A Digital Collection Study and Framework Exploration – Applying Textual Analysis on Source Code Collection

Wachiraphan Charoenwet
National Science Museum
Pathumthani, Thailand
wachiraphan@nsm.or.th

***Abstract*—There are many technical procedures to analyze and evaluate raw source code of any software. However, most of them focus on productivity and efficiency of the code rather than the concealed historical value, especially with the legacy software and programs. On the other hand, preservation of digital and technological collection has increasingly been one of the major concerns in certain places like museum and science center. Such activity, therefore, includes the exploration of digital collection and extraction of the undiscovered or disappearing knowledge. This study displays an effort to experimentally use topic modeling approaches to extract interesting facts and knowledge within a special kind of digital-born collection, software's source code. By exercising multiple techniques with selected subjects, this study also estimates the possibility to create a primitive framework to investigate digital collection, such as codebase, for further use in museum context.**

Keywords—Collection Study, Digital Collection, Software Heritage, Textual Analysis

I. Introduction

Over the years, high-level programming languages have gradually become the standard of protocol in software development. This particular advantage has made programming more accessible than ever, and hence allowed programmers to exert more creativity on abstract design of software system, regardless of size and business logic complexity.

Various source code analysis methods are a common practice in software development project. Many methods are frequently employed by developers to analyze and revise the source code to ensure precise output as well as promised performance of the program [1][2]. Most of the newly developed and deployed softwares have undoubtedly been gone through such pipeline of analyses.

In contrast, programming source code from legacy softwares, which had been outdated and removed from the current computer systems, is unlikely ever handled in this manner. From museum's perspective, there are multitudes of untouched knowledge in such source code, which can now be literally considered as digital collection. The classic programming source code from early age of modern programming hold historical value that is worth understanding inside raw content of the source code. This particular set of digital collection is deemed proper to be rediscovered using the software engineering techniques, which is yet to be considered a routine practice in the field of collection preservation in museum.

This study would evaluate feasible possibility to combine the aspects of software engineering and museum's responsibility on digital collection preservation [3][4], and to handcraft a fundamental framework that could unearth hidden information related to digital collection with unsupervised textual analysis techniques [5] and some other possibilities. This study also wants to offer some recommendations for digital collection curator; in order to organize this special type of collection, as the amount of the source code repositories is rapidly increasing in digital era.

II. Motivation and Research Question

Curatorial goals of the digital collection analysis are to discover and to gather pieces of information from the digital artifacts. Even though textual analysis techniques have been continuously used by data mining practitioners, this study would like to address the variation of such techniques to produce new potential approaches to analyze or examine a specific digital collection – software source code.

Differentiating mindsets – textual analysis in software engineering and in digital collection analysis, including the one in this study, should be seen as two distinguished types of process. Despite the fact that software engineers also infrequently use text mining techniques to assist documentation of source code [6], the final purpose still differs from the purpose of this study. The analysis of the source code in this study will solely put effort to search for information underneath the surface of historical source code without trying either to improve computing performance of codebase or to indicate unknown flaws, unless the flaws are relevant to the target value of the subject.

Simplifying process pipeline – another important point of concern in this study is the complexity of examination process itself. Even though artifact examination is typically a delicate procedure, digital collection curating should not be a resource-consuming process for curator. Also, the curator should not need a perfect understanding of each specific codebase. On the other hand, curator should be able to tweak the process and extract interesting information with ease. Moreover, perfecting technical capability of the tools was not a target of this exploratory study; creating guideline to adjust and fine-tuning each tool upon the nature of historical software would be kept

978-1-7281-0292-4/18/$31.00 ©2018 IEEE

for future work. Additionally, in this matter, digital collection has an ultimate advantage over other historical collections. They are replicable and, thus, virtually unbroken. The ability to completely and repeatedly dissect the digital artifacts should allow this study to simplify the handling process to the core.

According to stated motivations, this study would focus on the following research questions.

- RQ1 How well could topic modeling be used as an elementary tool to explore digital collection?
- RQ2 What should be baseline expectations or expectable outcomes for digital collection curator when applying topic modeling technique with software source code, in general?

III. Methodology

A. Framework Selection

Several tools were found during the review. Each framework or model has different advantages to consider. For example, Static Program Analysis [7] was among the most cited codebase analysis frameworks. It has strong advantage for finding programming flaws in order to prevent unseen bugs in software product. Static Program Analysis tools are available in many forms by numerous providers. Another example of source code analysis framework is the wide-range of visualization platforms [8][9] [10][11].

