Service Evolution Analytics: Change and Evolution Mining of a Distributed System

Animesh Chaturvedi, Aruna Tiwari, Dave Binkley, and Shubhangi Chaturvedi

***Abstract*—Changeability and evolvability analysis can aid an engineer tasked with a maintenance or an evolution task. This article applies *change mining* and *evolution mining* to evolving distributed systems. First, we propose a *Service Change Classifier* based Interface Slicing algorithm that mines change information from two versions of an evolving distributed system. To compare old and new versions, the following change classification labels are used: inserted, deleted, and modified. These labels are then used to identify subsets of the operations in our newly proposed Interface (WSDL) Slicing algorithm. Second, we proposed four *Service Evolution Metrics* that capture the evolution of a system's *Version Series* VS $= \{V_1, V_2, \ldots, V_N\}$. Combined the two form the basis of our proposed *Service Evolution Analytics* model, which includes learning during its development phase. We prototyped the model in an intelligent tool named AWSCM (Automatic Web Service Change Management). Finally, we present results from experiments with two well-known cloud services: Elastic Compute Cloud (EC2) from the Amazon Web Service (AWS), and Cluster Controller (CC) from Eucalyptus. These experiments demonstrate AWSCM's ability to exploit change and evolution mining.**

***Index Terms*—Change classification, cloud services, distributed systems and services lifecycle management, evolution metrics.**

I. Introduction

DISTRIBUTED COMPUTING models such as Grid Computing, Cloud Computing, Utility Computing, and Service Oriented Computing rely upon service frameworks. Grid Computing, the mother of all four, combines a distributed collection of computing resources to achieve scalability. Cloud Computing enables convenient, on-demand shared computing resources at several levels (e.g., at the infrastructure, platform, and software levels). Utility Computing enables an on-demand, pay-as-you-go billing method in which a service provider makes computing resources available to customers. Service Oriented Computing supports loosely coupled distributed systems through the sharing of remote Web Service.

Manuscript received May 29, 2019; revised November 19, 2019 and February 4, 2020; accepted April 8, 2020. Review of this manuscript as arranged by Department Editor P. Hung. *(Corresponding author: Animesh Chaturvedi.)*

Animesh Chaturvedi is with the Indian Institute of Technology Indore, Indore 453552, India, and also with the Department of Informatics, King's College London WC2R 2LS, London, U.K. (e-mail: animesh.chaturvedi88@gmail.com).

Aruna Tiwari is with the Indian Institute of Technology Indore, Indore 453552, India (e-mail: artiwari@iiti.ac.in).

Dave Binkley is with the Loyola University Maryland, Baltimore, MD 21210 USA (e-mail: binkley@cs.loyola.edu).

Shubhangi Chaturvedi is with the Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, Jabalpur 482005, India (e-mail: chaturvedishubhangi51@gmail.com).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TEM.2020.2987641

An evolving distributed system is a kind of evolving system [1], [2] that often stored in a software repository to support its continuous evolution. Similar to systems in other domains [3], continuous changes to the requirements of distributed systems mandate their ongoing maintenance and evolution. This process becomes increasingly challenging as system complexity grows and thus tool support is needed to reduce the cost (or effort) of maintaining an evolving distributed system.

We are interested in studying the changeability [4]–[6] and evolvability [7], [8] of distributed systems. This article has two parts based on two data mining techniques. The first, involves *change mining* [9], contrast mining [10], or change impact analysis [31], which aim to discover change information. The second part, evolution mining, aims to discover evolution information e.g., evolution mining in social networks [11], [12].

Our approach makes use of data mining techniques to extract information from an evolving distributed system. The particular distributed systems we focus to support data mining of web-based systems. Traditionally, such web-mining techniques come in three types: web content mining, web usage mining, and web structure mining [13], [14]. We consider the latter, web structure mining, which retrieves information from service interface documents (e.g., HTML, XML, and WSDL documents) as well as the underlying implementation. In doing so, we consider a cloud service as a time-varying dataset and apply techniques combining aspects of *change mining* and *evolution mining*.

The motivation behind applying change and evolution mining for distributed systems is to uncover change and evolution information over time. Firstly, we use classification theory to label the changes that differentiate two versions of a distributed system. Secondly, we devise software metrics to capture the evolution over a series of versions. These two kinds of information help to characterize the changes and evolution of a cloud system. Creation of automated techniques that exploit this information is a challenging task.

Our earlier work [15]–[17] introduced a *Subset WSDL construction algorithm*, and defined *interface slicing*. The resulting WSDL slices (or Subset WSDLs) are useful to select a subset of the test cases for regression-testing a subset service [18]–[20]. Building on this work, this article proposes a change mining of services algorithm, a definition for WSDL slicing, and several service evolution metrics.

