Recovering a Balanced Overview of Topics in a Software Domain

Matthew B. Kelly, Jason S. Alexander, Bram Adams, Ahmed E. Hassan
School of Computing
Queen's University
Kingston, ON, Canada
{matthew, jason, bram, ahmed}@cs.queensu.ca

***Abstract*—Domain analysis is a crucial step in the development of product lines and software reuse in general, in which domain experts try to identify the commonalities and variability between different products of a particular domain. This identification is challenging, since it requires significant manual analysis of requirements, design documents, and source code. In order to support domain analysts, this paper proposes to use topic modeling techniques to automatically identify common and unique concepts (topics) from the source code of different software products in a domain. An empirical case study of 19 projects, spread across the domains of web browsers and operating systems (totaling over 39 MLOC), shows that our approach is able to identify commonalities and variabilities at different levels of granularity (sub-domain and domain). In addition, we show how the commonalities are evenly spread across all projects of the domain.**

***Keywords*-domain analysis, topic modeling, empirical study**

I. Introduction

Domain analysis is the "process by which information used in developing software systems is identified, captured, and organized with the purpose of making it reusable when creating new systems" [29]. Essentially, domain analysis reveals the commonalities and variability between systems in the domain under study. For example, in the domain of text editors, major commonalities are the ability to edit and print text, whereas the range of supported file formats is a major point of variability. The ultimate goal of domain analysis is to reuse commonalities across products, for example as the basis of a software product family [18], [24], [26], [27] or of a reusable library [7], whereas variability typically results into value-adding features.

Despite the importance of domain analysis, it is mostly a manual, labour-intensive task that requires significant domain expertise [13], [29]. To distill the main domain concepts, domain experts consult any available data source, such as requirements, design documents and the source code of existing or competing products (if available) [11]. They then try to reconcile the distilled concepts amongst each other by building a unified domain model. Achieving a consensus between all team members takes dozens of meetings and several months depending on the scope of the project [2], [17], [18].

Although tool support exists for domain analysis, it either aims (1) at later stages of the analysis [11], [14], [18], when analysts want to organize the identified concepts and their constraints, or (2) at abstracting concepts from one particular system [9], [32], [39]. Identifying and finding commonalities or variability across multiple systems in a domain is still work in progress. Tools have been proposed to automatically categorize software systems based on their documentation [20], [36] or source code identifiers [3], [16], [35]. Such approaches seem promising for commonality analysis, yet they either are not targeted towards source code, or tend to bias their categorization towards the larger software projects.

We propose an approach to compare common domain concepts from the source code of multiple software projects in a balanced way, using topic modeling. Topics are collections of words that co-occur frequently in a corpus of documents (in our case: source code files), and hence are likely related to the same semantic concept. We apply topic modeling on each software project separately, then cluster the resulting topics across all projects in a particular domain. Clusters shared by multiple projects contain common concepts, whereas clusters belonging to only one or two projects correspond to variable concepts. Domain analysts can use the output of our approach to bootstrap their manual analysis.

This paper makes two primary contributions:

- An automatic approach to apply topic models on several corpora of varying sizes to identify commonalities and variability across different systems in a domain.
- A case study of 19 systems across two domains, totaling over 39 MLOC, showing that our approach provides a balanced overview of the concepts in a (sub-)domain.

The remainder of this paper is organized as follows. Section II motivates our work and discusses related work. Section III outlines our approach. Section IV presents our case study results, followed by a comparison of our balanced approach to an unbalanced topic mining approach in Section V. Section VI identifies the threats to the validity of our research, and Section VII concludes the paper.

II. Background and Related Work

This section motivates our work, provides background on the topic modeling technique that we use, and discusses related work.

A. Domain Analysis

In the context of software product families [18], [24], [26], [27], [29], domain analysis is the first step in a process called "domain engineering", which tries to build and document a generic architecture for a particular domain based on an extensive set of data available about that domain. The generic architecture is said to capture the "commonalities" of the modeled domain. Later "application engineering" processes then reuse this architecture to produce custom software products that add value by providing elements of variability. Reuse of commonalities across a product family (theoretically) enables companies to produce better products in a shorter time frame at a fraction of the cost, similar to assembly lines of car manufacturers [10].

Various methodologies exist for domain analysis, all of which essentially specify the process of how to identify and capture the commonalities and variability in a domain [4], [11], [14], [15], [32], [38]. This process primarily is a human activity, involving the detailed review and comparison of requirements specifications, design documents, source code, manuals, contracts, personal communication and any other available source of data [13], [29]. Given the scale and depth of the considered domains, significant personal interaction and management skills are required to arrive at a compromise between all domain analysts, often after months of meetings and discussions [2], [17], [18].

Despite the age of the field, scalable tool support for domain analysis is still an open issue, especially for analyzing commonalities and variability within existing software systems, typically developed by competitors [10], [13], [18]. The DARE methodology [11] provides partial automation using lexical techniques to identify important phrases and words in the domain documents, and to cluster these phrases and words based on their conceptual similarity. More recent incarnations of DARE feature source code analysis tools like cflow and source navigator, yet the source code analysis part of DARE is still the most time-consuming domain analysis step [39]. Other methodologies use tools for code instrumentation [9], [32], state machine extraction [14] or management of UML models [18].

Software architecture recovery and concern mining (also referred to as feature/concept location [28] or aspect mining [22]) are two areas related to commonality and variability analysis, but targeted more at supporting software maintenance and program understanding. Software architecture recovery extracts a high-level concrete architecture from the source code of a software system and compares it to a reference architecture (e.g., [23]). Concern mining (e.g., [30]) identifies code regions that together form a conceptual unit, for example the "undo" feature of a text editor or all the code responsible for reliability. Software architecture recovery and concern mining are necessary techniques for domain analysis, yet require additional analysis on top of them to compare the recovered architecture and concerns of dozens of systems to each other. Such an analysis is the subject of this paper.

B. Topic Modeling

This paper proposes and evaluates an automatic technique, based on topic modeling, to bootstrap commonality analysis. Topic modeling techniques like LDA [5] (Latent Dirichlet Allocation) analyze a document or set of documents (a *corpus*) to produce a probabilistic model of word clusters (*topics*) that group semantically related words. This clustering is typically computed based on the co-occurrence of words in each document. In addition to the topics, i.e., groups of words, topic modeling also yields for each document the memberships of each topic, i.e., the percentage of the functionality of the document described by the topic.