Instead of using Static Program Analysis tools and direct codebase visualizations which normally focus on source code performance, this study will take an alternative method to examine and evaluate source code collection from museum's frame of reference. The coalition of following procedures will be introduced.

- Textual analysis is normally used to gain human-readable insights from the large unstructured textual datasets such as news, articles, or journals. Topic Modeling and Document Clustering [12][13] are common classification techniques in text mining. As opposed to grouping similar documents together with Document Clustering, Topic Modeling techniques would be used in this study because there should be more than one topic (or area of content) across the multiple documents (source code files). Latent Dirichlet Allocation (LDA) is a generative probabilistic modeling process that can produce topic models [14]. It works on a hypothesis that every document consists of multiple themes or topics; and each document is created by appending words or terms that relate to each theme with a certain probability. In the reverse manner, by counting appearance of each word and calculating relationship of each word across the set of documents, with a number of pre-determined themes or topics, the model should be able to compute the probabilities of relationship between each word and each unique theme.
- Source Code Flow Analysis can illustrate the relationship or connection between physical objects and logical objects in the source code. A spectrum of visualization tools for source code visualization that has ability to analyze source code [8][9] [10][11] often comes with a certain level of complexity and constraints for user, which is not compliant with the target of this study. A non-commercial version of a software called Sourcetrail [15] will be used to index target codebase and create a basic flow diagrams between groups of objects in the source code. Sourcetrail allows user to navigate through the linked objects no matter they are files, functions, or libraries with ease. Sourcetrail currently supports codebase in few programming languages including C, C++, and Java.

B. Subject Selection

In actual digital collection examination, subject selection relies heavily on availability of the collections acquired by the museum. As historical value matters, a few subjects that represent the underlying meaning of historical digital collection in public domain were selected for this experiment. Source code of two programs that were created and had been used by public, during the first wave of personal computer are the unquestioningly suitable for this study. Both of them were in the similar category, and were developed in similar technology stack, but their background of development was slightly different.

1) Chulalongkorn University Writer (CU Writer, CUW) was one of a few word processing programs over a period of time in Thailand during late 1980s to early 1990s. It was a product of Thai computer enthusiasts that supported Thai language prior to the period when computer systems were commercially internationalized. It was also one of the first free softwares developed and distributed by Thai developers for educational purposes. It was created with C programming language, and operated on Microsoft Disk Operating System (MS-DOS).
2) Microsoft Word (MS Word, MSW) version 1.1a was launched in early 1990s. Its source code has been made publicly accessible by Microsoft through Computer History Museum [16]. MS Word was developed by software engineers at Microsoft a few years after CU Writer was published. MS Word was also written mainly in C. However, it operated on MS Windows 2.11 which was among Microsoft's first generation of Graphic User Interface (GUI) operating systems.

Both codebases will be passed through the same examination process as described in following sections.

C. Pre-Processing

CUW source code has 4 modules: Common Function (COMMON), Font Editor (CUF), Print Management (CUP), and Word Processor (CUW); each module is stored in separate directory.

TABLE I. CUW CODEBASE INVESTIGATION

Subject	C Program (.C)	C Header (.H)	Low Level Program (.ASM)	Configuration (MAKE FILE, BAT, etc)	Total
COMMON	13	7	6	2	28
CUF	14	9	0	8	31
CUP	23	8	0	7	38
CUW	46	10	0	5	61
Total	96	34	6	22	158

Scattered source code files are moved into one directory for textual analysis. However, some modules have files with similar names — for example, CONST.H appears in every module. Therefore, prefix is added to label origin of the files. This decision is acceptable in this context because the source code would not be executed in programming manner, but they will be analyzed as if they are simple text documents. It should also be noted that some irrelevant files, i.e., configuration files that contain vague content, had been removed to reduce noises in topic modeling process.

MSW source code is organized in a different manner. Most of the source code files are kept in main directory while several sub-routines and resources are resided in subdirectories. In comparison to CUW, MSW obviously has more pre-compiled files such as hexadecimal data or bitmap which would also be removed prior to topic modeling.

TABLE II. MSW CODEBASE INVESTIGATION

Subject	C Program (.C)	C Header (.H)	Low Level Program (.ASM)	Configuration and Pre-Compiled Objects	Total
MSW	217	131	52	392	792

D. Textual Analysis

Several programming equipment can be used to utilize the algorithms mentioned earlier. In this study, an open source statistical analysis tool called R is used because of its diverse text-mining libraries, cross-platform compatibility, and moderate learning curve for new user.

