Diffusion of Software Features: An Exploratory Study

Ferdian Thung, David Lo, and Lingxiao Jiang
School of Information Systems
Singapore Management University, Singapore
{*ferdianthung,davidlo,lxjiang*}*@smu.edu.sg*

***Abstract*—New features are frequently proposed in many software libraries. These features include new methods, classes, packages, etc. These features are utilized in many open source and commercial software systems. Some of these features are adopted very quickly, while others take a long time to be adopted. Each feature takes much resource to develop, test, and document. Library developers and managers need to decide what feature to prioritize and what to develop next. As a first step to aid these stakeholders, we perform an exploratory study on the diffusion or rate of adoption of features in Java Development Kit (JDK) library. Our empirical study proposes such questions as how many new features are adopted by client applications; how long it takes for a new feature to spread to various software products; what features are diffused quickly; and what features are diffused widely. We perform an exploratory study with new features in Java Development Kit (JDK, from version 1.3 to 1.6) and provide empirical findings to answer the above research questions.**

I. Introduction

Different from heavy machinery that changes slowly, software systems change frequently. New features are continually added to a system. These features, often expressed as new APIs (e.g., methods or classes that can be used by other programs or users), range from new functionalities, new user interfaces, improved compatibility, optimized re-implementation of old functionalities for better performance and scalability, and much more.

Introducing new features, however, incurs cost. They need time to be developed. Testing and validation efforts are also needed to ensure that the new features work as intended. In addition, introducing new features might also introduce risks as old features that are previously working fine may now stop working due to incompatibilities with the new features. Such cost should be justified properly when resources are dedicated to develop the new features. The justification often comes when the new features are used by and benefit end users. For software libraries, a successful feature is one that is used widely by many client applications.

In the industry setting, the decision to incorporate new features are often made by market analysts. They often look for features that are well sought after by potential customers and thus the features are expected to be more "marketable" when introduced. Marketable features are diffused widely in the community of adopters. Analysts often investigate historical data and the characteristics of features that are previously marketable and base their decisions on such investigations. There have been many studies in the business and management community on factors that affect the diffusion of various products [21], [23], [24].

Despite the proliferation of such studies in the marketing community, there has been little research in the software engineering community that investigates the diffusion, or rate of adoption, of various software features. Such knowledge is important to guide decision makers in prioritizing features to be released first and in developing features that would be well adopted. Poor decisions in a commercial setting can prove detrimental to a company (*c.f.* [14]). Thus, there is a need for more studies that can guide decision makers in deciding properly what features to focus on first.

In this study, as a first step to fill the above mentioned need, we perform an exploratory study that analyzes the *diffusion*, or rate of adoption, of software features. We are interested to investigate how fast features get adopted and what kind of features get adopted quickly. We focus in particular on new features introduced in software libraries, such as Java Development Kit (JDK).

To perform this study, we download various versions of JDK and enumerate all new features in a downloaded version against its previous version. Also, we track a pool of 15 medium-large Java applications implementing various functionalities and see when they incorporate these new features (if ever). We then analyze the distribution of different rates of adoption and characterize features that are well adopted (or diffused) and those that do not.

The contributions of this study are as follows:

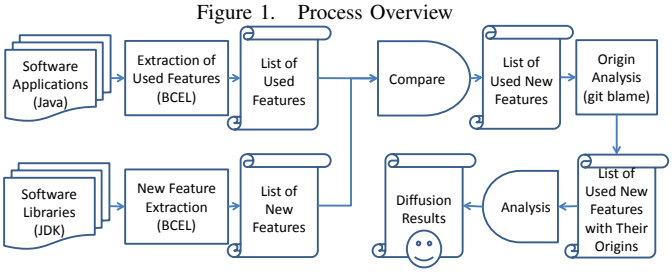
1) To the best of our knowledge, this is the first study that investigates the diffusion of software features.
2) We analyze a large collection of software features in JDK and track the diffusion of these features in a collection of Java applications.
3) We present a characterization of features that are well diffused and those that do not.