Firstly, we use fine-grain change mining information based on three change classifications (insertions, deletions, and modifications). This change information aids in extracting various WSDL subsets (i.e., slices), which are combined to construct a

WSDL slice. A WSDL slice is helpful in subset service analysis where it reduces the effort required in regression testing, reverse engineering, and refactoring. The effort is reduced because WSDL slice construction takes only a few milliseconds. The provider can also control the usage of a service with the help of an interface subset (another Subset WSDL), which helps in accessing a subset of a service, and also helps clients focus on the relevant subset of the service's functionalities. A WSDL slice is useful in the study of the impact of a change in one operation on other operations in a distributed system.

Secondly, we also propose four service evolution metrics that are helpful when evaluating the risk associated with an evolving cloud service. These metrics can be helpful in the maintenance of existing versions and the development of future versions. The metrics can also help a consumer while analyzing functionalities of the WSDL, XSD, and WS code. It is also helpful during regression testing of an upgraded cloud service system.

We introduce an evolution analytic model, which exploits the history (e.g., as provided by a version-control system) to reduce the human effort required while analyzing the service evolution over the series of versions. Both approaches are useful for establishing and maintaining Service Level Agreements (SLAs) and Quality of Service (QoS). Service providers often update or enhance a service to meet new requirements, which may cause an SLA violation where an incorrect change leads to incorrect behaviour. Thus, QoS monitoring is required to check and re-establish the specific QoS described in the SLA [21], [22]. Reduced cloud/web service maintenance effort can further reduce the effort required to guarantee the QoS found in the SLA [23], [24]. Regression testing's goal is to maintain QoS and SLA; reducing the regression test suite can save effort, and thus reduce costs [17]–[20].

This article makes the following two contributions, which are presented in the following sections.

1) In Section II, we propose a service change classifier algorithm that performs change mining based on two service versions. The change information is used to perform interface slicing, which results in an interface slice (a subset of the service's interface). In addition, we propose a technique for evolution mining of multiple service versions using four new service evolution metrics.
2) In Section III, we describe a Service Evolution Analytics model and its implementation as a prototype component in an existing tool, AWSCM, which performs change mining and evolution mining of cloud services.

The rest of the article is organized as follows, the Section IV presents experiments that apply AWSCM to real-world evolving distributed systems. Then Section V presents related work. Finally, Section VI concludes the article.

II. Change Mining and Evolution Mining of an Evolving Distributed System

This section describes how we perform *change mining* on two versions and *evolution mining* on a version series of an evolving distributed system. Here a *version series* $\mathrm{VS} = \{V_1, V_2, \ldots, V_N\}$ is a series of version snapshots taken at times $\{t_1, t_2, \ldots, t_N\}$.

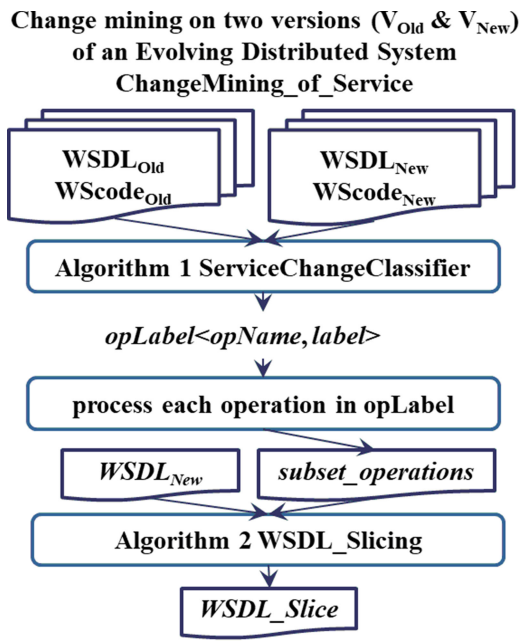

Fig. 1. Summary of the change mining of two versions.

Each version involves two key artifacts: a specification written in a Web Services Description Language (WSDL) and the Web Service (WS) code itself.

The next two sub-sections provide details regarding the two kinds of mining used. The first sub-section describes change classifiers that are applied to two versions of a system. The crux of this step is to classify changes between versions to make an Interface slice. Then the second sub-section introduces the four service evolution metrics, which are used to study the evolution of an evolving distributed system. The crux of this step is to study evolution in a version series.

A. Change Mining of Two Versions: Service Change Classification

The goal of service change classification is to capture the changes between an old and a new version of a distributed system (thus the four parameters of the actual algorithm). We include changes at both the WSDL and the WS code levels.

Our change mining approach has two steps as summarized in Fig. 1. The *ChangeMiningOfService* takes old and new versions of a system as input and first invokes *Algorithm 1, ServiceChangeClassifier*. This algorithm adds the following change labels to the WSDL and WS code of the new version: inserted, deleted, and modified. It then gathers up all operations (as *opName*) that are labeled "*inserted*" or "*modified.*" It omits the operations labeled "*deleted*" because deleted operations are not present in the new WSDL, thus there is no need to retain them in the WSDL Slice. The gathered operations are kept in the set of operations, *subset_operations*. Finally, it invokes *Algorithm 2, WSDL_Slicing,* to construct a subset of the WSDL, referred to as the *Subset WSDL,* which is captured by the WSDL slice returned by our algorithm.