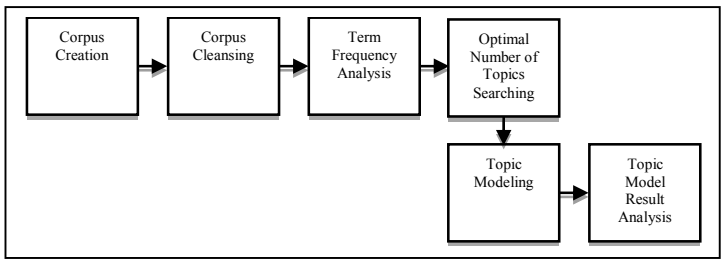

Fig. 1. Textual analysis process flow

E. Corpus Creation

Pre-processed files are parsed into a basic form of data structure for Natural Language Processing algorithms called Corpus. Basically, a corpus is a container that stores documents, each of which also stores words or terms that appear in each file. Depending on the tools, corpus can be manipulated to execute mathematical operations or linguistic analyses on individual terms across every document.

F. Cleansing

Since corpus has collected every word from the imported files, there are some meaningless terms, strings, or even the chain of symbols that should not be taken into the analysis. Cleaning process would remove unwanted data from the corpus. In this study, punctuations, numbers, and standard stop words in English are removed from both corpora. An additional step is done in CUW's corpus to replace consecutive non-English ASCII characters, which can assumingly be counted as Thai words [17], with label "thai_words" to track the appearance of Thai language in source code.

G. Word Frequency

Before creating topic-model, a quick primitive analysis with frequency of each unique word in the entire source code could depict an overview of the corpus. Frequency of each unique word is counted across the corpus. From curatorial point of view, this information can be first potential resource to demonstrate the simple technical characteristics of source code. It might as well be a clue that leads to hypothesis questions which would help framing the study — for example, "How many 'define' statements were used in a program compare to the others?" or, "Did Assembly language contribute to this software more than C language?"

TABLE III. WORD-FREQUENCY LIST

CUW		MSW	
case	1,413	int	17,997
mov	1,376	define	14,860
break	968	struct	12,287
char	964	mov	8,870
int	911	return	8,728
define	714	ffalse	7,369
void	710	extern	7,012
unsigned	659	char	6,694
else	464	endif	6,496
include	412	else	6,043
character	350	ifdef	5,882
shiftdown	349	case	5,793
extern	324	ftrue	5,457
thai_words	320	include	5,208
return	302	push	5,112

CUW		MSW	
returntempindx	281	doc	4,492
push	229	debug	4,379
tempindx	224	break	3,865
dispstrhgc	216	pch	3,817
current	209	goto	3,752

One interesting fact from word frequency analysis is the appearance of Non-ASCII characters. CUW has a significant amount of Thai content written in source code, which may be contrast to the conclusion of a previous study that open-source software usually contains less Non-English content in comparison to industrial software, either in source code comment or in identifier [18].

Another exhausted observation on the word-frequency table finds names of a few CUW developers in a number of source code files. The word "Suttipong" and "Kanakakorn", which are first name and last name of a prominent developer in this version of CUW, appear 122 times and 115 times across the source code. However, it is unclear during word frequency analysis whether similar practice also occurs in MSW source code.

Word frequency checking also leads to removing of trivial outliers in the corpus. There are two cases in which the words would be taken out: 1) words that reoccur over 90% of highest frequency and 2) words that occur less than 2% from the highest frequency. This step is necessary because basic LDA uses frequencies of words to calculate the probabilistic relationship without normalization. The biased result is likely to be produced if these outliers are not removed.

H. Topic Modeling

This study uses Gibbs sampling method to create LDA topic model. For typical topic modeling, it is nearly impossible to indicate a correct number of topics within new different input data [19]. However, as some mathematical scores can be used to evaluate model's performance, they can also preliminarily evaluate whether or not a given topic number fits the data – and, therefore, creates a good model. Several systematical and manual methods to find the scores are available in various implementations [20]. By creating topic models with a certain set of configurations and a variation on number of topics repeatedly, a range of number of topics that gives most appropriate scores would be implied as the optimal parameter for actual topic model that would be created later.

R library called "ldatuning" is used to identify an acting optimal number of topics for both subjects. By repeatedly creating the models with different number of topics, Fig. 2. shows that LDA seems to generate the "acceptable" models for CUW and MSW when defining number of topics in the range of 16-24 topics and 36-52 topics, respectively.