Topic modeling traditionally has been used for the analysis of written language, and has been shown to increase understanding of a corpus by generating a semantical overview. Such overviews have proven useful for indexing, classifying, and characterizing documents [25]. The last decade, topic modeling techniques have been gaining traction in the field of software engineering research, for example to find the underlying architecture of a software system [37], to locate software features [28], and to analyze the evolution of a software system [33], [34].

The most closely related application of topic modeling to this paper is the automatic categorization of software systems as being for example a "development tool" or "implementing hashmap functionality" [3], [16], [20], [35], [36]. These techniques essentially combine topic modeling with clustering of the topics to identify the dominating commonalities (main categories) between up to thousands of software systems. In other words, these techniques only focus on a small subset of all common topics, and disregard most of the variability between topics. In contrast, domain analysts need to consider *all* commonalities and variability across a set of software systems.

III. Topic Model-based Domain Analysis

This section describes our topic model-based approach for domain analysis. This approach enables domain analysts to study the commonalities and differences across different groups of applications, at various levels of granularity. Figure 1 provides an overview of the approach.

Our approach first pre-processes the source code of the analyzed software systems to extract meaningful words. Each project's extracted words are then analyzed by a separate LDA process, followed by post-processing of the resulting topics to identify the topics' most frequently occurring words. Finally, the topics are clustered at varying levels of granularity (subdomain or domain). The resulting clusters can be analyzed by domain experts to drive their detailed manual analysis. The following subsections discuss each step of our approach in detail.

A. Source Code Pre-processing

Prior to performing LDA topic analysis on a particular version of a project, the source code must first be preprocessed. Traditionally, unwanted words and punctuation are removed to eliminate frequently occurring words that contain no semantic meaning with respect to the code's functionality [19], [37]. This removal comprises the following steps:

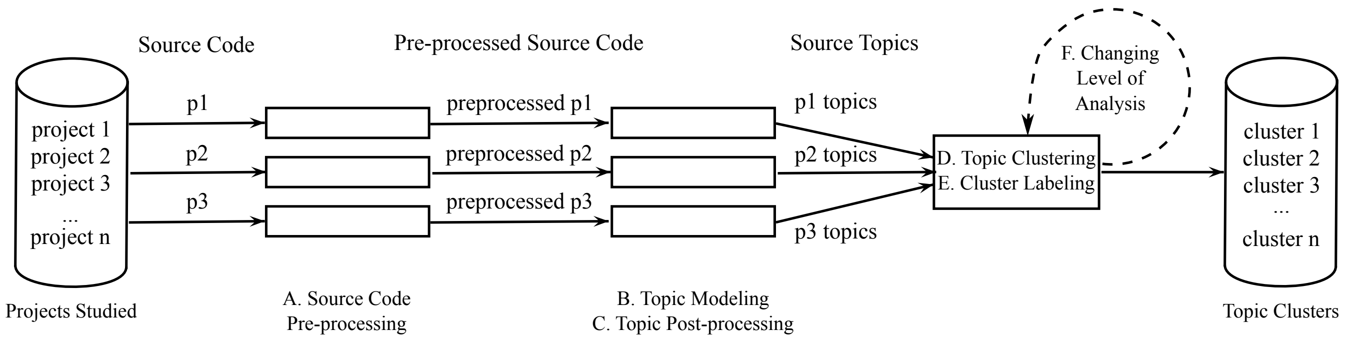

Fig. 1. An overview of our approach.

1) **Removal of language-specific words.** This step is the only programming language-specific step of our approach. Although the study in this paper focuses on C and C++ software systems, supporting a different language only requires compiling a different list of reserved words. Examples of the C/C++ reserved words that we removed are `template`, `typename`, `class`, `friend`, `private` and `this`.
2) **Removal of comments.** Similar to the software categorization work [3], [16], this paper only uses source code as input data. It is well-known that, contrary to code comments, source code is quite a low-level artifact [24], [29], yet many software systems are only poorly commented or have outdated comments. Hence, in this paper we consider the worst case scenario that no comments are available in order to obtain a lower bound on the performance of our approach. In future work, we plan to consider higher-level artifacts like source code comments and design documents.
3) **Breakdown/splitting of identifiers.** For example, `myVariableName` would be split into `my`, `variable`, and `name`.

B. Topic Modeling

Prior work has shown that topic modeling techniques like LDA are biased towards corpora with a larger size [34]. In our case, this would mean that the resulting topics are primarily derived from the larger projects, since they dominate the documents of smaller projects in sheer size. In fact, all software categorization techniques that we discussed in Section II-B apply LDA or a similar topic modeling technique to all software projects at once. This makes sense in their context, since software categories correspond by definition to the dominant groups of commonalities between projects. In the remainder of this paper, we refer to these approaches as "unbalanced" approaches.

In the context of our work, unbalanced approaches make sense for prioritizing topics (e.g., to decide which commonalities to analyze first), but not for an initial, unbiased analysis of commonalities and differences across all projects. For this reason, we use a "balanced" approach that first applies the LDA topic analysis technique to each project's pre-processed data independently, followed by a global clustering across the extracted topics of all projects. This clustering treats the topics of all projects as equal, regardless of project size. This allows us to identify common topics of smaller projects, or even crucial differences between larger and smaller products in the domain under study.

For our LDA analysis, we use the open source Mallet tool [21]. Mallet is run with an optimizing interval of 10 to find the parameters that maximize the log-likelihood function of the dataset under analysis. We configured Mallet to produce a set of 20 topic clusters with 20 words per cluster, but this is by no means the only possible configuration.

