	Merge pull request #1 from Miserlou/master Icon v1

Table 2: Screenshot of our dataset sample(Generated in MS Excel)

5. DESCRIPTIVE ANALYSIS

We should always perform appropriate Descriptive Analysis before further analysis of our data. This way we get more familiar with our data, check for obvious mistakes, learn about variable distributions, and learn about relationships between variables. That is why this aspect actually contributes in giving our research more visual clarity, to say the least and also builds the foundation for analytical software project management.

RQ1: How can we relate the lifecycle model of an app based on the pattern of releases?

An application has multiple releases. And there is a certain time span between each release. Again inside each release there are multiple commit messages. Analyzing the span between releases and later on analyzing the commit messages of each release will help us understand release activity of all the apps.

RQ1.1: What is the frequency distribution of the number of releases across various apps?

We generated a visual pattern of the number of releases and their frequencies across our dataset.
We analyzed and approximated, the number of releases that are common for apps. We have observed that the frequency of

release is considerably small for apps with large number of releases. If we dig deeper it is clear from the data provided to us, that apps with less than 40 total releases tend to have higher frequencies.

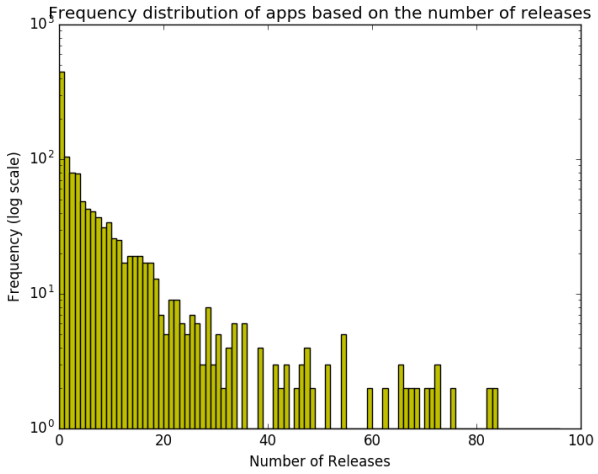

Figure 3: Frequency distribution of number of releases (Generated by Python Script)

But how disperse is the distribution of number of releases?

A **percentile or quartile** distribution would give us a good understanding of the population of the percentage trend that the release numbers follow in the data provided to us. The maximum number of releases of any app is 566. (There is only one such app). We observe that the 25th percentile or 1st quartile has a value of 2.0. The 50th percentile or 2nd quartile has a value of 17.0. The 75th percentile or 3rd quartile has a value of 53.25. The 100th percentile or 4th quartile has a value of 566.

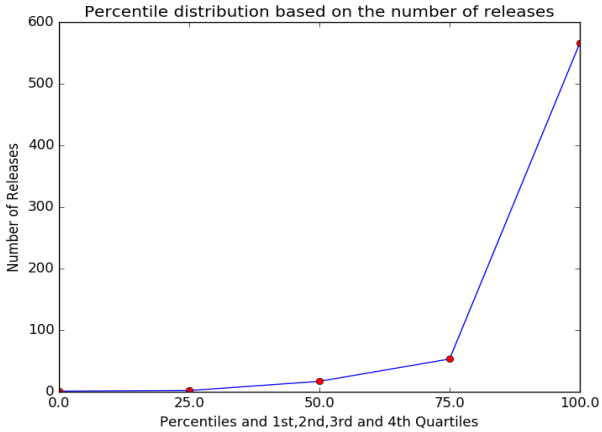

Figure 4: Percentile distribution based on number of releases(Generated by Python Script)

It thus becomes necessary now to understand the spread of releases (and the median distance) across various apps.

Let us take a look at the Box and Whisker analysis shown in figure 5, for apps with the largest number of releases (releases up to 566) and then finally a scaled plot for the apps with a maximum of 40 releases. We observe that almost 50% of the apps have 4 or less releases. (The median of the spread). Also 75% of the apps have less than 15 releases. And almost all of them (approximately 100%) have less than 30 releases. And very few apps have releases ranging from 31 to 566.

RQ1.2: What is the frequency distribution of the number of days, weeks or months of app releases on Github ?

The pie chart in figure B in appendix indicates that 70% of the total apps that are taken into account, their release time difference is between 0 to 20 days. It implies that apps make more frequent releases on github between 0 to 20 days. Further analysis has shown that 70% of the revisions take place in first 20 days of app release, 64% of the total apps have their release time difference of up to 5 weeks and 70% of the total apps have their release time difference of up to 2 months. There is a decreasing trend as the number of days/weeks/months are increased.

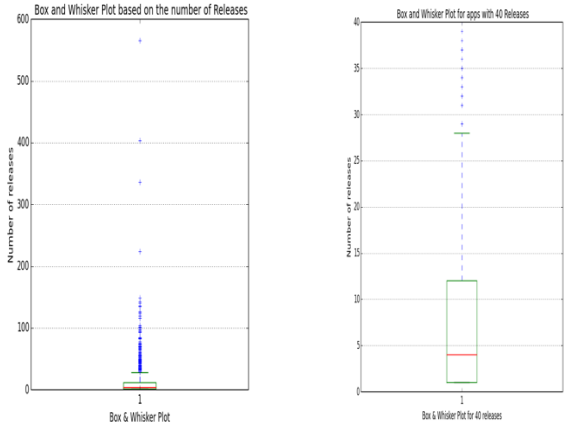

Figure 5(a) Figure 5(b)

Figure 5(a) & (b): Box and whisker analysis based on the number of releases
(Generated in Python Script)

MAJOR FINDINGS:

From the descriptive analysis we figured out certain aspects of data and the relationship between different attributes of data. The main findings were:

1. **Apps with less than 40 releases tend to have higher frquencies.**
2. **70% of the app revisions take place in the first 20 days of app release.**

These results were significant in the general understanding of the trend that data follows but we could not get something related to lifecycle modeling. Therefore, the pattern of releases alone cannot help us determine the lifecycle stages of an app. More regarding this, has been discussed in the final portion of our research.

6. COMPARATIVE ANALYSIS

Comparative analysis helps us in exploring various techniques and choosing the most appropriate technique based on the feasibility and validity of results

RQ2: How do we acquire the topics (for life cycle modelling) based on the commit messages found from the release data of each app?

In this part of the research we try to do a comparative study among various machine learning techniques to figure out the number of topics in a data corpus. A question arises that how can we select an ideal number of topics. Initially we used clustering techniques like K means and then we performed a perplexity analysis to figure out which method yields the correct number of topics before we model them. The results have been discussed at the end of this module. This report also shows a classic technique called PCA (Principal Component Analyis) for reducing the number of dimensions from the previously generated vector space.