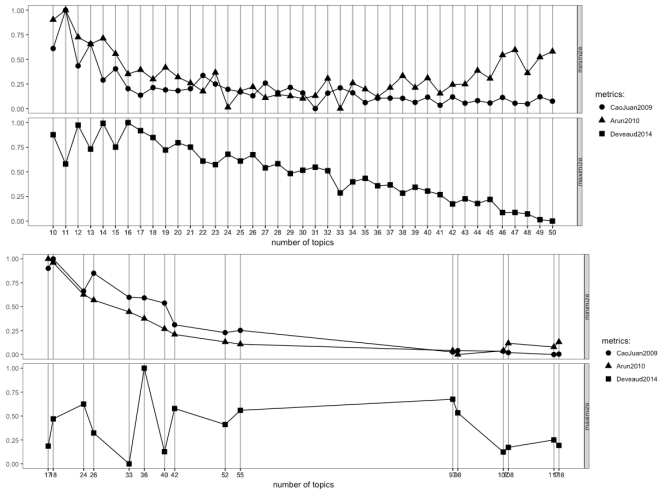

Fig. 2. Topic number searching for Gibbs LDA (CUW and MSW)

By using presumably optimal numbers of topics from previous step, topic models are created. Some word-topic lists with high interpretability level from both codebases are shown in Table IV. Topic 9 and Topic 15 of CUW has some keywords that describe system interfacing between software and hardware such as EGA (a type of computer monitor) and Printer; while Topic 11 shows that "thai_words" are used more frequently in program's user interface section than in developer's comment section. Unique names of Thai Characters in Topic 16 specifically show that a part of source code had been dedicated to development that would enable Thai language for user.

TABLE IV. SAMPLE WORD-TOPIC LISTS (CUW AND MSW)

CUW				MSW			
Topic 9	***Topic 11***	***Topic 15***	***Topic 16***	***Topic 8***	***Topic 14***	***Topic 21***	***Topic 34***
include	thai_ words	int	returnte mpindx	assert	endif	hwnd	struct
suttipong	dispstrhgc	extern	tempindx	owner peterj	macro	null	char
null	yes	char	cut	return	else	return	unsigned
mode	center factor	printerlin efeedinch	tempind xm	struct	ifndef	message	uns
scrmode	thai_ wordsc	left	returntem pindxm	extern	ifdef	window	long
kanakak orn	curmenu	boolean	right margin	char	error	dcl	used
inch	key	bufi	tempindxp	include	can	owner chic	sizeof
function	else	line	check	null	save	else	sttb
ega	row	number	static	bool	set	handle	table
file	lev	right	switch	hnil	state	bool	size
argv	ebioskey	cpi	end	else	default	struct	short

CUW				MSW			
Topic 9	***Topic 11***	***Topic 15***	***Topic 16***	***Topic 8***	***Topic 14***	***Topic 21***	***Topic 34***
stderr	char	esc	returnfail	rgw	used	long	block
printer	savepic	file	vowel	must	must	wparam	number
use	thai_ wordsdc	print	norenoo	sizeof	use	hnil	new
found	retpic	suttipong	returnte mpindxp	default	special	windows	code
page break	filename	area	may	check	user	dont	heap
set	line	inch	rorereo	ftrue	message	1	must
written	attr	kanakak orn	horheeb	user	table	get	use
graphic	int	tue	minus	export	document	mode	typedef
stdcode	thai_ wordse	page	soreseo	can	first	returns	following

Topic modeling also shows probability that each topic may appear in each document. Curator can create visualization of topic's probability distribution across the source code files.

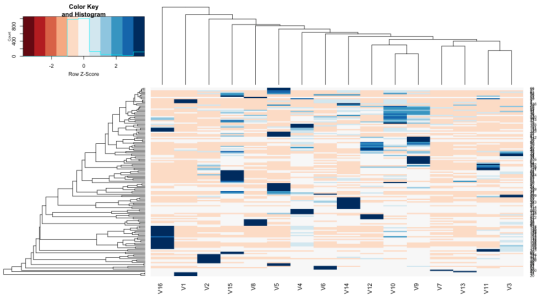

Fig. 3. CUW Document-topic probability distribution

Even though word frequency analysis has revealed the appearance of CUW developer's names in previous step, it is unable to confirm such practice in MSW. Instead, topic modeling discovers that some MSW developers also left their presence in the code. Topic number 1, 3, 8, 17, 21, 26 and 35 has word "owner" following by a developer's name, which means that topic modeling was able to recognize relationship between a portion of source code and keywords that frequently appear together, in this case—the names of software developers. However, it cannot guarantee that every occurrence will be noticed due to weight of the keywords in entire input data. This issue is related to technical tuning of topic modeling, which is not in the scope of this study.