Two separate output files are generated, i.e., one file with the list of words belonging to each topic and one file with topic proportions (topic memberships) per source code file. For example, the first file of the TinyOS subject system contains:

```
  ...
3 addr ip sock tcp tcplib tun frag udp ... udp
4 avr page ds addr bits poll disk ... lock
5 msg net deluge addr id data dhv dip ... packet
...
18 sim cc current log channel time ... link
```

In each topic, the words are ranked from most to least important, starting at the left side of each cluster. The second file maps the files to the topics that they describe. It contains the topic memberships per source code file. For example, in the TinyOS system:

```
  ...
/tinyos/packet.h A 0.75 B 0.23 C 0.02 ... 0 0.0
/tinyos/HplAt45db.h X 0.63 Y 0.37 Z 0.0 ... 0 0.0
...
```

For *packet.h*, the primary topic is topic A with a 75% membership. The second most important topic for this file is topic B with a 23% membership. For *HplAt45db.h*, the most important topic is topic X with a 63% membership. The second most important topic for *HplAt45db.h* is topic Y at 37%.

C. Topic Post-processing

After the LDA analysis, the topic clusters need to be pruned. We find that in most cases only the top 5 words contribute significantly to each topic (based on the topic membership percentages). To reduce the complexity of our approach and to make the next step of our approach easier, only the top 5 words are selected from the original 20. We then assign weights to the top 5 words, with the highest weight awarded to the most significant word, and decreasing weight as the importance of the words declines.

D. Topic Clustering

To perform the actual analysis of commonalities and variability between the different software projects, we now cluster all project-specific topics together using the standard K-means clustering algorithm. K-means is a robust, iterative clustering algorithm that tries to find the natural K cluster centres in a group of n topics by minimizing the distance between each topic to the centre of the cluster it belongs to.

We first need to convert the rankings of the 5 words in each topic into vectors of dimension U, with U the number of unique words across all topics considered during the clustering. Then, we use the open source Weka tool [12] to cluster the topics based on the distance between the vectors. The actual value of K depends on the desired granularity in the study. Increasing the number of clusters sacrifices topic cohesion for granularity and vice versa. We choose 20 clusters, since that is the number traditionally used by topic modeling techniques.

We define the "spread" of a topic cluster as the percentage of all considered software systems that cover the topic cluster. This percentage indicates the degree of commonality between the projects for a particular topic cluster. High spread indicates that a particular topic cluster is common throughout a domain. Low spread indicates variability.

E. Cluster Labeling

To clarify the semantics of the clusters, we synthesize a textual label that summarizes all the words in a cluster. For this, we rank the words of each cluster by summing the ranks of the words in each of the cluster's constituent topics. In cases where succinct labels cannot be determined based on the top words, online project documentation was used by one of the authors to check the validity of the corresponding cluster, and, if valid, to assign a more meaningful label. In practice, interpretation of the topic labels is also the first task domain analysts perform.

For example, for one of the operating systems that we studied (FreeDOS), we obtained the label "`pddt, ddt, drive, bpb`". These words appear to be local variables or function arguments, without relevance to the larger scope of the project. However, the project documentation revealed that these words represent the FreeDOS initial kernel disk, yielding a more meaningful label for the topic cluster.

F. Changing Level of Analysis

Up until now, we did not specify the sets of projects for which we generate and cluster topics. As a matter of fact, comparisons can be made at different levels of granularity, ranging from "all text editors that are a clone of GNU Emacs", over "all console-based text editors" to "all text editors". We refer to the highest level of granularity in a particular context as the "domain level", and to the other levels as a "sub-domain level". One could even compare systems across domains [3], [16], [20], [35], [36], although project-specific naming conventions potentially reduce the effectiveness of LDA at this level [20].

We typically follow a bottom-up approach, starting with for example a sub-domain level where we cluster the projects' topics and assign labels to each cluster. To proceed to the domain level, we repeat the analysis with the topics of all systems in all considered sub-domains. This approach is repeated all the way up to coarser-grained levels.

At each level, the resulting labeled clusters and their mapping to individual projects' topics (and source code) enables domain experts to bootstrap their analysis of commonalities and variability of products in a particular (sub-)domain. Since topic clusters group multiple topics of multiple projects, the experts can proceed by manually investigating the major topics in each cluster, maybe deciding to repeat our approach with other thresholds to obtain more or finer-grained topics.

IV. Case Study

We performed an exploratory empirical study to qualitatively evaluate our methodology on 19 C/C++ projects in two separate domains. The operating system (OS) domain, illustrated in Figure 2, is divided into three sub-domains: embedded real-time operating systems (RTOS), monolithic kernels and cloud-based operating systems. The only cloud operating system in our study is Google's Chromium OS, due to a lack of available candidates. The web browsing domain is divided into two sub-domains: GUI-based and text-based browsers.

Table I documents all of the software systems investigated in our case study. For each system, it lists the project name, the analyzed version of the software, the domain of the system, the number of C/C++ files parsed, the number of analyzed C/C++ source lines of code (SLOC, calculated using SLOCCount [8]), as well as the number of tokens extracted from each system. The systems cover a wide range of sizes. Although most of the operating systems are older than the web browsers, the Chromium and Firefox browsers have a similar size as some of the larger operating systems.

Our study performs our methodology at the sub-domain and domain levels. We show the corresponding topic clusters at each level, together with the following attributes:

1) The topic cluster's label.
2) The spread of the topic cluster.
3) The cluster size, i.e., the percentage of all topics in the investigated (sub-)domain that belong to the topic cluster. This will be used as a measure of the size or importance of each cluster. A high cluster size indicates a topic cluster with many topics.
4) An ordered list of the top words in each topic cluster.

Some clusters contain no meaningful information, only arbitrary code fragments. For example, the "`eric, progname, perror, ...`" cluster collected from Ultrix (monolithic

TABLE I
SOURCE CODE ATTRIBUTES OF THE SOFTWARE SYSTEMS USED IN THE EMPIRICAL STUDY.