After our data collection process as discussed earlier, we created a word cloud to get an understanding of the most frequent words.

Figure 6: Word cloud showing the most frequent words in commits(Generated by Python Script)

An analysis of Skilit Learn K Means[3] suggested to choose the value of K as the number of topics we need for our analysis. This value of K is calculated by finding the elbow point. So in this case that was 5 [1,2]. We didn't know how many topics this corpus should yield so we decided to compute this by reducing the features to two dimensions, then clustering the points for different values of K (number of clusters) to find an optimum value [4,6]. Gensim offers various transforms that allow us to project the vectors in a corpus to a different coordinate space [8]. Later we figured out a much easier way to get the number of topics.

Next we clustered the points in the reduced 2D LSI space using K Means, varying the number of clusters (K) from 1 to 10. The objective function used was the Inertia of the cluster, defined as the sum of squared differences of each point to its cluster centroid. This value is provided directly from the **Scikit-Learn K Means** technique [3]. Let's see the output of our K-Means graph.

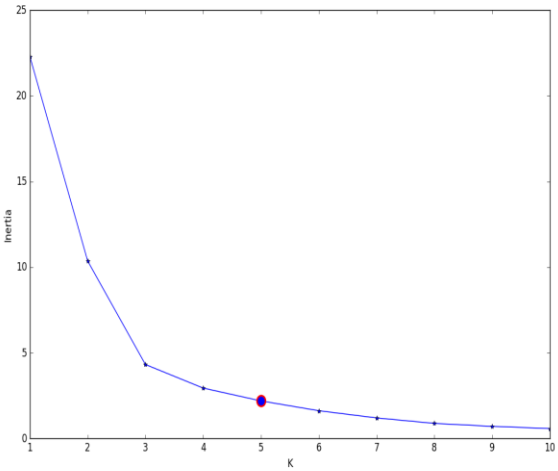

Figure 7: A K-Means Graph (Generated by Python Script)

- The elbow point happens here for K=5 and is marked with a red dot in the graph below.
- K-means is a simple unsupervised machine learning algorithm that groups a dataset into a user-specified number (k) of clusters [9].
- Therefore, when using k-means clustering, users need some way to determine whether they are using the right number of clusters.
- One method to validate the number of clusters is the *elbow method*. The idea of the elbow method is to run k-means clustering on the dataset for a range of values of *k* (say, *k* from 1 to 10 in our analysis above), and for each value of *k* calculate the sum of squared errors (SSE).

We then re-ran the K Means algorithm with K=5 and generated the clusters in figure 8.

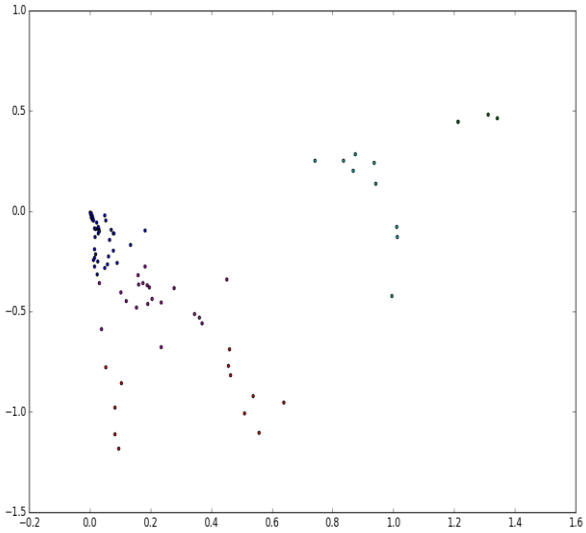

Figure 8: Generating Clusters (Generated by Python Script)

The cluster analysis gives us the following advantages: Fast, robust and easier to understand. Gives best result when data set are distinct or well separated from each other.

We then ran the full LDA transform against the BoW corpus we had , with the number of topics set to 5. And for LSI, we load up the corpus and dictionary from files, then apply the transform to project the documents into the LDA topic space. A topic analysis tool like LDA will try to find N independent word distributions within the word distributions of all the messages. These N word distributions effectively form topics: cross-cutting collections of words relevant to one or more of our commit messages. LDA extracts topics in an unsupervised manner; the algorithm relies solely on the source data, word distributions of messages, with no human intervention.

LDA and LSI are conceptually similar - both are transforms that map one vector space to another.

SYSTEM DESIGN

We chose Python as the programming language throughout this report, mainly because of its straightforward, compact syntax, multiplatform nature and ease of deployment. We used a package in Python named Gensim[15] to achieve topic modelling along with their probabilities.

The following steps clearly demonstrate the methodology involved in generating LDA topics in a simple and detailed manner:

This process is not strictly sequential. There are multiple ways to achieve similar results. The above series of steps summarize in a nutshell the steps we followed for achieving modelled topics efficiently using GENSIM.

The results are as follows. It has been seen, that each topic is made up of a mixture of terms. Let's look at the following diagram to see the distribution of topic words for topic 1 of an app named BART.

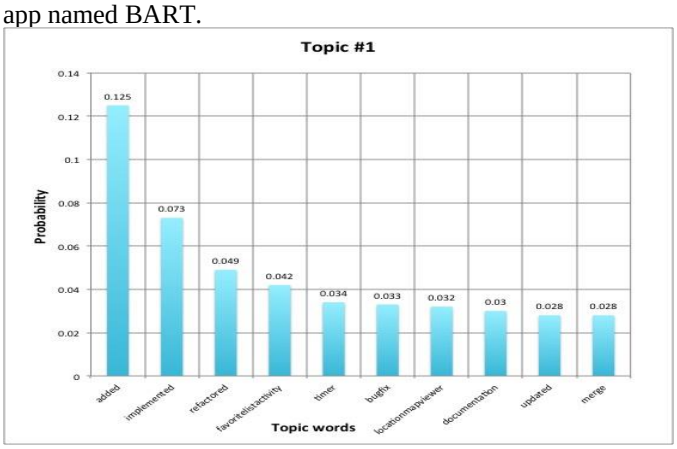

Figure 9: Topic words and their probabilities

Thus for more details the probability for the app BART of the terms in each of it's topics are shown below:

i. 2016-11-06 19:19:18,162 : INFO : topic #0 (0.200): 0.125*"added" + 0.073*"implemented" + 0.049*"refactored" + 0.042*"favoritelistactivity" + 0.034*"timer" + 0.033*"bugfix:" + 0.032*"locationmapviewer" + 0.030*"documentation" + 0.028*"updated" + 0.028*"merge"

ii. 2016-11-06 19:19:18,163 : INFO : topic #1 (0.200): 0.072*"icon" + 0.055*"response" + 0.041*"handling" + 0.039*"improved" + 0.038*"route" + 0.033*"timer" + 0.031*"notification" + 0.031*"local" + 0.031*"test" + 0.031*"reminder"