Digital collection curator takes a responsibility to interpret the detonation of each topic and to summarize the insights of the collection from lists that topic model has produced. For example, appearance of developer's name on specific topic might identify the programming effort that developer had contributed. However, the systematic method to confirm such observation does not exist. The interpretation procedure is essentially state-of-the-art that requires professional experience of curator. This study does not discuss collection interpretation.

I. Source Code Flow Analysis

After going through the first phase of analysis, a limitation of LDA on text documents is noticed. The technique may initially fit for human-readable document such as book or novel, but it may not be the case for source code which is machine-readable document; especially source code that lacks immediate documentation within the content. Therefore, an additional step is required.

Source code analysis tools are available in many forms, and for many objectives. Tools such as GCC, CBMB, Flawfinder, and Cppcheck [21] can create a bird-eye-view perspective for codebase via static analysis process. Alternatively, Sourcetrail [15] would index raw source code files and create flow diagrams between individual modules. For two subjects of this study, following diagrams are generated.

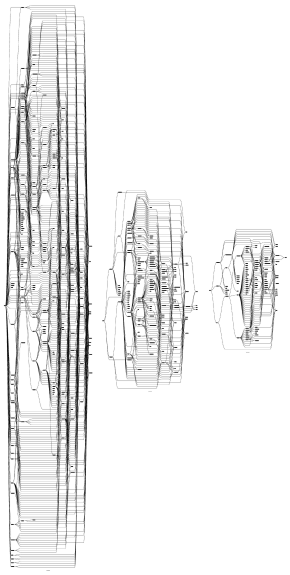

Fig. 4. CUW, CUP, and CUF Function flow diagrams

CUW has three mutually exclusive modules that share common libraries. Therefore, the flow diagram consists of three separate functional flows. Each flow starts from its own main function on the left side, then travels to the leaf functions on the right side through each level of function calls.

Sourcetrail allows user to interactively browse through function flow diagram. Therefore, insights can be obtained in multiple ways, for example, as shown in Fig. 4., it is noticeable that CUW and CUF have the deepest non-recursive function calls at 6 levels (left to right), while CUP has 9 levels. Deep function call in a software development convinces that the software is more inclined toward stack-memory-intensive behaviors. Every new function call would be added into "call stack", in order to be traced back to the origin. Information from this type of investigation could be addressed as a technical feature to compare a software with other softwares of similar class.

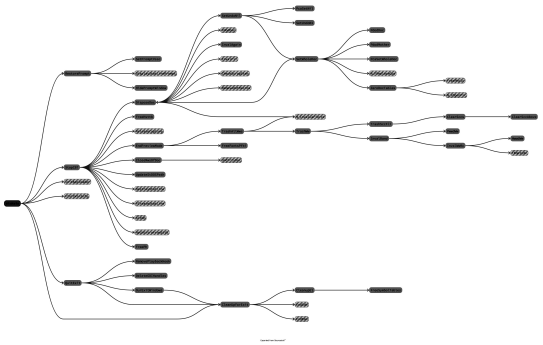

Fig. 5. Partial MSW function flow diagram

In contrast to CUW, Fig. 5 shows unusually tidy function flow of MSW. This is because the diagram is a reduced form of entire flow, due to the fact that MSW does not have main function as other traditional C console applications. MSW version 1.1a is a GUI application which has inconsistent programming syntax and function structure compare to normal C. Therefore, Sourcetrail could not properly index files with a large portion of GUI functions by its default settings. Some edits are made to "WinMain" function in source code to assist indexing process. Only C console application functions, which are shown in the diagram, have made the diagram significantly smaller than the actual complete flow diagram.

Fig. 6. Partial MSW File dependency diagram

J. Notable Remarks in Museum's Context

As mentioned previously, a complete interpretation of topic model is not in the scope of this study. However, some sample remarks have been found in the outputs of textual analysis and manual analysis. Remarks are listed here as the examples of elementary outcomes that supplies facts for further collection interpretation.

1) The word-frequency table is a productive output to consider. Some basic investigations could lead to conceptual questions for further analysis; such as "Did hardware in early 1990s normally require low-level interactions from software via Assembly code?" (A significant amount of words that appear in the tables are Assembly code) or "How did developers implement Thai language for personal computer in early 1990s?" ("thai_words" label appears in CUW source code more than 300 times).