System	Version	Sub-Domain	C/C++ Files	C/C++ SLOC	Parsed Tokens
FreeRTOS	6.1.0	Embedded RTOS	3,075	615,456	898,910
TinyOS	2.1.1	Embedded RTOS	649	59,995	85,794
Inferno	3e	Embedded RTOS	1,654	479,193	495,216
eCos	3.0	Embedded RTOS	4,141	945,371	2,286,171
FreeBSD	8.1	Monolithic OS	24,726	7,960,207	20,475,256
OpenBSD	4.8	Monolithic OS	16,840	4,047,191	8,567,978
Minix	3.1.1	Monolithic OS	431	66,930	184,846
Unix	v7	Monolithic OS	991	137,034	183,424
Linux	2.6.0	Monolithic OS	12,424	3,805,097	9,498,413
Haiku OS	R1/Alpha2	Monolithic OS	24,734	5,475,559	12,643,109
FreeDOS	1.1	Monolithic OS	5,893	411,850	146,795
OpenSolaris	2009-06	Monolithic OS	22,474	8,268,300	18,527,066
Ultrix	11	Monolithic OS	2,339	421,248	576,047
Chromium OS	r66904	Cloud OS	13,251	2,393,061	6,925,789
Chromium	4.0.222.12	GUI Browser	7,192	1,427,210	4,572,548
Firefox	3.0	GUI Browser	9,428	2,415,448	7,238,830
Konqueror	4.5.2	GUI Browser	224	31,831	115,147
Lynx	2.8.7	Text Browser	217	129,263	296,818
Links	0.9.8	Text Browser	42	23,612	63,945
		Total:	**150,725**	**39,113,856**	**93,782,102**

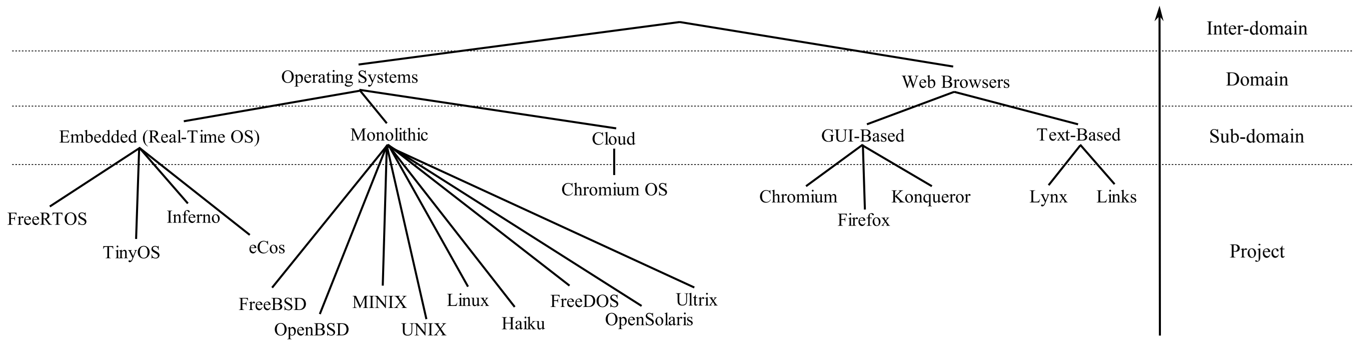

Fig. 2. Hierarchical organization of the operating system and web browser domains in Table I.

OS) contains a developer's name as the top word. Since such implementation-specific clusters tend to disappear when considering a whole (sub-)domain, we did not filter these topics out up-front.

Except for Table II, we only show the top 10 topic clusters at each level to avoid drowning the reader with results, and for each topic we only report the most salient words. The complete results can be found online [1]. We now discuss our results for each level to determine whether our approach works in practice.

A. Sub-domain Level

The sub-domain level consists of systems with similar goals and analogous functionality. The sub-domains investigated for the OS domain are embedded RTOS systems, monolithic operating systems and one cloud operating system. The browser domain covered text-based and GUI-based browsers.

Embedded RTOS

For this first sub-domain, Table II also provides a breakdown of the spread in the RTOS sub-domain, in addition to spread and cluster size. Similar tables were generated for all results (even at the project-level), but were not included because of space restrictions [1].

From the results in Table II, we see that embedded RTOS systems primarily deal with low-level hardware, network communication and timing. Examples of low-level hardware concepts are the *architecture definitions/specifics* and *hardware controller* clusters. Topic clusters *networking primitives*, *micro ip tcp/ip stack*, *ethernet transmission* and *ethernet controller* belong to network communication. Finally, topics *microcontroller interrupts*, *scheduling*, *clock* and *task scheduling* correspond to timing. Note that some clusters, like *architecture definitions* and *architecture specifics* should actually be merged into one.

TABLE II
EMBEDDED REAL-TIME OS AT SUB-DOMAIN LEVEL.

Topic Label	Spread (%)	Cluster Size (%)	eCos	FreeRTOS	Inferno	TinyOS	Top Words
architectural specifics	100	46.3	11	7	4	15	mcf, addr, reg
snmp	50	3.8	0	1	0	2	msg, snmp, addr
flash filesystem	50	2.5	1	0	0	1	gr, rf, jffs
networking primitives	50	2.5	1	1	0	0	ip, tim, addr
hardware controllers	50	2.5	0	1	1	0	avr, big, lcd, res
micro ip tcp/ip stack	50	2.5	0	1	1	0	fp, uip, fs, ip
ethernet transmission	50	2.5	1	1	0	0	eth, rx, tx, phy
security	50	2.5	1	1	0	0	ercd, flag, sadb, key
constants	50	2.5	1	1	0	0	xf, xff, xfe, xd
serial interface	50	2.5	1	0	0	1	serial, packet, cyg
microcontroller interrupts	50	2.5	0	2	0	0	reg, pio, mci
scheduling	50	1.3	0	0	1	0	dp, cp, min
constants	50	1.3	1	0	0	0	xf, xc, xe, xb
plan9 types	25	13.8	0	0	11	0	bp, ftl, tkt, uart
architecture definitions	25	3.8	0	3	0	0	bit, io, byte, mask
error control	25	2.5	2	0	0	0	err, test
inferno vlrt runtime	25	1.3	0	0	1	0	lo, vlong, rv
clock	25	1.3	0	0	0	1	config, tda, mhz
task scheduling	25	1.3	0	1	0	0	task, port, queue
ethernet controller	25	1.3	0	0	1	0	ctlr, ether, port
Total:	**4 systems**	**80 topics**	**20 clusters**	**20 clusters**	**20 clusters**	**20 clusters**	**—**

Table II also shows the results of the topic breakdown. For example, clusters like *flash filesystem* and *serial interface* are shared between eCos and TinyOS, while FreeRTOS and Inferno share the *micro ip tcp/ip stack* cluster. Some clusters map to only one project, such as *Plan 9 types* in Inferno (which is a derivative of the Plan 9 OS), and the *task scheduling* cluster of FreeRTOS.