iii. 2016-11-06 19:19:18,163 : INFO : topic #2 (0.200): 0.095*"service" + 0.063*"route" + 0.043*"serve" + 0.036*"text" + 0.034*"touch" + 0.027*"input" + 0.026*"user" + 0.026*"messages" + 0.025*"display" + 0.025*"real-time"

iv. 2016-11-06 19:19:18,164 : INFO : topic #3 (0.200): 0.060*"add" + 0.052*"merge" + 0.051*"tweakin" + 0.037*"use" + 0.033*"night" + 0.033*"late" + 0.031*"display" + 0.029*"initial" + 0.029*"stations" + 0.028*"absolute"

v. 2016-11-06 19:19:18,164 : INFO : topic #4 (0.200): 0.052*"view" + 0.051*"notification" + 0.051*"remove" + 0.039*"update" + 0.034*"routeresponse" + 0.034*"service" + 0.031*"begin" + 0.031*"etd" + 0.028*"initial" + 0.027*"check"

We also analyzed the top 5 terms of each topic from 5 android apps namely BART, batphone, iFixitAndroid, Indic-Keyboard, KISS . The results have been shown in table 3.

Next we conducted a Principal Component Analysis (PCA). It is useful for eliminating dimensions. So, we've plotted the data along a pair of lines: one composed of the x-values and another of the y-values. PCA components are the underlying structure in the data. They are the directions where there is the most variance, the directions where the data is most spreadout. The figure 9 shows the PCA plot.

	Topic Number	***Top 5 terms***
	1)	bart, fix, train, service, allow
	2)	icon, response, handling, improved, route
	3)	service, route, implemented, text, touch
	4)	add, merge, tweakin, use, night
	5)	view, notification, remove, update, routeresponse
	6)	added, minor, update, add, match
	7)	update, develop, branch, androidimagemanager, developer
	8)	remove, fixed, topic, screen, display
	9)	add, changes, image, back, move
	10)	rename, guide, view, move, use
	11)	name, device, add, list, project
	12)	clean, progressbar, added, add, holoeverywhere
	13)	imagemanager, readme, loaderimage, sort, submodule
	14)	fix, change, added, warnings, page
	15)	closes, make, list, updated, guide
	16)	use, clean, 'master', db, merge
	17)	branch, merge, better, commit, useless, cleanup
	18)	updated, serbian, translation, created, documentation
	19)	added, implemented, refactored, favoritelistactivity, bugfix
	20)	renamed, igeopointinfo.iconurl, merge, bookmarklistactivity, fixed

Table 3 : Topic 5 words of 20 topics across various apps

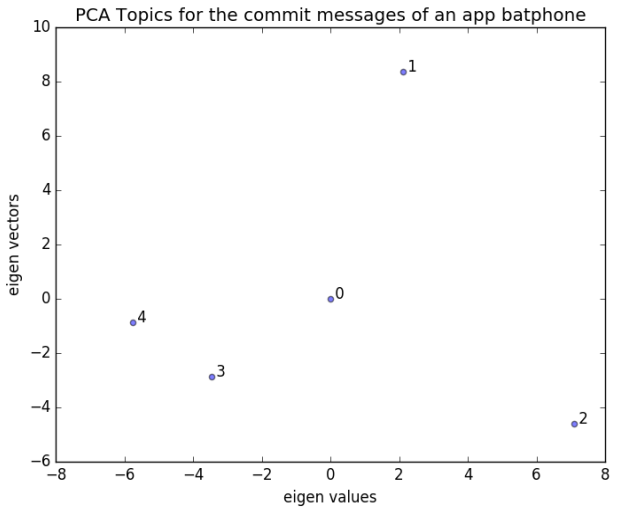

Figure 9: A labelled PCA graph based on the topics of batphone app (Generated by Python Script)

When we get a set of data points, we can deconstruct the set into eigenvectors and eigenvalues. Eigenvectors and values exist in pairs: every eigenvector has a corresponding eigenvalue. An eigenvector is thus a direction (vertical, horizontal, 45 degrees etc.). An eigenvalue is a number, telling us how much variance there is in the data in that particular direction, in the results generated the eigenvalue is a number telling us how spread out the data is on the line. **The eigenvector with the highest eigenvalue is therefore the principal component.** Thus we computed our PCA graph (figure 9) for the topics generated from the commit messages of an application named batphone.

PERPLEXITY ANALYSIS

Unlike Scikit learn K- Means, Gensim provides a mechanism for reporting of model perplexity. While analyzing the LDA log files for an app named "location Map Viewer" with 5 topics we monitored lines like the following:

INFO: -7.756 per-word bound, 216.2 perplexity estimate based on a held-out corpus of 59 documents with 79 words

This means that gensim took the current mini-batch and estimated perplexity based on the adjusted variational Evidence Lower Bound (ELBO), on this held-out corpus. The adjusted bit means that likelihood is re-weighted as if the held-out corpus was the entire training corpus. This is done to make comparisons between models using a different number of topics/model settings easier, as ELBO also includes model regularization parts that can otherwise dominate the score.

This held-out perplexity is an estimate, so different mini-batches will give different scores. But in general we'll see the perplexity value decrease as training progresses.

Thus we trained our model more and reduced the number of topics to 3 and got the following result:

INFO : -6.471 per-word bound, 88.7 perplexity estimate based on a held-out corpus of 59 documents with 79 words

This shows a clear decrease in the value of perplexity extimate and an increase in the per-word bound indicating that we have selected the correct number of topics.

We also compared the perplexity of 2 separate apps namely BART and our previous location Map viewer to check the perplexity of both with 5 topics. This would help in analyzing which app can be modelled more conclusively with 5 topics. As shown earlier the location Map viewer app had a perplexity of 216.2 but that of BART app was as follows:

-8.955 per-word bound, 496.2 perplexity estimate based on a held-out corpus of 93 documents with 139 words

So, in this case while comparing the 2 apps we see clearly that the app BART needs some more training since the perplexity estimate can be reduced even more and the per word bound is also pretty low. Thus the app location Map viewer is more conclusive with 5 topics.

For further analysis, we set a threshold value of 500 for a total of 20 apps and tried to figure out an ideal number of topics based on that score. The following table shows a subset of a list of 5 apps, their corresponding perplexity score and the number of topics selected.

APP NAME	PERPLEXITY SCORE	***Number of Topics***
BART	498.3	5
CAMPYRE	484.0	6
BATTERYWIDGET	393.2	9
XABBER-ANDROID	459.5	11
WEICIYUAN	494.8	15

Table 4: Number of topics based on perplexity score

We created a pie graph which would show us the percentage distribution of the number of topics across 20 selected apps. We observe from the graph that most of the apps (30 %) have 5 topics after topic modelling.