2) Topic models of CUW and MSW might have illustrated differences between commercial-based enterprise software and non-commercial education software. To some extent, topic words of CUW are more convenient to interpret than topic words of MSW. The cause of this issue might be a few aspects, including 1) MSW developers used a framework that was initiated in restricted environment for business purposes—thus, there is no limitations for complexity, unlike CUW where the convenience of development might have been one of major concerns; and 2) MSW may have a constituted coding convention for team of multiple developers, but CUW may not have such direction since there were only few developers working on the project at time. Both assumptions contribute to the result that each topic words in MSW topic model are more homogenous, systematic, and difficult to understand by people outside the project.

3) Evidence that source code analysis could discover historical remark for social dimension that was preserved in digital collection—incapability of Sourcetrail to index Microsoft Windows GUI application leads to investigation on file dependency. Preprocessor condition, which basically determines whether to include some statement in output program before compilation time, with an unusual name is found as shown in Fig. 6. An identifier call "COMING_SOON_TO_A_WORD_PROCESSOR_-NEAR_YOU" was declared to trigger an event when user pressed certain buttons on the keyboard, but it was "commented out" by programmer due to the insufficient information from another team. The name of this identifier is clearly a funny imitation of the tagline "Coming soon to a theater near you" which was widely used in film commercials during the period that MSW was developed.

These remarks are also displaying the expectable outcomes of the process (RQ2).

IV. Discussion and Recommendation

A. Viability of topic modeling as elementary tool for digital collection analysis (RQ1)

Topic modeling is the main technique in this study. It has demonstrated the ability to shorten time-consuming process for digital collection curator to manually read the vast content of source code files. Even though the process to setup a new pipeline for each codebase may take some time to master, it should still be shorter and more robust than relying entirely on human labor. Given that digital collection topic modeling process is an accumulative experience for curator, it might be able to conclude that analysis pipeline setup time would also be shorter for experienced curator as time goes by.

However, in terms of accuracy for collection analysis, topic modeling may not necessarily be the best option. This issue has led to the introduction of manual analysis tool that is not only focusing on the performance of source code, but also allows curator to have a walkthrough of the codebase.

B. Expectable outcomes from the proposed process (RQ2)

As described in RQ1, digital collection analysis with topic modeling could have shortened time-consuming process for curator. It should give a perspective that links to deeper analysis in specific area of the source code.

Similar to working with historical artifacts, digital collection like codebase is very sensitive and subject-matter process. Having a set of hypothesis questions in mind before analyzing a digital collection should help curator to achieve the

result faster than roaming around aimlessly. For example, "Does the version of source code that was created during early development of C language contain pointer variables more often than the recent versions?" or "How much does the programmer regularly comment in Assembly modules compare to C modules?" As a result of this exploratory study, the following framework is proposed as a general reference for investigating software codebase, in order to extract necessary knowledge and information to fulfill the requirements for digital collection preservation or for making an exhibition based on such kind of collection.

1) Unsupervised exploratory topic modeling — Human expertise has always been major resource for collection analysis. Unsupervised text mining exploratory for digital collection should eliminate amount of time that curator need to acclimatize themselves to new subject. However, equipment and framework selection is nonetheless critical decision in this aspect. Even though the entire concept is advantageous, the passage to obtain proper topic model for next steps can be a hinder instead of a help. It should be noted that most new off-the-shelf tools have offered a satisfying level of convenience that could help curator get beyond the complexity of statistics and programming literacy.

2) Cherry picking — Once unsupervised exploratory is done, human judgment must take the lead. Upon topic modeling process, curator must browse and choose topics that are relevant to value of the subject. For example, as presented in remark section, if curator is interested in social dimensions of historical source code, they can take "COMING_SOON_TO_A_-WORD_PROCESSOR_NEAR_YOU" as starting point to look further into the content to see whether codebase has wrapped other social matters from their age in the digital collection time-capsule.

3) Manual investigation and flow analysis – Selected topics may require further attention to extract detail information. Manual investigation and flow analysis with tool such as Sourcetrail helps curator to look closely at source code and to have a walkthrough of the program control flows, either from function perspective or file-structure perspective.