The structure of this paper is as follows. In Section II, we present our methodology and data collection steps. In Section III, we present our research questions and empirical findings. We discuss interesting issues in Section IV. We present related work in Section V. We conclude with future work in Section VI.

II. Methodology

Our approach is divided into three stages as illustrated in Figure 1: (1) new feature extraction that extracts new methods introduced by a new version of a library, (2) feature tracking that extracts all used methods in an application and decides the origins of the code that uses the methods, and (3) feature analysis that analyzes how new methods introduced by a program are diffused to and used in other programs.

Figure 1. Process Overview



A. New Feature Extraction

In this step, we extract new features from software libraries. We compare and contrast pairs of versions of a software library to detect new methods that only exist in the newer version. For each JDK version, we use Byte Code Engineering Library (BCEL) [1] to extract all methods that appear in all classes in the version. We then compare and contrast the list of methods from two consecutive versions to get a list of new *public* methods introduced in the newer version. In the list of new methods introduced, we use the date when the newer version of JDK was released to the public as the date when the methods were introduced.

B. Feature Tracking

In this step, we track which and when new features are used in the potential client population. In this paper, we take a sample client population which is comprised of 15 medium to large Java applications implementing various functionalities. For each application, we clone its Git [2] repository and analyze the latest version of the application to find out all of the new features of a particular version of JDK that are used in this application. The algorithm we use to identify the new features used in an application and the time they are first used is illustrated in Figure 2.

In the algorithm, given an application, we first extract the list of all method invocations together with their signatures and the line numbers corresponding to the call locations by using Byte Code Engineering Library (BCEL) (lines 1-3). We then only keep the methods that are new features of a particular version of JDK (lines 4-7). At this point, we get the new features used in the application.

When analyzing the diffusion of new features, we should only consider features that are *explicitly* used by programmers, excluding ones that are only *implicitly* used due to automatic program transformation done by Java compilers. Thus, in the feature tracking algorithm, we only keep explicit methods (line 6-7). We do this by checking the extracted method (from Java bytecode) against the actual line of code in the source file. If that line actually contains a method name matching the extracted method invocation, we consider it as explicit, otherwise we consider it as implicit.

Procedure GetUsedNewFeatures
Inputs:
new : new features in JDK
$client$: client application
git_repo: clone of client application git repository
Output:
$Result$: A set of used new features and the time when they are first used
Methods:
1 : Let L_c be a set to contain all method invocations together with their signatures and the line numbers in source code;
2 : For each method invocation m in $client$
3 : Add m's signature together with its line number to L_c;
4 : Let L_z be a set to contain used new features in $client$;
5 : For each $item$ in L_c
6 : If the method in $item$ is in new and is an explicit call
7 : Add $item$ to L_z;
8 : For each $item$ in L_z
9 : Run `git blame` for $item$'s line in git_repo;
10: Get the time the method invocation was introduced;
11: Add the method together with the time to $Result$;
12: Output $Result$;

Figure 2. Feature Tracking Procedure

In addition, since we want to understand how quickly JDK new features are adopted by Java applications, we also need to find out the time when the features are used in each application. For this purpose, we utilize `git blame` [3] command on the given application's Git repository. This command would tell us the time when a given line of code in the application is last modified or added. When the line contains a call to a new feature of a JDK version, the time from `git blame` likely indicates the time when the new feature is used in the application (line 9). This is often called *origin analysis* in the literature [4], [11], [16], [22], [26]. When multiple lines of the application contain calls to the same feature, we take the earliest time from multiple `git blame` commands as the time when the new feature is first used in the application (line 10-11). Note that `git blame` is only performed for explicitly called methods (line 5-11).

C. Feature Analysis

At the end of the above steps, we would have retrieved all of the new features that are used in an application together with the time when they are first used in the application.

For each feature, we compute the period of time that has lapsed since the new feature was introduced to the public, and consider this period of time as the time needed for the feature to be diffused to the target application.