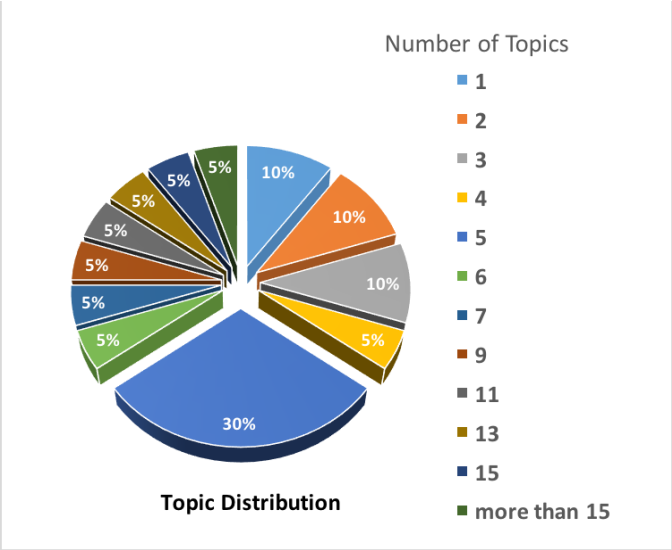

Figure 10: Distribution of the number of topics across several apps

MAJOR FINDINGS:

We compared two techniques, namely K- Means Clustering and Gensim's Perplexity Analyzer for selecting the correct number of topics and found a few major results:

1. **As the perplexity estimate value keeps on decreasing we start getting a clearer picture of the number of topics to select.**
2. **Anything below the value of 500 is an excellent choice as we have figured out experimentally.**
3. **Perplexity analysis is a better option to choose among the 2 techniques because it is more accurate. Since K means mostly points at 5 topics.**
4. **Perplexity analysis is also better since computation is easier. Also lesser lines of code are required.**

SIMILARITY AMONGST THE TOPICS OF THE SAME APP

For similarity amongst topics of the same app, we also created a network graph in figure 11, by increasing the number of topics to 10, so that we could get a better visualization which clearly illustrate the links between the top 10 Topics of the BART app. It shows us how strong the connections between the topics are and where the connections are most. Thereby implying which topics tend to cluster more.

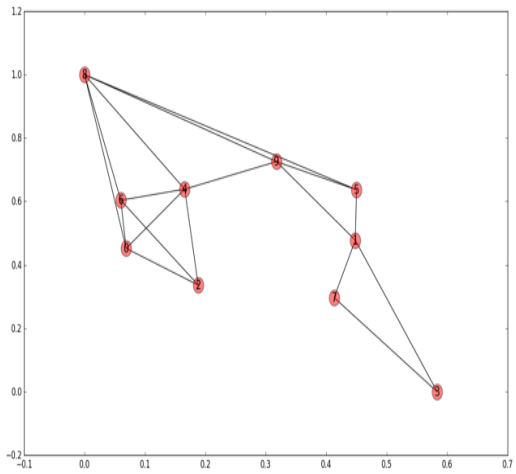

Figure 11: A network graph based on 10 topics (Generated by Python Script)

We also performed a Hierarchical cluster Analysis on topics generated from the commit messages of the BART application. The dendrogram in figure 12 is a tree diagram, showing taxonomic relationships among the topics. This would give us an insight into the similarities between multiple topics.

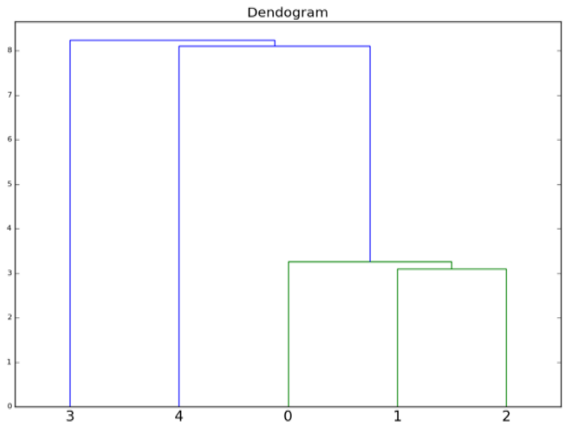

Figure 12: A Dendrogram for the 5 topics generated from the BART application commits (Generated by Python Script)

The arrangement of the clades tells us which leaves are most similar to each other. The height of the branch points indicates how similar or different they are from each other: the greater the height, the greater the difference. In our analysis, we compare the distribution of different topics among whole texts or segments of texts. Thus our results: Chunks 1 and 2 are most

similar to each other than they are to 3 and 4. Similarly we see that chunk 4 is a bit similar to the combination of chunks 0 and (1,2). Finally we can say that chunk 3 is most different from chunks 1 and 2.

Blei et al.[11,17] showed the importance of topic modelling in multiple scenarios. How immensely LDA helps in topic modelling. His contributions provide us with valuable insights for our research. Our research on topic modelling is extremely crucial in real world scenarios, because to have a better way of managing the explosion of electronic document archives these days, it requires using new techniques or tools that deals with automatically organizing, searching, indexing, and browsing large collections. On the side of today‘s research of machine learning and statistics, it has developed new techniques for finding patterns of words in document collections using hierarchical probabilistic models. These models are called topic models. Discovering patterns often reflect the underlying topics that united to form the documents, such as hierarchical probabilistic models are easily generalized to other kinds of data; topic models have been used to analyze things rather than words such as images, biological data, survey information and even commit messages. The four methods that topic modeling rely on are Latent semantic analysis (LSA), Probabilistic latent semantic analysis (PLSA), Latent Dirichlet allocation (LDA) and Correlated topic model (CTM). The basic drawback of unsupervised machine learning is that labelling the topics is a tiresome prospect. And a certain amount of knowledge in the field of software engineering is necessary to label these GIT commit message topics.

7. EXPLORATORY ANALYSIS

Exploratory Data Analysis (EDA) is an approach to analyze data sets and summarize their main characteristics, often with visual methods. Primarily EDA is used for seeing what the data can tell us **beyond the formal modeling or hypothesis testing task**.

RQ3: Can we define the lifecycle stages of an app based on the topics after the topic modelling process?

The previous similarity analysis helped us in grouping similar topics of the same app. But our main concern is to observe the similarities of 2 apps based on commit topics. For example app 1 and app 2 are similar for the commit topic 5. We would look into the actual commonalities between the topics. This would help us in grouping similar apps under the same topic.

We would later check the results and model the similar apps under a particular life cycle stage of a software development lifecycle model. But at times it is not possible and not valid to perform mapping to existing life cycle models.