4) Collection evaluation – Results from step 2) and 3) can be reconciled in this step. Curator can evaluate the accuracy of topic model against the flow diagrams and internal program dependencies discovered in manual investigation, and vice versa. At this point, curator should have a complete picture of source code without having to spend time reading every line of code. Additionally, they should be able to appreciate the historical value of software as a digital collection.

5) Represent the findings – Another aspect that curator must be concerned is the responsibility of museum on digital collection. Curator should be able to draw a virtual line between information that is supposed to be preserved with the collection and the information that should be developed into an exhibition. For example, from two subjects in this study, various flow diagrams from Sourcetrail are highly potential to be exhibited. Even visitors without programming experience can appreciate the intricacy behind software development process visualized by the diagrams. Additionally, insights from topic modeling can be turned into good exhibition content with proper storytelling technique. It can also be combined with information from people who were involved in software development, which can be obtained via direct interview and personal conversation. On the other hand, information such as level of function call and number of topics in certain codebase can be used as one of metadata features for the digital collection preservation purpose.

C. *Threats of validity*

- Nature of historical codebase – archived source code are often unorganized. While modern codebases are more systematically collected with the help of code revision frameworks such as GIT and Subversion. Generally, codebases in early programming period have, more often than not, scattered in storage without revision history and document. Single revision of source code as the input data to unsupervised analysis process is potentially pushing the analysis at the risk of underfitting topic model. It is because other source code revisions are not available for validation. Even though this study has no solution to mitigate the issue, the future of this situation is brighter than in the past. Once current well-documented codebases have eventually been brought to museum as next generation of digital collections [22] this issue would be gradually eliminated.

- Oversimplification of process – while process simplification is a goal of this study, weighting between creating simple analysis process and accurate result is still an ongoing issue. As mentioned in discussion section, readily available tools may help curator to get beyond the barrier of statistical and technical knowledge; and be able to extract the insights from source code. It is the low-hanging fruit of digital collection examination. From the blind side, those tools can also prevent user from actually understanding behind-the-scene mechanism of the algorithms which is extremely necessary for further advanced levels of analysis process. Trial-and-error procedure is practical enough to get the work done, but it could also be seen as a trade-off for sustainability in the long run.

V. FUTURE WORK

1) According to the limited subjects used in this study, the historical progress examination by comparing each revision of the source code of the same software [23][24] is yet to be achieved. This study can be expanded in the way that multiple versions of the same software are included and analyzed. Determining development trends and progresses along various versions of individual software should give more proper benchmarking matrices than the comparison between different software products.

2) LDA is a handy technique for topic modeling, but it does not always provide the best performance. It is noted that LDA topic modeling process in this study can take over ten hours on a personal computer to select an optimal topic numbers and create topic models. Some studies suggest technical feasibility to adjust parameters and algorithms in LDA [25] to reduce the time for searching optimal topic numbers, and creating model. In addition to increasing final accuracy, if time is sufficient, other technique such as Latent Semantic Analysis (LSA) may provide alternate comprehensions or the "second opinion" which adds up to the LDA interpretation of the same codebase [26]. Combination of LDA and LSA is also recommended for the better analysis result [27].

VI. Conclusion

This study had explored the possibility to create a framework to examine source code, while recognizing source code as a special set of digital collection. It proposed the investigation steps that involved two technical tools; Latent Dirichlet Analysis (LDA) and Source Code Flow Analysis to achieve the two goals: 1) contrasting digital collection analysis from typical software analysis and 2) creating and simplifying the digital collection analysis pipeline in museum context.

After the experiment with selected subjects, it was summarized that the recommended pipeline process includes 1) use unsupervised topic modeling to explore the input data, 2) manually select interesting issues from the topic models for further investigation, 3) observe alternate perspectives of codebase with flow diagrams, 4) reconcile the results from previous steps and valuate the codebase collection, and 5) decide over what part of the results should be preserved with the collection—and what should be made into exhibition.

The study also discussed the future area to expand the work into, including 1) exploratory framework should also suit codebase that consists of multiple versions of a software and 2) substitution or combination of other topic modeling techniques may increase overall performance and accuracy of the process.

Acknowledgment

The author would like to thank owners of all open public source code repositories that provide access to historical codebases used during the course of this study, whether being mentioned in this paper or not. Also, thanks to my colleagues at National Science Museum, Thailand, including external reviewers, who voluntarily offered valuable recommendations from museum professionals' perspective and IT professionals' point of view.

References

[1] G. C. Murphy, A. Lai, R. J. Walker, and M. P. Robillard, "Separating features in source code: an exploratory study," 2001, pp. 275–284.