TABLE I
CLOUD SERVICE CHANGE CLASSIFIERS

Label of the change	Level	Purpose at location of change	Captured in Subset WSDL
Insertion I_{WSDL}	WSDL	Insertion of an operation in WSDL or Datatype in XSD	DWSDL
Insertion $I_{WS\ Code}$	WS Code	Insertion of code for new operation	UWSDL
Deletion D_{WSDL}	WSDL	Deletion of operation from WSDL to make its disfunction to client	None. Deletions does not affect, thus ignored [25].
Deletion $D_{WS\ Code}$	WS Code	Deletion of operation from WS code to make disfunction at WSDL	
Modification M_{WSDL}	WSDL	Modification of XSD	DWSDL
Modification $M_{WS\ Code}$	WS Code	Modification at code lines without affecting WSDL	UWSDL

ChangeMiningOfService ($WSDL_{Old}$, $WSDL_{New}$, $WScode_{Old}$, $WScode_{New}$)

1. $opLabels =$ ***ServiceChangeClassifier*** ($WSDL_{Old}$, $WSDL_{New}$, $WScode_{Old}$, $WScode_{New}$)
2. Initialize an empty set *subset_operations*
3. **For each** pair $< opName, label >$ in *opLabels*
   a. **If** $label ==$ "*inserted*" or $label ==$ "*modified*"
   subset_operations $\leftarrow$ *subset_operations* $\cup$ *opName*

End for

4. $WSDL\_Slice =$ ***WSDL_Slicing*** ($WSDL_{New}$, *subset_operations*)

Return *WSDL_Slice*

The first step performs change classification on the WSDL and the WS code. Table I overviews the types of changes identified. Each change includes the level of the change (WSDL or WS code), the purpose of the change, and the WSDL Slice (explained later in this section) that captures the change.

A WSDL description has six major parts [26], which are summarized in Table II. The first part provides the *definitions* of the services provided. The second include the *schema XSD,* which represents the data structures for the input and output of each operation. The third part, *message calling,* supports communication between the client and the operation (we use the term "*operation*" to denote a procedure that is exposed as part of the service's external functionality available to a client). The fourth part provides the *port* used for each operation. Fifth is a *binding* that binds input and output data types to each operation. Finally, the sixth part defines the *service* that is accessed through a URL.

The goal of the first step is to classify the elements of the WSDL and the WS code. A change can be made to any of the following three service elements: the operation code, the messages used for communication, and the input-output parameters. A common approach from data mining used to classify items is to attach *classification labels*. We can therefore identify changed operations by labeling the three kinds of service elements. Step 2 simply gathers all the operations (as *subset_operations*) labeled *inserted* or *modified*.

TABLE II
CHANGES IN WSDL PROPERTIES

Property	Effect of changes on the properties of WSDL
Definition change	Change in the xml version, name of service, xmlns or targetNamespace
XSD Type (schema)	The XSD includes sequence of two kind of data types in the form of XML elements: complex data types and simple data types. Changes may be in Simple Types or Complex Types, which are represented by elements, names, or data type. The simple data types contain only literal text, while the complex data types contain nested attributes and parameters.
Message	Changes may be in the message name of an operation. Changes may also be in the elements and part name of a message.
Port Type change	Change may be in the port type name or operation. Changes may also in the way message input and output were used.
Binding change	Change may be in the binding name and soap binding. Changes may also in the soap body (input - output).
Service change	Changes may be in the service name, binding address, or soap address location.

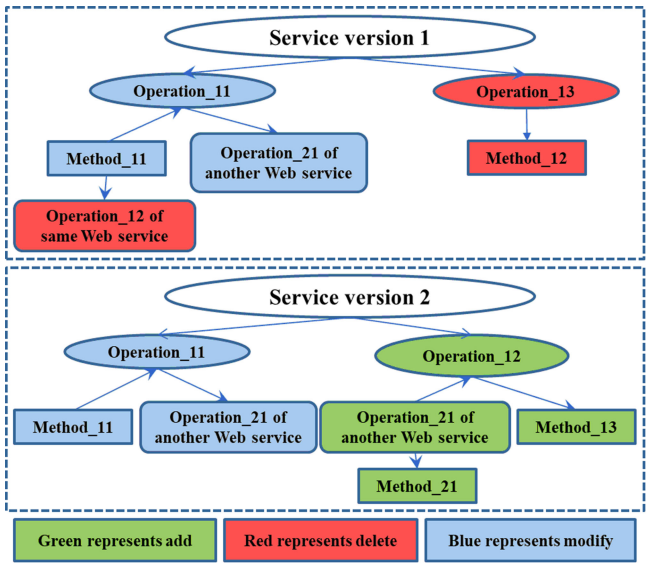

Fig. 2. Simple representation of changes in a service.