Thus a question arises, is there a lifecycle model which is new for apps? We would explore this particular question and we would also show a comprehensive analysis of the frequency distribution of new app phases.

SIMILARITY ANALYSIS BETWEEN SEPARATE DOCUMENTS

This particular analysis helps us to show similarity between two separate documents. Helps us realize the relevance of two apps and would thereby help us in categorizing them. But what is more interesting is to figure out the similarity of apps in terms of commit topics instead of the whole document.

- We compared the topics of two apps batphone and BART, both had 3 topics and our result was the following:
- [(0, 0.18635691041580779), (1, 0.64397432426124468), (2, 0.16966876532294756)]
- We then set a threshold value of 0.5 for similar topics. Anything equal to or above 0.5 suggests that these two commit topics are similar for the two apps.
- In this case topic 2 has a value of 0.644 (>0.5) and thus we can say that the topic 2 is similar for both the apps.
- Similarly, we conducted this experiment for 5 apps.

BUT HOW DOES OUR SIMILARITY ANALYSIS WORK?

We created table 5 which shows how the topic mentioned in each row are similar to all the topics mentioned in each column. So the entries in the table indicate how similar a particular topic of an app in the row is to all the topics of a different app in the columns.

To clarify this even more let’s take a look at the table 5.

APP NAME	BART	CAMPYRE	WIFIFIXER	ZXING	KINDMIND
BART	----------	----------	----------	Topic 2	----------
CAMPYRE	----------	----------	----------	----------	----------
WIFIFIXER	Topic 0	Topic 0	----------	Topic 0	Topic 0
ZXING	----------	Topic 4	Topic 0	---------	Topic 1
KINDMIND	Topic 0 Topic 1	Topic 0 Topic 1	Topic 0 Topic 1	Topic 0	----------

Table 5: Similarity between topics

RESULT ANALYSIS

Also let us say, for example that the app BART has 3 topics but the app ZXING has only 2 topics, i.e., the number of topics for the apps are not the same (based on perplexity analysis). Then how do we do a similarity analysis? In the table above for the first row we can see Topic 2 is present at the intersection between BART and ZXING, what this means is that the Topic 2 of the app BART is most similar when compared to all the topics of the app ZXING. This is how we would later classify the lifecycle stages which are similar between two apps. That portion of our analysis would be more clear after we label our topics and see the labelled table of table 5 again.

It should be noted that this similarity analysis provided by genism works on probability comparisons and thus in no way does it mean that Topic 2 is the same for both the apps. But in retrospect we could say that this particular topic of the app BART (Topic 2) is similar to all the topics of the app ZXING. This analysis is the same for all the other app entries in the table.

LABELLING OUR TOPICS

For this particular field of our research we had very little prior literature available at our disposal. Thus this labelling was done more based on intuition. There were two major parameters which we focused on while labelling our topics:

1. The term probabilities of the topic terms obtained after LDA.
2. And the commit files themselves to get a sensible label.

This would later help us understand which lifecycle stage (if any) the app belongs to. The table 6 illustrates a few apps and their labelled topics.

We then placed the labelled topics in a short table (Table 7) to check which topic in the apps are most similar to each other.

APP NAME	BART	WIFIFIXER	ZXING
BART	----------	----------	BUGFIX
WIFIFIXER	BUGFIX	----------	BUGFIX
ZXING	----------	RELEASE COLLABOR-ATION	----------

Table 7: Similarity table

RESULT ANALYSIS

Table 7 shows us that the app BART's topic named "BUGFIX" is most similar, when compared to all the topics of the app named ZXING. Also, since labels have been defined based on the probability of terms and our intuition that is why the labelled topics might vary between the two apps.

8. LIFECYCLE MODELLING

For this portion of our research we figured out the underlying lifecycle stages based on the labelled topics and the commit dates for 20 apps. We created a new lifecycle model based on the unique stages and the release. We also found a few familiar labels which match existing life cycle stages and thus it was possible for us to map them to existing software lifecycle models as well.

BART (Number of Topics)	Top 5 Terms	Labels	**WIFIFIXER** (Number of Topics)	Top 5 Terms	Labels	**ZXING** (Number of Topics)	Top 5 Terms	Labels
Topic 0	service, icon, merge, initial, fix	INITIAL PLAN	Topic 0	added, change, bugfix, removed, release	BUGFIX	Topic 0	maven-release-plugin, release, prepare, zxing, rollback	RELEASE ROLLBA-CK
Topic 1	onresume, notification, usherservice, routeresponse, focus	ROUTING	---------- (No more topics)	----------	----------	Topic 1	changes, c++, german, inspection, remove	CODE CHANGES
Topic 2	bugfix, text, begin, updaterouterespo nsewithetd, properly	BUGFIX	----------	----------	----------	Topic 2	issue, pdf, add, test, remove	ISSUE TESTING
Topic 3	response, view, service, route, remove	MODIFICA-TIONS	----------	----------	----------	Topic 3	update, c++, port, issue, fix	BUGFIX
Topic 4	check, notification, icon, timer, permission	PUSH NOTIFICA-TIONS	----------	----------	----------	Topic 4	issue, fix, add, use, remove	BUGFIX

Table 6: Labelling of topics

We plotted the various lifecycle stages for the 20 apps, which we had labelled to monitor the distribution of the stages in figure 13. These stages were thus obtained by manually labelling the varied number of topics obtained and finally by defining them as lifecycle stages. This graph would give us a frequency of each distinct stage which we had discovered in the process.

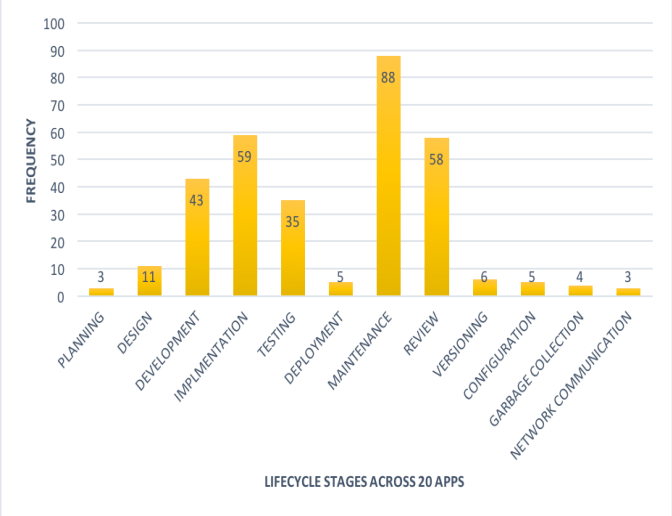

Figure 13: Frequency distribution of app lifecycle stages

One rather interesting thing which was observed in the process was the sheer difficulty of mapping to existing lifecycle models. For example, some apps like Wiktionary Mobile in our dataset follows 3 possible models namely spiral, prototyping or iterative and moreover, we could not map majority of the stages into any of the existing SDLC models and these stages have been shown in figure 15. **These two reasons primarily motivated us to figure out a new lifecycle model which most android apps might follow.** Figure 14 shows us the mapping to existing lifecycle models and from the diagram it becomes quite evident that unidentifiable models are more in number compared to familiar ones. This would clearly illustrate the percentage distribution of the apps following each model.