[2] A. Dreweke, I. Fischer, T. Werth, and M. Wörlein, "Text Mining in Program Code," in *Handbook of Research on Text and Web Mining Technologies*, IGI Global, 2009, pp. 626–645.

[3] S. J. M. M. Alberti, "Why Collect Science?," *Journal of Conservation and Museum Studies*, vol. 15, no. 1, Dec. 2017.

[4] S. Chan, "Collecting the present: digital code and collections," *MW2014: Museums and the Web 2014.* [Online]. Available: https://mw2014.museumsandtheweb.com/paper/collecting-the-present-digital-code-and-collections/. [Accessed: 10-May-2018].

[5] E. Linstead, L. Hughes, C. Lopes, and P. Baldi, *Software Analysis with Unsupervised Topic Models*. 2018

[6] R. Witte, Q. Li, Y. Zhang, and J. Rilling, "Text mining and software engineering: an integrated source code and document analysis approach," *IET Software*, vol. 2, no. 1, p. 3, 2008.

[7] A. Møller and M. I. Schwartzbach, *Static program analysis.*

[8] A. M. Saeidi, J. Hage, R. Khadka, and S. Jansen, "ITMViz: Interactive Topic Modeling for Source Code Analysis," 2015, pp. 295–298.

[9] J. Baldwin, D. Myers, M.-A. Storey, and Y. Coady, "Assembly Code Visualization and Analysis: An Old Dog CAN Learn New Tricks!"

[10] M. Ogawa and K.-L. Ma, "code_swarm: A design study in organic software visualization," *IEEE Transactions on Visualization and Computer Graphics*, vol. 15, no. 6, pp. 1097–1104, 2009.

[11] M. Pinzger, K. Grafenhain, P. Knab, and H. C. Gall, "A Tool for Visual Understanding of Source Code Dependencies," 2008, pp. 254–259.

[12] R. Alghamdi and K. Alfalqi, "A survey of topic modeling in text mining," 2015.

[13] A. Kuhn, S. Ducasse, and T. Gîrba, "Semantic clustering: Identifying topics in source code," *Information and Software Technology*, vol. 49, no. 3, pp. 230–243, Mar. 2007.

[14] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Mar. 2003.

[15] *Sourcetrail*. Coati Software KG, 2018.

[16] L. Shustek, "Microsoft Word for Windows Version 1.1a Source Code | Computer History Museum", *Computerhistory.org*, 2018. [Online]. Available: http://www.computerhistory.org/atchm/microsoft-word-for-windows-1-1a-source-code/. [Accessed: 10- May- 2018].

[17] "Standard for Thai Character Codes for Computers", *Nectec.or.th*, 1990. [Online].Available:https://www.nectec.or.th/it-standards/std620/std620.html. [Accessed: 10- May- 2018].

[18] T. Pawelka and E. Juergens, "Is this code written in English? A study of the natural language of comments and identifiers in practice," in *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, 2015, pp. 401–410.

[19] H. M. Wallach, I. Murray, R. Salakhutdinov, and D. Mimno, "Evaluation methods for topic models," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 1105–1112.

[20] M. Ponweiser, "Latent Dirichlet allocation in R," 2012.

[21] L. Torri, G. Fachini, L. Steinfeld, V. Camara, L. Carro, and É. Cota, "An evaluation of free/open source static analysis tools applied to embedded software," in *Test Workshop (LATW), 2010 11th Latin American*, 2010, pp. 1–6.

[22] R. Di Cosmo and S. Zacchiroli, "Software Heritage: Why and How to Preserve Software Source Code," in *iPRES 2017: 14th International Conference on Digital Preservation*, 2017.

[23] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Studying software evolution using topic models," *Science of Computer Programming*, vol. 80, pp. 457–479, Feb. 2014.

[24] S. L. Abebe, S. Haiduc, A. Marcus, P. Tonella, and G. Antoniol, "Analyzing the Evolution of the Source Code Vocabulary," 2009, pp. 189–198.

[25] A. Asuncion, M. Welling, P. Smyth, and Y. W. Teh, "On smoothing and inference for topic models," in *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence*, 2009, pp. 27–34.

[26] J. I. Maletic and A. Marcus, "Using latent semantic analysis to identify similarities in source code to support program understanding," 2000, pp. 46–53.

[27] T. Williams and J. Betak, "A Comparison of LSA and LDA for the Analysis of Railroad Accident Text," *Procedia Computer Science*, vol. 130, pp. 98–102, 2018.