The example shown in Fig. 2 illustrates the first two steps. The "Service version 1" is upgraded to make "Service version 2". In "Service version 1," a *modified* "Operation_11" calls *modified* "Method_11" (which further calls "Operation_12," which was *deleted*) and *modified* "Operation_21." Furthermore, "Operation_13" calls "Method_12," which were both *deleted* in

"Service version 2." From the perspective of "Service version 2," *modified* "Operation_11" calls *modified* "Method_11" and *modified* "Operation_21." Further, *inserted* "Operation_12" calls *inserted* "Operation_21" and subsequently *inserted* "Method_21," and *inserted* "Method_13." Note the "Operation_21" is provided by some other service.

Algorithm 1, ServiceChangeClassifier, takes four input artifacts: WSDL_{New} (WSDL of new version), WSDL_{Old} (WSDL of old version), $\text{WScode}_{\text{New}}$ (WS code of new version), and $\text{WScode}_{\text{Old}}$ (WS code of old version). The algorithm involves the following six steps.

1) Identify *semantic WSDL difference* between WSDL_{New} and WSDL_{Old} based on semantic differencing. Store the result in $\text{WSDL}_{\text{Changes}}$.
2) Identify *semantic code difference* between $\text{WScode}_{\text{New}}$ and $\text{WScode}_{\text{Old}}$ based on semantic differencing. Store the result in $\text{WScode}_{\text{Changes}}$.
3) Initialize the list opLabels, which will hold the labels of the differences between the old and new versions. Retrieve the opName_change (from $\text{WSDL}_{\text{Changes}}$ and $\text{WScode}_{\text{Changes}}$), which provides the name of the changed operation along with its corresponding label (insert, delete, and modify).
4) While scanning the $\text{WSDL}_{\text{Changes}}$ for operation changes:
   a) if an operation was added to the new version of the WSDL, then assign it the inserted label in opLabels;
   b) if an operation is deleted from the old version to create the new version of the WSDL, then assign it the deleted label in opLabels;
   c) if an operation from the old version is modified while migrating to the new version of the WSDL, then assign it the modified label in opLabels.
5) While scanning the $\text{WScode}_{\text{Changes}}$ for operation changes:
   a) if any line of code in an operation was added to the new version of the WS code, then assign it the inserted label in opLabels;
   b) if any line of code in an operation is deleted to create new version of the WS code, then assign it the deleted label in opLabels.
   c) If any line of code in an operation is modified while migrating to the new version of the WS code, then assign it the modified label in opLabels.

Note that in this algorithm the order of Steps 4 and 5, which scan the WSDL and the WS code, is important because the second scan may reset a label from the first. Thus, if the $\text{WSDL}_{\text{Changes}}$ analysis is done before the $\text{WScode}_{\text{Changes}}$ analysis, then the WS code analysis takes priority over the WSDL analysis. In contrast, if Step 4 and Step 5 are interchanged, then the WSDL analysis takes precedence. Furthermore, either step may be skipped enabling the approach to be applied separately by a cloud service owner or its consumer.

In the implementation, finding the differences between the two versions of the WS code and the two versions of the WSDL is performed using a semantic differencing tool. The algorithm depends on the accuracy of this tool in its ability to distinguish between the three kinds of changes: insert, delete, and modify.

Algorithm 1: ***ServiceChangeClassifier*** ($WSDL_{Old}$, $WSDL_{New}$, $WScode_{Old}$, $WScode_{New}$)

1. $WScode_{Changes}$ = ***semanticCodeDiff***($WScode_{Old}$, $WScode_{New}$)
2. $WSDL_{Changes}$ = ***semanticWSDLDiff***($WSDL_{Old}$, $WSDL_{New}$)
3. Initialize array list *opLabels*<*opName, label*> with *unchanged* label, and retrieve *opName_change* from $WScode_{Changes}$ and $WSDL_{Changes}$ along with the change labels.
4. **For** each *opName_change* in $WSDL_{Changes}$
   a. **If** (*opName_change* has label *insert*)
      Assign label *inserted* to *opName* in *opLabels*
   b. **If** (*opName_change* has label *delete*)
      Assign label *deleted* to *opName* in *opLabels*
   c. **If** (*opName_change* has label *modify*)
      Assign label *modified* to *opName* in *opLabels*
   End For
5. **For** each *opName_change* in $WScode_{Changes}$
   a. **If** (*opName_change* has label *insert*)
      Assign label *inserted* to *opName* in *opLabels*
   b. **If** (*opName_change* has label *delete*)
      Assign label *deleted* to *opName* in *opLabels*
   c. If (opName_change has label *modify*)
      Assign label *modified* to *opName* in *opLabels*
   End For

Return *opLabels*<*opName*, *label*>

The next step captures the changes between the old and new versions in a WSDL slice with WSDL Slicing define as follows.