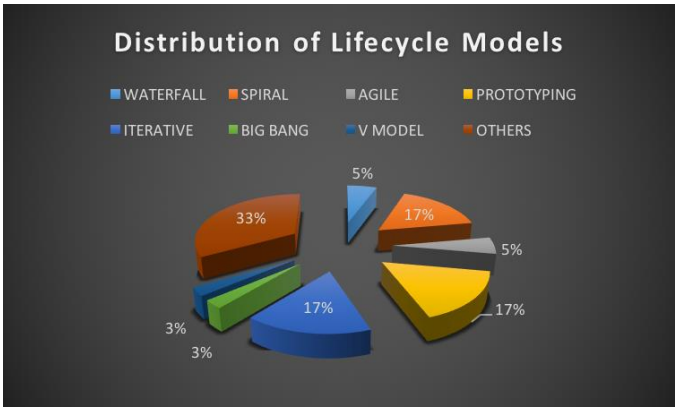

Figure 14: Percentage Distribution of the apps following each lifecycle model

From the figure 14 we can see that the 33% of the apps could not be classified under a predefined model and thus we categorized them under the submodule named "OTHERS".

The figure 15 below shows us the stages in particular apps, which could not be mapped to existing lifecycle models. We are performing this analysis to give our research a better direction in figuring out the number of such stages which are in majority and if they have an impact on our lifecycle model.

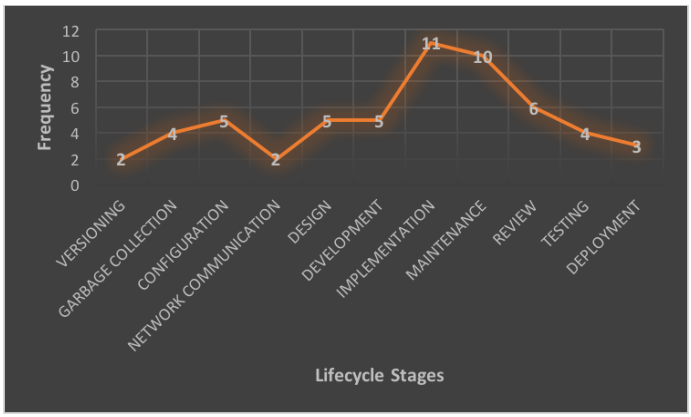

Figure 15: Frequency Distribution of the lifecycle stages which could not be mapped to a predefined SDLC model

Based on the above stages we created our very own lifecycle model. Figure 15 helps us in building our model based on the frequency of these stages. The more frequent stages have been logically included in our model so that the essential stages are not missed and also new stages are included. Thus we propose our model **Android App Development Lifecycle.**

ANDROID APP DEVELOPMENT LIFECYCLE MODEL (AADLC)

Software Development Lifecycle is crucial in application development. Mobile applications are being developed for deployment in smart phones. Looking at the rising need of mobile applications and the associated development complexity, it is imperative to have a dedicated framework lifecycle for mobile application.

Initially the mobile applications were only developed to implement calculator, calendar, alarm and currency converter functionalities. With the advent of 2G and 3G mobile networks, web based mobile applications were implemented on a variety of platforms; many of the existing web based applications were ported to platforms on the mobile device [28].

As the mobile applications have complex functionality and are different from the desktop applications, we propose the following System Development Lifecycle model to enable a systematic approach in development. This model is a combination of Iterative/ Prototyping and Spiral model. The phases of AADLC Model are:

1. CONFIGURATION
2. CONSTRUCTION
3. ANOMALY DETECTION
4. GARBAGE COLLECTION
5. DEPLOYMENT
6. MAINTENANCE

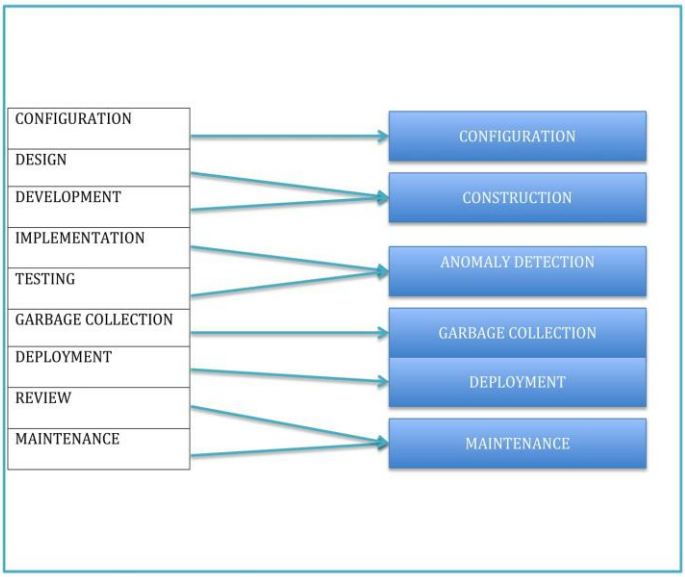

Figure 16: The lifecycle stages of the AADLC model

The first phase of our proposed model AADLC is **Configuration** phase where the parameters are tuned based on requirement analysis of a project. The second phase is **Construction** where all the design and development of an app is performed. Then comes an interesting phase named **Anomaly Detection** where the error checking is accomplished and the changes are implemented. The **Garbage Collection** phase is a cleanup phase where clean up functions are executed to uninstall obsolete software versions and delete files which are useless. The next phase is **deployment phase**. Often many of these pieces are overlapped, for example, it's common for development to be going on while the UI is being finalized, and it may even influence the UI design. Additionally, an application may be going into a testing phase at the same, that new features are being added to a new version. In the **Maintenance** stage the issues are reported and contributions are made by other developers to make an app more successful. Furthermore, these phases can be used in any number of SDLC methodologies such as Agile, Spiral, Waterfall, etc. We also felt the importance of considering release cycle times for lifecycle modelling and thus we performed an analysis on the commit messages by grouping them into release cycle stages based on these release times.