Half of the clusters are shared between at most half of the projects (median spread of 50%). This is especially due to the first cluster, which combines almost half of the topics across all sub-domain projects. Hence, there are substantial commonalities between the four systems. To identify these, it suffices to examine the topics that are clustered together into the *architecture specifics* cluster. Those topics correspond to basic concepts like *low-level IO* and *memory management*, *inter-process communication* and *DMA handling*.

Monolithic OS

Table III shows the top 10 results for the monolithic OS sub-domain. The results clearly differ from those of the embedded RTOS domain, since the focus shifts towards higher-level *memory management*, *(peripheral) device controller*, and various *filesystem* topic clusters, as well as an emerging topic cluster for *authentication*. These are the components one might expect from a modern monolithic operating system [6].

An important difference between monolithic systems and embedded RTOS is that the topic clusters of various projects overlap less, i.e., this sub-domain has a median spread of 33.3% (for the top 20 clusters) compared to 50% for RTOS projects. In other words, there is more variability between these systems, which is not surprising given that this sub-domain contains more than twice as many software systems. Still, five clusters contain highly common concepts across at least 50% of the systems.

The distribution of cluster size is similar for both sub-domains, i.e., the first cluster (*architectural specifics*, shared by 8 projects) again dominates the other topics, together with the *error control* cluster (17.8% of all topics). The mapping from clusters to the topics of individual projects tells us that the *architectural specifics* cluster comprises *low-level memory management*, *network/serial communication*, *file system inode management*, *SCSI device support* and *inter-process communication*, whereas the *error control* cluster combines various *logging* and *return value handling* topics.

Cloud-Based OS

The last OS sub-domain that we investigate are cloud-based OSes, which only consist of a beta-release of the Chromium OS. Since we only investigate a single system, each cluster is actually one topic, as shown in Table IV.

The Chromium OS results indicate a major shift towards web-based integration with a browser front-end, as is expected from a cloud OS prototype. Topics like *video and audio streaming*, *tabbed browsing*, and *webkit* indicate heavy use of

TABLE III
MONOLITHIC-BASED OS (TOP 10) AT SUB-DOMAIN LEVEL.

Topic Label	Spread (%)	Cluster Size (%)	Top Words
architectural specifics	88.9	40.6	size, port, dev, pci
tcp/ip stack	77.8	8.9	ip, addr, dev, pdp
constants	66.7	3.3	xa, xf, xc, xff, xe
error control	55.6	17.8	error, register, cp
device controller	55.6	2.7	tx, ieee, rx, mac, priv
libdisasm	44.4	5.6	lp, op, reg, insn
peripheral devices	44.4	3.9	acpi, status, device, stp
filesystem directory structure	44.4	2.7	node, path, cmd, entry
filesystem	44.4	2.2	file, entry, size, archive
teletype & digital signal processing	33.3	2.8	tty, dsp, irq
Total:	**9 systems**	**180 topics**	—

TABLE IV
CLOUD OS AT SUB-DOMAIN LEVEL.

Topic Label	Spread (%)	Cluster Size (%)	Top Words
url	100	5	url, id, entry
audio & video streaming	100	5	stream, video, audio
security authentication	100	5	key, ssl, cert
tabbed browsing	100	5	view, tab, browser
filesystem	100	5	file, path, base, dir
browser extension	100	5	extension, service
architecture specifics	100	5	operand, register, reg
http request	100	5	request, cache, http
gui layout	100	5	rect, width, window
webkit	100	5	web, kit, plugin
Total:	**1 system**	**20 topics**	—

web interaction. We also see more traditional OS topics such as the *architecture specifics*, *filesystem*, and *registers* topics.

GUI-Based Browsers

Similar to the findings of the OS sub-domain study, the browser sub-domains expose important commonalities and variability between different projects. First, we studied GUI-based browsers. The results of the GUI-based study are displayed in Table V.

TABLE V
GUI-BASED BROWSERS (TOP 10) AT SUB-DOMAIN LEVEL.

Topic Label	Spread (%)	Cluster Size (%)	Top Words
browser concepts	100	38.3	entry, sqlite, png
gui events	100	10.0	event, view, window, frame
gui elements	66.7	8.3	item, box, dlg, dialog
gtk ui framework	66.7	5.0	gtk, view, window, action
browser concepts	66.7	3.3	bookmark, node, action, menu
xml	33.3	5.0	xml, xslt, ctxt
page layout	33.3	5.0	frame, node, element
fonts	33.3	3.3	ft, face, glyph
qt ui framework	33.3	3.3	qstring, qwidget, parent, window
tabbing	33.3	1.7	tab, current, slot
Total:	**3 systems**	**60 topics**	—

The dominant topic clusters correspond to GUI, rendering and authentication concepts. Similar to the OS sub-domains, the top cluster bundles the majority of topics (here 38.3%). This time, the top cluster combines various typical *browser concepts*, mapping to individual project topics like *buffer handling*, *address handling*, *network socket management*, *persistence* (e.g., the sqlite database functionality of Firefox and Chromium), *rendering* and *caching*.

Of the top 20 clusters, 5 are shared between at least 2 of the 3 projects (2 between all projects), comprising 65% of all topics. Those clusters contain the majority of commonality in the GUI-based browser sub-domain. The other 15 clusters contain variability. For example, topic clusters for the *webkit* engine and *tabbing* are specific to Chromium. Similarly, clusters like *xml* and *page layout* are also specific to 1 project.

Text-Based Browsers

The last sub-domain that we study is that of text-based browsers (Table VI). Given their niche market, neither of the two browsers can rely on standard libraries for HTML processing and rendering. As such, most of the topic clusters are related to those concepts. The only commonalities between both browsers are the handling of *http connection*s, support for HTML standards (*documentation standard*) and input handling from the terminal (*gnu gettext*).

Although only 3 of the 20 topic clusters are shared between the two projects, these clusters amount to 50% of all topics. For example, the top cluster covers *caching*, *URL handling*, *cookies* and *(SSL) connection management*. In addition, many 1-system clusters are ultimately similar topics that did not get coalesced by the K-means clustering algorithm. For example, the *html* and *html table* clusters of Links and the *htlist* cluster of Lynx clearly should be merged. We believe that incorpo-

TABLE VI
TEXT-BASED BROWSERS (TOP 10) AT SUB-DOMAIN LEVEL.