We perform the above steps for every application and perform a number of analysis for new features, such as what is the distribution of the time needed for a feature to be adopted and how many applications use each feature. We also compare and contrast such information for various versions of JDK (1.3-1.6). Furthermore, we show some features that have been diffused quickly and widely (in Section III).

Table I
JDK RELEASE DATE

JDK	Release Date	Size(kLOC)
J2SE 1.2	(December 8, 1998)	484.801
J2SE 1.3	(May 8, 2000)	572.312
J2SE 1.4	(February 6, 2002)	1,179.732
J2EE 5.0	(September 30, 2004)	1,883.740
Java SE 6	(December 11, 2006)	2,033.103

Table II
NUMBER OF NEW FEATURES FOR JDK VERSIONS 1.3-1.6

JDK	# Features	# New Features
1.3	26759	5262
1.4	44449	18923
1.5	47079	10303
1.6	53780	7861

III. EXPERIMENTS

We describe our dataset, research questions, and empirical findings.

A. Dataset

In this paper, we analyze new features introduced in JDK versions 1.3 to 1.6, and thus need to download JDK versions 1.2 to 1.6. We ignore JDK 1.0 and 1.1 as many open source projects written in Java did not exist at the time when older versions of JDK were released and JDK may not be mature enough at that time. We also ignore latest versions of JDK (1.7 and above) as it is not old enough to be diffused to various applications. Table I show the various versions of JDK that we use and their release dates.

For JDK versions 1.3 to 1.6, we extract newly introduced methods in each version by constrasting all methods contained in the version and its previous version as explained in Section II. The numbers of new methods (i.e., new features) for the JDK versions are shown in Table II along with the total numbers of methods. To find how well diffused these new features are, we collect a set of 15 open source Java projects shown in Table III which use these features.

B. Research Questions

RQ1. How many new features in a library are utilized by client applications?

Table III
CLIENT PROJECTS ANALYZED IN THIS STUDY

Project	History	Size(kLOC)	Functionality
ant	13/01/00-28/12/11	204.024	Java build tool
batik	01/10/00-12/01/12	337.176	SVG manipulation toolkit
derby	11/08/04-13/01/12	1,167.936	Relational database
fop	31/10/99-10/01/12	237.788	XSL-FO implementation
hadoop	27/01/06-18/06/09	327.743	Large cluster application framework
ivy	16/06/05-15/01/12	74.140	Project dependencies tool
junit	03/12/00-31/12/11	11.989	Unit testing tool
log4j	16/11/00-17/01/12	43.788	Logging tool
lucene	11/09/01-25/11/09	81.300	Text search engine library
nutch	23/01/05-14/01/12	29.800	Web-search software
poi	31/01/02-17/01/12	156.776	APIs for Microsoft's OLE 2 Compound file formats
santuario-java	28/09/01-13/01/12	51.008	XML primary security standards
tomcat	27/03/06-17/01/12	336.703	Web server and servlet container
xalan-j	09/11/99-1/01/12	186.617	XSLT processor
xerces2-j	09/11/99-2/01/12	193.041	XML manipulation library

Table IV
NUMBERS OF NEW FEATURES USED BY THE SAMPLE PROJECTS

JDK Version	# New Features Used	Proportion
1.3	15	0.285%
1.4	595	3.144%
1.5	181	1.757%
1.6	160	2.035%

RQ2. How long would it take for a new feature to spread to various applications? What features are diffused quickly? What are some characteristics of the features that are diffused *quickly*?

RQ3. How many applications would adopt a new feature? What features are diffused widely to many applications? What are some characteristics of features that are diffused *widely*?

We present our empirical findings for answering the above research questions in the following subsections based on the dataset described in Section III-A.

C. RQ1: Utilization of New Features

To answer this question, we analyze the numbers of newly introduced methods for various versions of JDK used in the 15 software projects. Table IV provides this information.

We notice that 595 new features of JDK 1.4 are used by the 15 applications. On the other hand, only 15 new features of JDK 1.3 are used in the applications. Proportion wise, we find that only a small percentage of new features are used (0.265%-2.969%). This shows that most features are not widely diffused to many software applications.