Definition: WSDL Slicing is a variation of interface slicing that extracts a subset of a given system's WSDL (as an interface) to capture a subset of the original WSDL's behaviour. The resulting interface slice is referred to as a WSDL slice and contains a subset of WSDL's operations. For example, if a bank service supports two operations "deposit" and "withdraw." A WSDL slice might contain only the "deposit" functionality.

As an interoperable subset of a service, a WSDL slice can aid an engineer (e.g., when testing) by focusing their attention on the relevant subset of the service. Furthermore, the slice of the distributed service's code can provide a similar focus. As seen in Table I, our approach makes use of two WSDL Slices: the Difference WSDL (DWSDL) and the Unit WSDL (UWSDL) [15]–[17]. The DWSDL is constructed based on differences in the WSDL. The UWSDL is constructed based on differences in the WS code.

The WS code differences are obtained from the WS change classification. This process first parses the old and the new versions of the code to separate-out the operations and methods of each, and then identifies differences including the location (file and line) of each change. On one hand, the DWSDL (a subset of the WSDL) is constructed to capture WSDL-level changes. The DWSDL is a type of WSDL Slice. On the other hand, building on the intra-operational and intra-procedural changes, we then identify operations that have inter-operational dependencies on

Algorithm 2: ***WSDL_Slicing*** (*$WSDL_{New}$*, *subset_operations*)

String *cStartDef*, *cXSD*, *cMsg*, *cPort*, *cBinding*, *cService*, *cEndDef*

Array of String[] *$opWSDL_{New}$*

1. [*subset_operations* $\in$ *$opOfWSDL_{New}$* | *subset_operations* = operation required to construct *WSDL_Slice*]
2. *cStartDef* = **sliceStartDefinition**(*$WSDL_{New}$*)
   cXSD = **sliceXSD**(*$WSDL_{New}$*, *subset_operations*)
   cMsg = **sliceMessage**(*$WSDL_{New}$*, *subset_operations*)
   cPort = **slicePort**(*$WSDL_{New}$*, *subset_operations*)
   cBinding = **sliceBinding**(*$WSDL_{New}$*, *subset_operations*)
   cService = **sliceService**(*$WSDL_{New}$*)
   cEndDef = **sliceEndDef**(*$WSDL_{New}$*)
3. *WSDL_Slice* = *cStartDef* + *cXSD* + *cMsg* + *cPort* + *cBinding* + *cService* + *cEndDef*

Return *WSDL_Slice*}

changed operations and methods. These changed operations are used to construct the UWSDL, which is a WSDL Slice that captures changes at the WS code level. This makes the UWSDL useful to access and execute changes in the WS code. Based on the labels in *opLabels*, we capture changes in the following three ways.

1) First, inserted operations are captured in both the DWSDL and the UWSDL.
2) Second, deleted operations are ignored in the *Subset WSDLs* because they are not part of the new version of the service.
3) Third, modified operations occur in two places – in the WSDL and in the WS code. Modifications at the WSDL level mean changes in port, binding, and XSD (input, output, or both) as described in Table II. The *DWSDL* captures these changes. Modifications at the WS code level are captured in the *UWSDL*.

Finally, *Algorithm 2 WSDL_Slicing* takes the WSDL of the new version and the identified subset operations as input; these two are used to construct the *WSDL_Slice* using the functions sliceXSD, sliceMessage etc., which each extract a subset of the WSDL. These subsets (captured as substrings) are combined (concatenated) to form the *WSDL slice*.

A WSDL slice cleanly captures a semantically meaningful subset of a service's functionality. As noted above, it can be computed in the absence of the code. However, as WSDL provides access to the WS code and thus the WSDL slice can also provide access to a reduced version of the code while regression testing.

B. Evolution Mining of a Version Series: Service Evolution Metrics

This sub-section describes evolution mining of a service using the four novel *service evolution metrics*. The metrics are based on five important quantitative attributes: *number of operations*, *WSDL lines*, parameters, messages, and operation code lines. These five attributes are used to generate a quantitative evaluation of a version series VS = $\{V_1 \ldots V_i \ldots V_N\}$ such that V_i represents *i*th version of the service. Using the five attributes, we define four Service Evolution Metrics. The intuition behind the Service Evolution Metrics is to represent cloud service evolution information by aggregating attributes over multiple versions.

1) Effective Lines Per Operation in the WSDL: The first metric is the ratio of number of WSDL lines (noncomment and nonblank) to the number of operations in the WSDL. For version V_i, LOWSDL_i denotes the Lines per Operation in the WSDL_i and is defined as follows:

$$\text{LOWSDLi} = \frac{\text{Number of source lines in WSDL of } V_i}{\text{Number of operations in version } i}.$$

This defines a set as $\text{LOWSDL} = \{(V_1, \text{LOWSDL}_1) \ldots (V_i, \text{LOWSDL}_\text{i}) \ldots (V_N, \text{LOWSDL}_\text{N})\}$.