STAGE	COMMIT DATES	COMMIT MESSAGE
Pre Alpha Stage	8/09/2013–11/09/2013	-First commit, adding a readme file -Second commit to Bitbucket. Also removed the cloned directory
Alpha	13/09/2013–09/02/2014	Version 0.1.1_alpha Version 0.2.2-alpha Version 0.3.0_alpha Version 0.4.0_alpha Version 0.5.0_alpha
Beta	09/02/2014–04/03/2014	Beta realease version 8.0.1 Version 0.9.0_beta
Open and Closed Beta	04/03/2014–29/03/2014	0.14.10_BETA 0.14.13_BETA 0.14.19_BETA
Release Candidate	31/03/2014–11/02/2015	1.0.0_BETA

Table 7: Software release cycle stages of KINDMIND app

STAGE	COMMIT DATES	COMMIT MESSAGE
Pre Alpha Stage	14/03/2014	-Update README.md -Move things around in README.md
Alpha	17/03/2015–02/10/2016	-Yaaic 1.1 Release -Update version name to '2.0 -Use build tools version 22 -Update support library dependency to version 22.1.1 -Update SDK and dependencies to v23.*
Beta	02/10/2015–05/04/2016	-Bump version to 2.0.1 for beta release -Set version to 2.0b (beta builds)
Open and Closed Beta	05/04/2016–18/04/2016	-Use build tools 23.0.2 to build pircbot library -Update gradle to 2.10 and the Android plugin to 2.0.0
Release Candidate	07/06/2016	-Update Android gradle plugin to version 2.1.2

Table 8: Software release cycle stages of YAAIC app

We successfully found a strong relationship between **release cycle times** and **release lifecycle stages**. The following two tables (Table 7 and Table 8) show us the software release stages of the two apps KindMind and Yaaic. Here we show a clear illustration of the commit dates, messages and their corresponding release stages.

9. THREATS TO VALIDITY

1. Size of the commit messages is not clearly indicative of the number of topics. For example a commit file with a size of just 864 bytes (Xabber android) produces 11 topics compared to 147 KB (Wifi Fixer) which produces just a single topic. We anticipate that there might be a lot of repetitive words which make the size so huge, and that the number of topics are based more on the uniqueness of the bag of word values and the complexity of their spread in the vector space which is quite fascinating.

2. We performed a comparison between K means clustering and LDA topic modelling to find the correct number of clusters.

In our K means diagram the value k=5 for the app BART shows an estimation of the number of topics to be selected. But later on after perplexity analysis we can see that number of topics= 3 models our topics in a better way. This validates that perplexity analysis is a better approach for figuring out the number of topics.

3. The number of releases do not give us any information regarding the lifecycle models followed by an app. But instead we can only derive the patterns of releases and how dispersed they are. Also since the releases are nothing but a collection of commits by the developers, the relation between the releases and commit messages had to be validated. We performed several machine learning techniques on the number of releases and used several data analytics tools to find a possible relation between the number of releases and life cycle stages but could not find a valid result. And thus we finally arrived at our RQ3 where we look in to the individual app releases and can observe a possible relation between the app release lifecycle and the life cycle models. However, in the final portion of our analysis we perform release lifecycle modelling by labelling the various release cycle stages. This final portion is valid since it is based entirely on real world data. Thus the mapping to release cycle stages was possible.

10. FUTURE WORK

After manually labelling the several topics generated from the multiple commit messages of innumerable apps, we would like to check if automated topic labelling is a feasible option or not [24]. We would also like to check for the aliveness of our apps by digging into the commit dates and looking for the interval since the last commit. If possible, we would further strengthen our analysis by looking into the number of commits and check for the mention of pull and push requests inside the messages. This might help us in two things. To check how

popular an app is among developers and thus if popularity correlates with characteristics of a repository. Does popularity correlate with characteristics of a repository like number of commits and the keywords in the commit messages?

For our exploratory analysis we looked into a few apps but better analysis is possible if more apps are considered. We would also try to devise a platform where automatic labelling is possible using powerful programming languages like JAVA or Python and with a backend data base support possibly MySQL.

We would also like to propose more such innovative models in the future based on both user and developer requirements.

6. REFERENCES

[1] https://datasciencelab.wordpress.com/2013/12/27/finding-the-k-in-k-means-clustering/

[2] http://www.analyticbridge.com/profiles/blogs/identifying-the-number-of-clusters-finally-a-solution

[3] http://scikitlearn.org/stable/modules/generated/sklearn.cluster.KMeans.html

[4] http://stackoverflow.com/questions/6645895/calculating-the-percentage-of-variance-measure-for-k-means

[5] http://www.nltk.org/book/ch02.html

[6] https://bl.ocks.org/rpgove/0060ff3b656618e9136b

[7] Marcus, Andrian, and Jonathan I. Maletic. "Recovering documentation-to-source-code traceability links using latent semantic indexing." *Software Engineering, 2003. Proceedings. 25th International*

[8] Li, Shengdong, et al. "The key technology of topic detection based on K-means." *Future Information Technology and Management Engineering (FITME), 2010 International Conference on*. Vol. 2. IEEE, 2010.

[9] Chen, Rudi, and Ivens Portugal. "Analyzing Factors Impacting Open-Source Project Aliveness."

[10] Gousios, Georgios, Martin Pinzger, and Arie van Deursen. "An exploratory study of the pull-based software development model." *Proceedings of the 36th International Conference on Software Engineering*. ACM, 2014.

[11] Blei, David M., Andrew Y. Ng, and Michael I. Jordan. "Latent dirichlet allocation." *Journal of machine Learning research* 3.Jan (2003): 993-1022.

[12] Blei, David M., and John D. Lafferty. "Dynamic topic models." *Proceedings of the 23rd international conference on Machine learning*. ACM, 2006.

[13] Buntine, Wray, and Aleks Jakulin. "Applying discrete PCA in data analysis." *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 2004.

[14] https://radimrehurek.com/gensim/

[15] Alghamdi, Rubayyi, and Khalid Alfalqi. "A Survey of Topic Modeling in Text Mining." *International Journal of Advanced Computer Science and Applications (IJACSA)* 6.1 (2015).

[16] Rahman, Mohammad Masudur, and Chanchal K. Roy. "An insight into the pull requests of github." *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 2014.

[17] Hindle, Abram, Daniel M. German, and Ric Holt. "What do large commits tell us?: a taxonomical study of large commits." *Proceedings of the 2008 international working conference on Mining software repositories*. ACM, 2008.

[18] http://www.itl.nist.gov/div898/handbook/eda/section1/eda11.html

[19] Nayebi, Maleknaz, Bram Adams, and Guenther Ruhe. "Release Practices for Mobile Apps--What do Users and Developers Think?." *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*. Vol. 1. IEEE, 2016.

[20] Cai, Deng, et al. "Modeling hidden topics on document manifold." Proceedings of the 17th ACM conference on Information and knowledge management. ACM, 2008.