D. RQ2: Speed of Diffusion

We define *speed* as the measure of how fast the new features were used in the client applications after the corresponding JDK version was released publicly. There are two parts to this research question: first, we would like to find the diffusion speed of various new features; second, we want to find features that have the highest diffusion speed.

To answer the first part, Figure 3 (left) plots the period of time that has elapsed until a feature is adopted (x-axis) versus the number of new features with this behavior (y-axis). Figure 3 (right) is the corresponding cumulative graph. We note that only 77 features (8% of all adopted features) are adopted within 100 days. The 600-700 days time interval bucket has the highest number of features. It can also be noted that 50% of all new features that are eventually adopted are used in less than 1000 days (i.e., 2.74 years). From the graph, we can also infer that the likelihood of a feature being adopted decreases over time. Thus unpopular features are likely to remain unpopular.

We also compare the average, minimum, and maximum diffusion speeds of adopted features for different versions of JDK. We show these in Table V. The last column of the table is the average adoption time of features that are adopted within 5 years. We use 5 years as this is the period time that has elapsed since the introduction of JDK 1.6 to

Table V
AVERAGE DIFFUSION SPEED PER JDK VERSION (IN DAYS)

JDK	Avg	Min	Max	Median	Avg 5 Years
1.3	1513.67	228	3425	1039	414.375
1.4	1243.44	8	3569	1112	647.2
1.5	1186.23	145	2661	1145	865.02
1.6	849.28	238	1803	605	849.28

today. We notice that the averages generally increase (except from JDK 1.5 to 1.6) with each newer JDK version. This might be due to the increasing maturity of Java API such that more functionalities could be done easily with existing set of features.

To answer the second part, Table VI highlight the top 10 features that has the highest speed of diffusion. The top 10 features include methods related to DOM and XML. These methods are the core methods in their corresponding classes. The fast adoption of these features is affected by the popularity of DOM and XML which are the "hot technologies" in the market at that time. We also note that all the top-10 features with highest diffusion speed comes from JDK 1.4.

We also show the top-3 features with the highest diffusion speed per JDK version in Table VII. The top-3 features of JDK 1.3 consists of methods related to sound functionality. The top-3 features of JDK 1.4 are related to DOM. The top-3 features of JDK 1.5 are related to DOM and XML. The top-3 features of JDK 1.6 are related to general data types and structures.

E. RQ3: Breadth of Diffusion

We define *breadth* as the measure of how many client applications adopt the new features. Again, there are two parts to this research question: first, we would like to find the breadth of diffusion of various new features; second, we want to find features that has the widest diffusion breadth.

To answer the first part, Figure 4 (left) plots the number of adopting client applications (x-axis) versus the number of new features with this behavior (y-axis). Figure 4 (right) is the corresponding cumulative graph. It can be noted that most adopted software features are only used in one project. This shows that many new features are very specific to a particular need and thus only affect a small subset of the client applications.

We also compare the average, minimum, and maximum diffusion breadths of adopted features for different versions of JDK. We show these in Table VIII. The last column of the table is the average diffusion breadth of features that are adopted within 5 years. We notice that JDK 1.4 has the highest average breadth. Upon closer inspection, we find that this is the case as many of the features released in JDK 1.4 relate to XML or general purpose data structures.

To answer the second part, Table IX highlights the top 10 features that has the widest diffusion breadth. The top 10 features include methods under `xml`, `dom`, and `lang` packages. We can see that these methods are either XML related methods, or general purpose methods that are used to manipulate common data structures.