2) Parameters Per Operation in the WSDL: The second metric is the ratio of the number of parameters in the XSD to the number of operations in the WSDL. This reflects the expected growth in a service's interfaces. It is denoted as PO_i the Parameters per Operation in the WSDL_i for V_i and defined as follows:

$$\text{PO}_i = \frac{\text{Number of parameters in WSDL of } V_i}{\text{Number of operations for version } i}.$$

This defines a set as $\text{PO} = \{(V_1, \text{PO}_1) \ldots (V_i, \text{PO}_i) \ldots (V_N, \text{PO}_N)\}$.

3) Messages Per Operation in the WSDL: The third metric is the ratio of the number of messages to the number of operations in the WSDL. Under normal conditions, the number of messages per operation is two, which reflects having one message for input and one for output. The metric is denoted as MO_i the Messages per Operation in the WSDL_i of V_i and defined as follows:

$$\text{MO}_i = \frac{\text{Number of messages in WSDL of } V_i}{\text{Number of operations for version } i}.$$

This defines a set as $\text{MO} = \{(V_1, \text{MO}_1) \ldots (V_i, \text{MO}_i) \ldots (V_N, \text{MO}_N)\}$.

4) Code Lines Per Operation in the WS Code: The fourth metric is based on the logic behind the WSDL. It is the ratio of the number of lines of code to the number of operations. This reflects the growth of service operations in terms of business logic. It is denoted as WSCLO_i the WS Code Line per Operation in WSCode_i of V_i

$$\text{WSCLO}_i = \frac{\text{Number of code lines in WS code of } V_i}{\text{Number of operations in version } i}.$$

This defines a set as $\text{WSCLO} = \{(V_1, \text{WSCLO}_1) \ldots (V_i, \text{WSCLO}_i) \ldots (V_N, \text{WSCLO}_N)\}$.

These four novel metrics can be expressed together as $(V_i, \text{LOWSDL}_i, \text{PO}_i, \text{MO}_i, \text{WSCLO}_i)$. With respect to the versions, these four metrics are used to create four time series graphs that are useful in the study of the evolving distributed systems. The evolution mining of a version series using Service Evolution Metrics is summarized in Fig. 3.

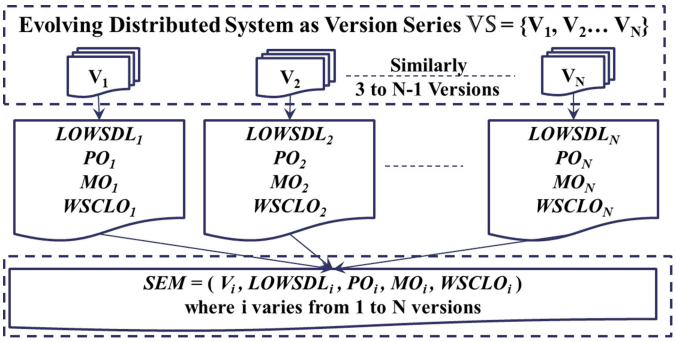

Fig. 3. Summary of the evolution mining of version series based on service evolution metrics (SEM).

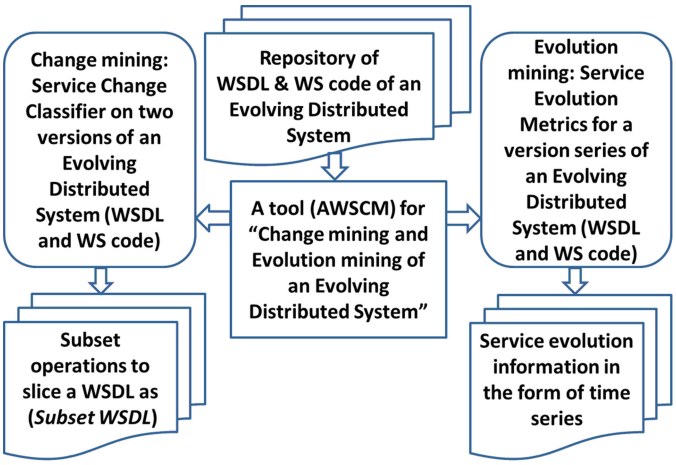

Fig. 4. Service evolution analytics model.