Topic Label	Spread (%)	Cluster Size (%)	Top Words
http connection	100	37.5	link, cookie, connection, cache
document standard	100	7.5	text, html, items
gnu gettext	100	5.0	hk, gettext, sys, html
host information	50	5.0	free, entry, host, status
terminal	50	5.0	term, terminal, ev
gui concepts	50	5.0	dlg, bookmark, menu
html	50	2.5	html, format, attr
content	50	2.5	charset, context, uc, xf
doc type	50	2.5	header, doc, size, buf
htlist	50	2.5	htlist, list, lex
Total:	**2 systems**	**40 topics**	—

TABLE VII
OS DOMAIN (TOP 10) AT DOMAIN LEVEL.

Topic Label	Spread (%)	Cluster Size (%)	Top Words
architectural specifics	100.0	52.9	reg, port, addr, regs, len, pci, serial, state
tcp/ip networking	71.4	17.9	ip, addr, client, smb, sctp
device control	57.1	7.1	dev, acpi, ieee, tx, pci, rx, card
compilation environment	50.0	5.4	op, tree, line, uart, reg, pkt
distributed filesystem	35.7	2.1	msg, snmp, raid, obj, node
low-level descriptions	28.6	2.1	fcb, hpgs, sft
scheduling	28.6	1.8	dp, cp, register, min
file parsing	28.6	1.4	fp, tok, fs, ptr, buf
authentication	21.4	1.1	krb, arp, ldap
posix threading	21.4	1.1	pthread, buffer, data, client
Total:	**14 systems**	**280 topics**	—

rating more project data (e.g., comments and documentation) would be able to avoid this topic aliasing problem.

Summary of Findings

Our study of sub-domain topic clustering shows that our balanced approach is able to identify topic clusters with highly varying spread and cluster size. A small number of clusters contains the majority of the commonality.

B. Domain Level

The domain level consists of systems that achieve similar goals, but may vary greatly in functionality. We analyze the OS domain by clustering the topic clusters of the embedded real-time, monolithic and cloud operating system sub-domains. The browser domain consists of the GUI-based and text-based browser sub-domains.

OS Domain

Table VII shows the results for the OS domain. The clusters span a wide range of concepts, such as *ip networking subsystem*, *architectural specifics* and *peripheral device support*. Only 4 clusters (spread across more than 80% of all domain topics) cover at least half of the projects. The top cluster contains basic functionality essential to any operating system, such as *low-level device management*, *low-level memory management* and *low-level IO*.

Furthermore, half of the clusters cover at most 3 projects (median of 21.4%). For example, Debian *binutils* is only employed by Haiku and FreeBSD, whereas *berkeley bind* is used by both BSD distributions and OpenSolaris. Given the wider range of systems, the lower spread is expected. Our approach exposes interesting variability in design choices of the corresponding operating system teams, which can be further explored by the domain analysts.

Browser Domain

Table VIII shows that the two browser sub-domains are highly related. This is evident by observing that the top two topics have 100% spread. In particular, *window layout* and *data contents* are universal amongst all browsers, whereas *html parser* and *threading* cover at least 4 of the 5 browsers. Those topic clusters cover topics such as *HTML rendering*, *cookie support*, *caching*, *(SSL) connection management*, *font management* and *bookmarks*. Half of the clusters cover at least two projects (median of 40%). We also see clusters emerging from a specific sub-domain, such as the *history* cluster of the GUI-based sub-domain.

Summary of Findings

The results of the domain-level study indicate that topic clustering is successful at identifying similar topics within a domain, topics that are specific to sub-domains and topics that are specific to individual systems. Contrary to the sub-domain level, clusters at the domain-level are spread across smaller subsets of projects. Similar to the sub-domain level, the top cluster groups the major commonalities between projects.

V. DISCUSSION

In Section III-B, we discussed the conceptual differences between a balanced application of LDA and an unbalanced application [3], [16], [20], [35], [36]. The former treats all systems' topics as equal, whereas the latter favours topics of larger systems [34]. In this section, we show that these differences indeed matter in practice.

We repeated our domain-level commonality analysis using an unbalanced approach. We applied LDA once on the ag-

TABLE VIII
WEB BROWSER DOMAIN (TOP 10) AT DOMAIN LEVEL.

Topic Label	Spread (%)	Cluster Size (%)	Top Words
window layout	100	44.0	view, window, event, widget, terminal, tab
data contents	100	8.0	data, entry, cache, link, format, size, attr
html parser	80	7.0	frame, context, style, web, html
threading	80	5.0	thread, signal, gl
gui window components	60	7.0	dialog, box, item, config, menu, font
xml	60	5.0	xml, xslt, ctxt, token
filesystem	60	3.0	file, path, home
page layout	40	3.0	item, module, sidebar, url
html links	40	3.0	anchor, entry, path, htanchor, dir
text-based components	40	2.0	text, items, line, term, htext
Total:	**5 systems**	**100 topics**	—

TABLE IX
(UN)BALANCED COMMONALITY ANALYSIS AT DOMAIN LEVEL.

	median spread		entropy	
	balanced	unbalanced	balanced	unbalanced
OS	21.4%	21.4%	0.99	0.71
browser	40.0%	40.0%	0.98	0.43

gregated source code of all operating systems and once on all web browser systems. Then, we mapped the topics back to the software systems in which they occur. For this, we calculated each project's membership in a particular topic, i.e., the percentage of the project's files containing the topic relative to the total number of files in the domain containing the topic. We mapped a project to a topic if its membership in the topic was at least 10%. This threshold reduces the impact of statistical noise in the LDA results.