Table VIII
AVERAGE DIFFUSION BREADTH PER JDK VERSION

JDK	Avg	Min	Max	Median	Avg 5 Years
1.3	1.13	1	2	1	1
1.4	1.47	1	5	1	1.53
1.5	1.28	1	4	1	1.31
1.6	1.08	1	4	1	1.08

We also show the top-3 features with the widest diffusion breadth per JDK version in Table X. There are only 2 features of JDK 1.3 that are used in more than 1 client application. The top-2 features of JDK 1.3 consists of methods in the URL class and Collection class – both are data structures commonly used in various applications. The top-3 features of JDK 1.4 are related to DOM and XML. The top-3 features of JDK 1.5 and 1.6 are related to general purpose data structures.

IV. DISCUSSION

We are interested in the diffusion properties of software features as it will give us some insights about characteristics of features that are adopted fast or used widely. By knowing what features are well diffused, developers of programming language libraries can focus on finishing and releasing these features first.

The following are the summary of our exploratory study:

1) Most new features are not diffused well in the applications analyzed in this study. Indeed only up to 3% new features of the various versions of JDK are used by the applications. This may suggest the need for promoting the utilization of these features — either by automatic program transformation or by leveraging more means (e.g., forums, social media, tutorials, etc.) to diffuse information to developers.
2) Only a few features (8%) are diffused fast (within 100 days). Most features take time to be adopted. Only features related to current hot technological trend get adopted quickly (e.g., in 8 days for some DOM related methods). Thus developers need to catch latest technological trend. If they are able to do this, there would be many adopters to their newly created features.
3) Most features are diffused slowly. A feature not adopted at a period of time is less likely to be adopted in future.
4) Most Java libraries provide specific features that are peculiar for a special functionality. However, these special functionalities are often not widely used. Most new features of JDK 1.3 to 1.6 are used in only 1 client out of the 15 client applications analyzed in our study.
5) Adding new functionalities to general purpose data structures are good as they are often adopted widely and quickly.

V. RELATED WORK

We present related work on product & information diffusion and feature recommendation in software engineering.

Figure 3. Speed of Diffusion: (Left) Time to Adoption vs. # Features, (Right) Corresponding Cumulative Graph

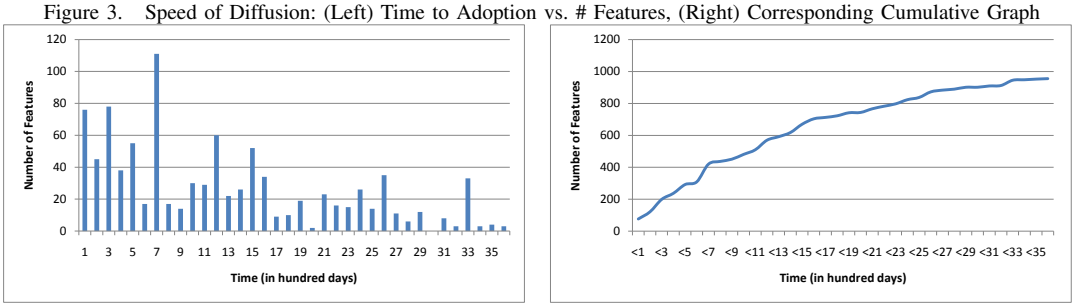


Table VI
TOP 10 NEW FEATURES WITH HIGHEST DIFFUSION SPEED

No	Method Signature	Speed(days)	JDK Version
1	org.w3c.dom.Entity.getNotationName()Ljava/lang/String;	8	1.4
2	org.w3c.dom.Entity.getPublicId()Ljava/lang/String;	8	1.4
3	org.w3c.dom.Entity.getSystemId()Ljava/lang/String;	8	1.4
4	org.w3c.dom.Notation.getPublicId()Ljava/lang/String;	8	1.4
5	org.w3c.dom.Notation.getSystemId()Ljava/lang/String;	8	1.4
6	org.apache.xpath.objects.XObject.execute(Lorg/apache/xpath/XPathContext;)Lorg/apache/xpath/objects/XObject;	26	1.4
7	org.xml.sax.SAXException.getException()Ljava/lang/Exception;	28	1.4
8	org.xml.sax.AttributeList.getLength()I	28	1.4
9	org.xml.sax.Parser.setDocumentHandler(Lorg/xml/sax/DocumentHandler;)V	28	1.4
10	org.xml.sax.AttributeList.getValue(I)Ljava/lang/String;	28	1.4