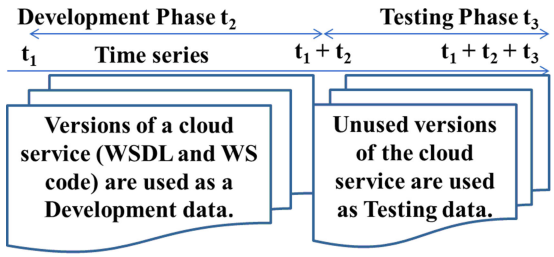

Fig. 5. Development and testing phases of the service evolution analytics tool. Assuming starting at time t_1 the *development phase* runs for time t_2. Let the subsequent testing phase starts at time $t_1 + t_2$ and runs for time t_3. Therefore, both the phases end at time $t_1 + t_2 + t_3$.

III. Service Evolution Analytics

We now describe our *Service Evolution Analytics* model and tool. The model, overviewed in Fig. 4, is based on the WSDL and the WS code of a version series. As illustrated in Fig. 5, tool construction includes a development phase (duration t_2) and a subsequent testing phase (duration t_3).

During the development phase, the tool learns from a prefix of the service's version series. A supervised learning approach is used: if the output is incorrect, the developer updates the tool's code and then recomputes the interface slice. Output correctness can be judged using two possible criteria: *software acceptance* and *human acceptance.* For software acceptance, an IDE (e.g., NetBeans and Eclipse) or testing framework (e.g., SoapUI and JMeter) must accept a subset of the WSDL as an interface slice. For human acceptance, an engineer determines if the tool's output is satisfactory. While generating a slice can be a complex task, a developer (or a tester) should be able to easily identify correct output. In this way, the tool learns to produce the correct interface slice from its developer. The process terminates when the IDE or testing framework accepts the tool's output as the desired interface slice. The interface (WSDL) slice should have the desired structure for supporting communication between its operations and the WS code. Both an IDE and a testing framework can exploit the WSDL slice in the development and testing of the WS code.

Next, in the testing phase the resulting tool is tested using the remainder of the service's version sequence. The performance of the tool is measured by comparing its output with the expected output, which includes the *WSDL (interface) slices*.

*AWSCM***:** Using this model, we incrementally developed a new slicing component of the tool AWSCM (Automated Web Service Change Management) [16]. AWSCM is an automated intelligent tool that seeks to understand the structure of the WSDL and the WS code. It slices the WSDL by performing change mining of evolving web service, which constructs a *WSDL slice*. It uses the Membrane SOA Model [60] to find semantic differences between two versions of the WSDL. To find the semantic differences between two versions of the WS code, it uses jDiff [59]. As mentioned in [57], [58], empirically jDiff provides correct coverage information 84.70% for the single test case, and 77.81% for the entire test suite. To incorporate these semantic differences, we developed several functions, e.g., a function for operation separation and a function that slices out portions of the WSDL. AWSCM's slicing code was repeatedly updated until the tool was able to independently generate the correct WSDL slice for each system studied in the next section. AWSCM was also updated to compute the four proposed service evolution metrics defined in Section II-B. This new feature was implemented in a new module named "Service Evolution Metrics."

TABLE III
Subset WSDL for Change Analysis

Web services	DWSDL	UWSDL	Available at
SaaS	Y	Y	Self-made
BookService	Y	Y	Self-made
Eucalyptus	Y	Y	GithHub
AWS	Y	N	GithHub

IV. Experiments on Distributed Systems

This section describes three experiments, which consider two self-made illustrative web services and two real-world web services (see Table III). The first subsection presents use of

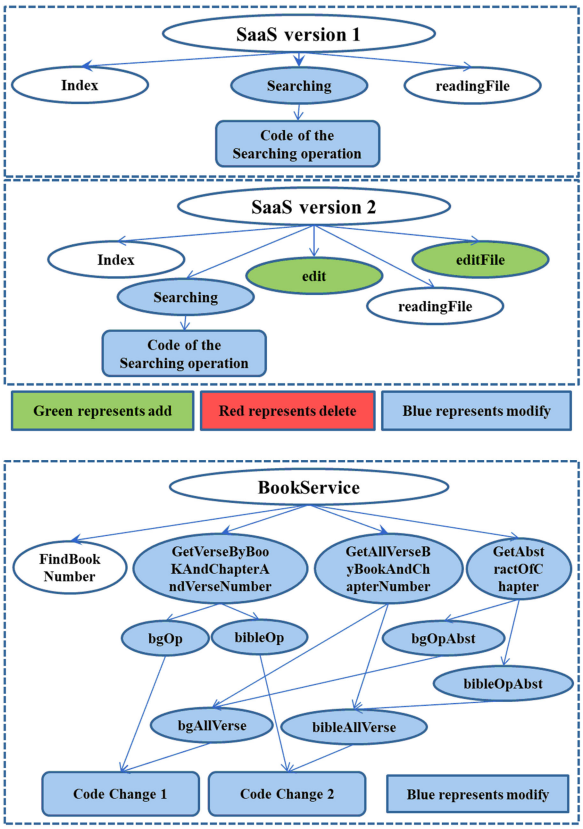

Fig. 6. Simple representation of the changes in two self-made web services, SaaS, and BookService.