In order to compare possible bias of the balanced and unbalanced domain-level analyses, we measure (1) the number of projects across which topics/clusters are spread, and (2) the fairness (uniformity) of the distribution of topic/clusters across projects. For the former, we calculate the median spread of the unbalanced topics/clusters, similar to what we did in Section IV. For the fairness, we measure the topic entropy [31] of each domain-level analysis, which is defined as $\left(\frac{-1}{log_{10}(N)}\right)\sum_{i=1}^{N} p_i.log_{10}(p_i)$ with N the number of software systems in a domain. The topic probability p_i for a project i is defined as $\left(\frac{t_i}{\sum_{j=1}^{N} t_j}\right)$, with t_i the number of topics for which project i has a membership larger than 10%. If all projects cover the same number of topics (i.e., $t_i = t_j, \forall i, j$), then $p_i = 1/N$ and the entropy becomes 1. If all topics are concentrated in 1 project k, then $p_k = 1$ ($p_i = 0, \forall i \neq k$) and the entropy becomes 0. In other words, the higher the fairness with which topics are spread across projects, the higher the entropy. As an alternative to entropy, one could also use related metrics like the Gini coefficient.

Table IX shows the resulting median spread and entropy values for the balanced and unbalanced commonality analyses of the operating system and web browser domains. A first observation is that the balanced and unbalanced analyses in each domain have identical values for median spread. Half of the operating systems clusters are spread across at least 21.4% of the projects (3 projects), whereas half of the web browser clusters are spread across at least 40% (2 projects).

However, we note that the entropies for our balanced approach are practically 1, i.e., total fairness, whereas the entropy for the unbalanced analyses of the OS and (especially) the browser domains are substantially lower. Closer investigation learns that the unbalanced browser analysis considers topics of only 2 out of the 5 systems (Firefox and Chromium), favouring GUI- and JavaScript-related topics. Similarly, the unbalanced operating system analysis only considers topics of 9 out of the 14 projects (TinyOS, Inferno, Minix, Unix and Ultrix are ignored), with a strong bias towards the monolithic operating systems (largest sub-domain). Only one topic ("interface") is Chromium-related.

This discussion shows that the differences between a balanced and unbalanced commonality analysis matter. Practitioners that want to prioritize common and variable concepts based on popularity in a domain should use the unbalanced approach. Yet, practitioners that want to extract all common or variable concepts in a domain, regardless of project size, should consider a balanced approach.

VI. THREATS TO VALIDITY

Although source code comments and documentation carry more semantic information, our case study only considered source code as input to obtain a lower bound on the performance of our balanced approach. Our results show a strong overlap between software projects in most of our (sub-)domain studies. In addition, additional data sources can be incorporated easily into our approach, since LDA is geared towards natural language data and only our pre-processing stage is source code- and programming language-specific.

Our case study only explores open source software systems. Further analysis is required on proprietary systems to validate our findings. We have some confidence that our findings would hold, since two of our operating system subjects, i.e., OpenSolaris and Haiku, both originate from a proprietary system (Solaris and BeOS, respectively). Furthermore, our results depend on human interpretation. To deal with this threat, two of the authors independently interpreted the results for the various (sub-)domains.

Finally, the thresholds for LDA, K-means clustering, and the mapping between topics and projects (Section V) were chosen based on previous work and experience. More thorough experimentation is needed to determine optimal values for these thresholds, possibly customized for each subject system.

VII. Conclusion

We presented an approach for commonality and variability analysis across multiple software projects at different levels of granularity (sub-domain or domain). By applying topic analysis on each project in isolation, then clustering the topics of all projects at various levels of granularity, our approach balances the contributions of each project in the analysis.

A large-scale case study on 19 C/C++ operating systems and web browsers shows that our methodology is indeed able to identify (sub-)domain-wide commonalities and variability. We found that a minority of topic clusters groups the majority of common topics. By mapping the clusters to the individual project topics, we were able to analyze the major commonalities between the different projects. In addition, we showed how our balanced approach behaves conceptually different from unbalanced topic modeling approaches.

Our results are promising as they can help domain analysts in their largely manual analysis of a particular domain. We are currently exploiting additional data sources like source code comments and documentation to enrich our topic models.

References

[1] http://sail.cs.queensu.ca/publications/pubs/scam2011data.zip, June 2011.

[2] M. Ardis, P. Dudak, L. Dor, W.-j. Leu, L. Nakatani, B. Olsen, and P. Pontrelli, "Domain engineered configuration control," in *Proc. of the 1st conf. on Software Product Lines (SPLC)*, Denver, CO, USA, 2000, pp. 479–493.

[3] P. F. Baldi, C. V. Lopes, E. J. Linstead, and S. K. Bajracharya, "A theory of aspects as latent topics," in *Proc. of the 23rd ACM SIGPLAN conf. on Object-oriented programming systems languages and applications (OOPSLA)*, 2008, pp. 543–562.

[4] J. Bayer, O. Flege, P. Knauber, R. Laqua, D. Muthig, K. Schmid, T. Widen, and J.-M. DeBaud, "Pulse: a methodology to develop software product lines," in *Proc. of the 1999 symp. on Software reusability (SSR)*, LA, CA, USA, 1999, pp. 122–131.

[5] D. Blei, A. Ng, and M. Jordan, "Latent dirichlet allocation," *The Journal of Machine Learning Research*, vol. 3, pp. 993–1022, 2003.

[6] I. T. Bowman, R. C. Holt, and N. V. Brewster, "Linux as a case study: its extracted software architecture," in *Proc. of the 21st Intl. Conference on Software Engineering (ICSE)*, LA, CA, USA, 1999, pp. 555–563.

[7] J. Coplien, D. Hoffman, and D. Weiss, "Commonality and variability in software engineering," *IEEE Softw.*, vol. 15, pp. 37–45, Nov. 1998.

[8] David Wheeler, "SLOCCount," http://www.dwheeler.com/sloccount/.

[9] S. Ferber, J. Haag, and J. Savolainen, "Feature interaction and dependencies: Modeling features for reengineering a legacy product line," in *Proc. of the 2nd Intl. Conf. on Software Product Lines (SPLC)*, 2002, pp. 235–256.

[10] W. B. Frakes and K. Kang, "Software reuse research: Status and future," *IEEE Trans. Softw. Eng.*, vol. 31, pp. 529–536, July 2005.

[11] W. B. Frakes, R. Prieto-Díaz, and C. Fox, "DARE: Domain analysis and reuse environment," *Ann. Softw. Eng.*, vol. 5, pp. 125–141, Jan. 1998.

[12] G. Holmes, A. Donkin, and I. Witten, "Weka: A machine learning workbench," in *Proc. of the 1994 2nd Australian and New Zealand Conf. on Intelligent Information Systems (ANZIIS)*, 1994, pp. 357–361.