[21] Hindle, Abram, et al. "Automated topic naming to support analysis of software maintenance activities." *The 33rd International Conference on Software Engineering, ICSE*. 2011.

[22] Borges, Hudson, Andre Hora, and Marco Tulio Valente. "Understanding the factors that impact the popularity of GitHub repositories." *arXiv preprint arXiv:1606.04984* (2016).

[23] Buntine, Wray, and Aleks Jakulin. "Applying discrete PCA in data analysis." *Proceedings of the 20th conference on Uncertainty in artificial intelligence*. AUAI Press, 2004.

[24] Automated Topic Naming Supporting Cross-project Analysis of Software Maintenance Activities (Abram Hindle, Neil A. Ernst, Michael W. Godfrey , John Mylopoulos) Abdesselam Redouane," Guidelines for Improving the Development of Web-Based Applications" Proceedings of the Fourth International Workshop on Web Site Evolution (WSE"02) 0-7695-1804-4/02 2002 IEEE

[25] Hadjerrouit, Said. "Web-based application development: a software engineering approach." *ACM SIGCSE Bulletin* 33.2 (2001): 31-34.

[26] Dr. Drrpshikha Jamwal "Analysis of software Development Models", ISSN: 2229-4333 (print), ISSN:09768491 (online), vol. 1ISSUE 2, December 2010.

[27] Vithani, Tejas, and Anand Kumar. "Modeling the mobile application development lifecycle." *Proceedings of the International MultiConference of Engineers and Computer Scientists*. Vol. 1. 2014.

[28] Cui, Yanqing, and Virpi Roto. "How people use the web on mobile devices." *Proceedings of the 17th international conference on World Wide Web*. ACM, 2008.

APPENDIX

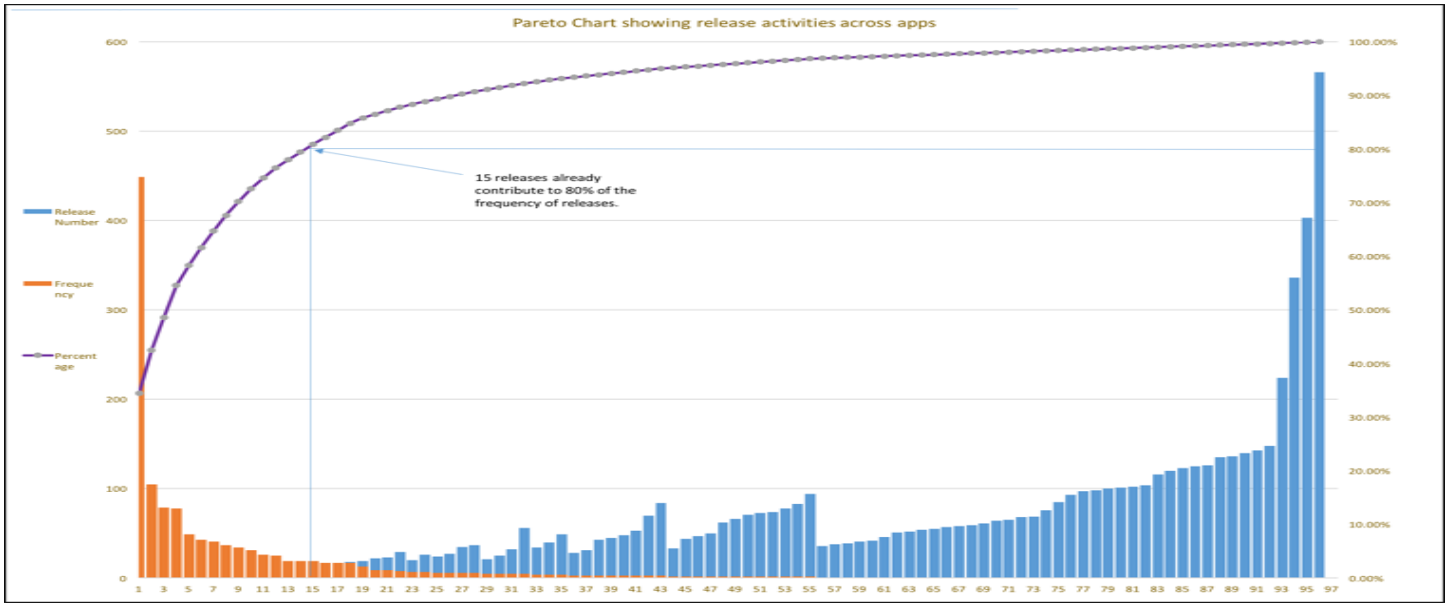

Figure A: Pareto Analysis showing release activities across

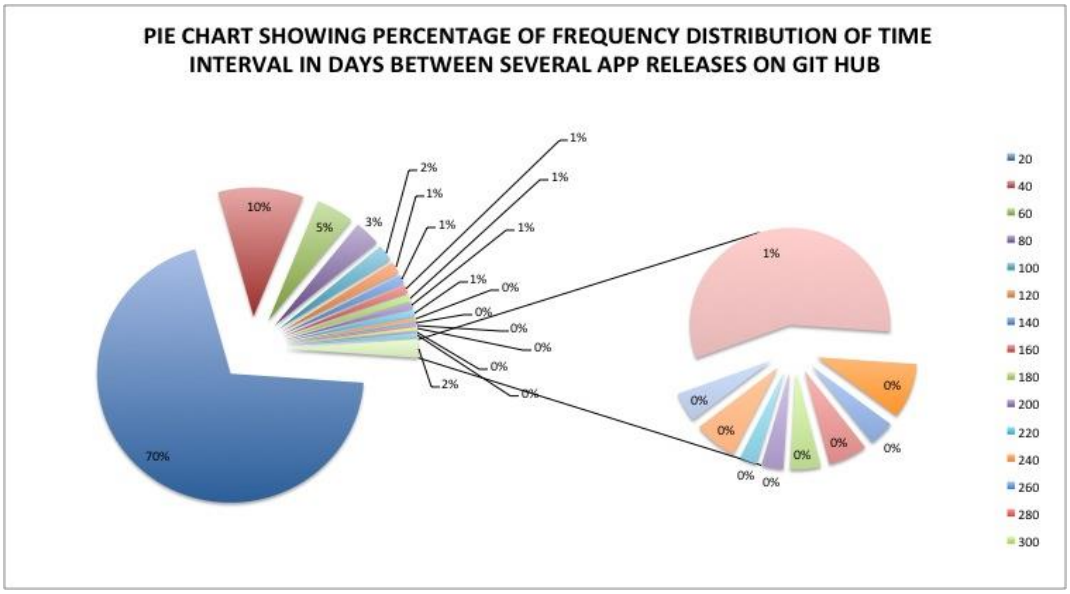

Figure B: Pie Chart showing time interval between app releases