Table VII
TOP 3 NEW FEATURES WITH HIGHEST DIFFUSION SPEED PER JDK VERSION

No	JDK	Method Signature	Speed(days)
1	1.3	javax.sound.sampled.Clip.open(Ljavax/sound/sampled/AudioInputStream;)V	228
2	1.3	javax.sound.sampled.Clip.loop(I)V	228
3	1.3	javax.sound.sampled.Line.close()V	228
4	1.4	org.w3c.dom.Entity.getNotationName()Ljava/lang/String;	8
5	1.4	org.w3c.dom.Entity.getPublicId()Ljava/lang/String;	8
6	1.4	org.w3c.dom.Entity.getSystemId()Ljava/lang/String;	8
7	1.5	org.w3c.dom.DOMErrorHandler.handleError(Lorg/w3c/dom/DOMError;)Z	145
8	1.5	javax.xml.datatype.Duration.getDays()I	218
9	1.5	javax.xml.datatype.Duration.getHours()I	218
10	1.6	java.util.Queue.add(Ljava/lang/Object;)Z	238
11	1.6	java.util.SortedMap.entrySet()Ljava/util/Set;	254
12	1.6	java.util.SortedMap.keySet()Ljava/util/Set;	254

Figure 4. Breadth of Diffusion: (Left) # Adopting Applications vs. # Features, (Right) Corresponding Cumulative Graph

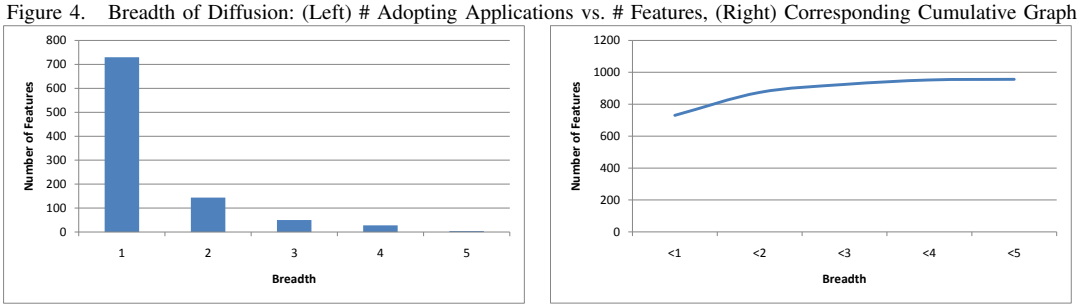


Table IX
TOP 10 NEW FEATURES WITH WIDEST DIFFUSION BREADTH

No	Method Signature	Breadth	JDK Version
1	javax.xml.parsers.DocumentBuilderFactory.newDocumentBuilder()Ljavax/xml/parsers/DocumentBuilder;	5	1.4
2	javax.xml.transform.TransformerFactory.newInstance()Ljavax/xml/transform/TransformerFactory;	5	1.4
3	org.w3c.dom.Document.getImplementation()Lorg/w3c/dom/DOMImplementation;	5	1.4
4	org.w3c.dom.Element.setAttributeNS(Ljava/lang/String;Ljava/lang/String;Ljava/lang/String;)V	5	1.4
5	java.lang.Boolean.valueOf(Z)Ljava/lang/Boolean;	4	1.4
6	java.lang.Integer.valueOf(I)Ljava/lang/Integer;	4	1.5
7	java.lang.String.replaceAll(Ljava/lang/String;Ljava/lang/String;)Ljava/lang/String;	4	1.4
8	java.lang.String.split(Ljava/lang/String;)[Ljava/lang/String;	4	1.4
9	java.lang.StringBuilder.append(Ljava/lang/Object;)Ljava/lang/StringBuilder;	4	1.5
10	java.lang.StringBuilder.append(Ljava/lang/String;)Ljava/lang/StringBuilder;	4	1.5