[13] I. John and J. Dörr, "Extracting product line model elements from user documentation," Fraunhofer IESE, Tech. Rep. 112.03/E, Oct. 2003.

[14] K. C. Kang, S. G. Cohen, J. A. Hess, W. E. Novak, and A. S. Peterson, "Feature-oriented domain analysis (foda) – feasibility study," Carnegie Mellon, SEI, Tech. Rep. CMU/SEI-90-TR-21, Nov. 1990.

[15] K. C. Kang, J. Lee, and P. Donohoe, "Feature-oriented project line engineering," *IEEE Softw.*, vol. 19, pp. 58–65, July 2002.

[16] S. Kawaguchi, P. K. Garg, M. Matsushita, and K. Inoue, "Mudablue: An automatic categorization system for open source repositories," in *Proc. of the 11th Asia-Pacific Software Engineering Conf. (APSEC)*, Busan, Korea, 2004, pp. 184–193.

[17] K. Lee, K. C. Kang, E. Koh, W. Chae, B. Kim, and B. W. Choi, "Domain-oriented engineering of elevator control software: a product line practice," in *Proc. of the 1st conf. on Software product lines (SPLC)*, Denver, CO, USA, 2000, pp. 3–22.

[18] F. J. v. d. Linden, K. Schmid, and E. Rommes, *Software Product Lines in Action: The Best Industrial Practice in Product Line Engineering*. Springer-Verlag New York, Inc., 2007.

[19] E. Linstead, P. Rigor, S. Bajracharya, C. Lopes, and P. Baldi, "Mining concepts from code with probabilistic topic models," in *Proc. of the 22nd IEEE/ACM intl. conf. on Automated Software Engineering (ASE)*, 2007, pp. 461–464.

[20] Y. S. Maarek, D. M. Berry, and G. E. Kaiser, "An information retrieval approach for automatically constructing software libraries," *IEEE Trans. Softw. Eng.*, vol. 17, pp. 800–813, Aug. 1991.

[21] A. K. McCallum, "Mallet: A machine learning for language toolkit," 2002, http://mallet.cs.umass.edu.

[22] K. Mens, A. Kellens, and J. Krinke, "Pitfalls in aspect mining," in *Proc. of the 2008 15th Working Conf. on Reverse Engineering (WCRE)*, Antwerp, Belgium, 2008, pp. 113–122.

[23] G. C. Murphy, D. Notkin, and K. Sullivan, "Software reflexion models: bridging the gap between source and high-level models," in *Proc. of the 3rd ACM SIGSOFT symp. on Foundations of Software Engineering (FSE)*, Washington, D.C., USA, 1995, pp. 18–28.

[24] J. M. Neighbors, "Software construction using components," Ph.D. dissertation, University of California, Irvine, 1980.

[25] D. Newman and S. Block, "Probabilistic topic decomposition of an 18th century American newspaper," *Journal of the American Society for Information Science and Techn.*, vol. 57, no. 6, pp. 753–767, 2006.

[26] D. L. Parnas, "On the design and development of program families," *IEEE Trans. Softw. Eng.*, vol. 2, pp. 1–9, Jan. 1976.

[27] K. Pohl, G. Böckle, and F. J. v. d. Linden, *Software Product Line Engineering: Foundations, Principles and Techniques*. Springer-Verlag New York, Inc., 2005.

[28] D. Poshyvanyk, Y.-G. Gueheneuc, A. Marcus, G. Antoniol, and V. Rajlich, "Feature location using probabilistic ranking of methods based on execution scenarios and information retrieval," *IEEE Trans. Softw. Eng.*, vol. 33, pp. 420–432, June 2007.

[29] R. Prieto-Díaz, "Domain analysis: an introduction," *SIGSOFT Softw. Eng. Notes*, vol. 15, pp. 47–54, April 1990.

[30] M. P. Robillard and G. C. Murphy, "Concern graphs: finding and describing concerns using structural program dependencies," in *Proc. of Intl. Conf. on Software Engineering (ICSE)*, May 2002, pp. 406–416.

[31] C. E. Shannon, "Prediction and entropy of printed english," *Bell System Technical Journal*, vol. 3, pp. 53–64, 1951.

[32] M. A. Simos, "Organization domain modeling (odm): formalizing the core domain modeling life cycle," in *Proc. of the 1995 Symp. on Software reusability (SSR)*, Seattle, WA, USA, 1995, pp. 196–205.

[33] S. W. Thomas, B. Adams, A. E. Hassan, and D. Blostein, "Validating the use of topic models for software evolution," in *Proc. of the 10th IEEE Intl. Working Conf. on Source Code Analysis and Manipulation (SCAM)*, Timişoara, Romania, 2010, pp. 55–64.

[34] ——, "Modeling the evolution of topics in historical software repositories," in *Proc. of the 8th IEEE Working Conf. on Mining Software Repositories (MSR)*, Waikiki, HI, USA, May 2011, pp. 173–182.

[35] K. Tian, M. Revelle, and D. Poshyvanyk, "Using latent dirichlet allocation for automatic categorization of software," in *Proc. of the 6th IEEE Intl. Working Conf. on Mining Software Repositories (MSR)*. IEEE, 2009, pp. 163–166.

[36] S. Ugurel, R. Krovetz, and C. L. Giles, "What's the code?: automatic classification of source code archives," in *Proc. of the 8th ACM SIGKDD intl. conf. on Knowledge Discovery and Data mining (KDD)*, Edmonton, AB, Canada, 2002, pp. 632–638.

[37] P. van der Spek, S. Klusener, and P. van de Laar, "Towards recovering architectural concepts using latent semantic indexing," in *Proc. of the 12th European Conf. on Software Maintenance and Reengineering (CSMR)*, 2008, pp. 253–257.

[38] D. M. Weiss and C. T. R. Lai, *Software product-line engineering: a family-based software development process*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1999.

[39] O. Yilmaz and W. B. Frakes, "A case study of using domain engineering for the conflation algorithms domain," in *Proc. of the 11th Intl. Conf. on Software Reuse (ICSR)*, Falls Church, VA, USA, 2009, pp. 86–94.