TABLE IV
CHANGES BETWEEN TWO SUBSEQUENT WSDLS

Example for Eucalyptus-Cluster Controller (CC)
Feb-13.Eucalyptus-CC.wsdl ➔ Aug-13.Eucalyptus-CC.wsdl
Operation Change: ModifyNode inserted; MigrateInstances inserted.
Schema Change: CT ccInstanceType has modified; CT virtualBootRecordType has modified; CT metricCounterType has modified; CT metricDimensionsType has modified.
May-15.Eucalyptus-CC.wsdl ➔ Dec-16.Eucalyptus-CC.wsdl
Operation Change: DescribePublicAddresses deleted; AttachNetworkInterface inserted; DetachNetworkInterface inserted;
Schema Change: CT DetachVolumeResponseType inserted; CT ccInstanceType has modified; CT netConfigType has modified;

Example for AWS-Elastic Compute Cloud (EC2)
Feb-13.ec2.wsdl ➔ Aug-13.ec2.wsdl
Operation Change: DescribeReservedInstancesModifications inserted; ModifyReservedInstances inserted.
August-13.ec2.wsdl ➔ Oct-13.ec2.wsdl
Operation Change: AcceptVpcPeeringConnection inserted; CreateVpcPeeringConnection inserted; DeleteVpcPeeringConnection inserted; DescribeVpcPeeringConnections inserted; RejectVpcPeeringConnection inserted.

*where CT stands for ComplexType input-output data structure
*for inserted operations, it is obvious that their input-output data-structure were also inserted, thus we skipped its details.
*for deleted operations, it is obvious that their input-output data-structure were also deleted, thus we skipped its details.

change classification in the WSDL slice construction. The second subsection presents empirical results demonstrating the potential reduction in regression testing cost. The third subsection presents experiments involving the Service Evolution Metrics.

A. WSDL Slice Construction

Using Algorithms 1, 2, and 3, the first experiment computes a WSDL slice. It applies change mining to the web service to compute the change classification. It then uses this change classification to construct the resulting WSDL slice. The WSDL Slices produced for each system considered are shown in Table III. As described in Section II-A, we consider two Subset WSDLs: the *Difference WSDL* (DWSDL) and the *Unit WSDL* (UWSDL). In the table a "Y" indicates that the Subset WSDL was constructed, while the single "N" indicates the one case in which the source was not available to us.

Fig. 6 shows the changes that we made to the two web services: SaaS and BookService. The figure demonstrates the effects of changes in the form of dependency graphs of the two web services. The changes involved are as follows.

1) SaaS version 1 has three operations: Index, Searching and readingFile. We made following two changes to version 1 to create version 2. First, we inserted two new operations: edit and editFile, which are captured by the DWSDL. Second, we modified the code of the Searching operation, which is captured by the UWSDL.
2) BookService version 1 is upgraded to version 2 with two modifications at the code level that affect the following operations. Code Change 1 affects the two operations: bgAllVerse and bgOp, which further affect their parent nodes in the graph. Code Change 2 affects the two operations: bibleAllVerse and bibleOp, which further affected their parent nodes in the graph. Overall, the two code changes affect three out of four BookService operations.

Table IV provides two examples of the three classification labels (inserted, deleted, and modified) for the two large-scale evolving distributed systems based cloud services: Eucalyptus Cluster Controller (CC) and the Amazon Web Service Elastic Compute Cloud (AWS-EC2). Looking at the output it is clear that the accuracy of the change detection is dependent on the semantic differencing tool; thus AWSCM depends on the accuracy of semantic differencing tool that it uses.

B. Service Maintenance: Reduced Regression Testing

Table V describes four experiments that consider the retrieval of WSDL Slices as well as their use in test-case reduction. For SaaS, we performed two experiments. In the first the DWSDL was created based on two operations, and in the second the UWSDL was created with one operation. Test case reduction for the two was 50% and 75% respectively. For BookService, we did two experiments using the two services: *BGWS* and BibileWS. Here AWSCM uses two changed operations from both BGWS

TABLE V
EXPERIMENTS FOR CHANGE MINING-BASED WSDL SLICES AND TEST CASE RETRIEVAL

Service	Test Cases (TCs) retrieval	WSDL Slice contains	TCs reduction in %
SaaS	2 out of 5 operations are added, the tool retrieves 2 of 4 TCs.	DWSDL has 2 Operations	50%
	1 out of 5 operations was changed in WS code. Thus, 1 out of 4 TCs is retrieved.	UWSDL has 1 Operation	75%
BookService	3 out of 4 operations in *BGWS* are dependent on the two changes. Thus, 2 out of 5 test steps are retrieved.	UWSDL has 3 Operations	60%
	3 out of 4 operations in *BibleWS* is dependent on the two changes. Thus, 2 out of 5 test cases are retrieved.	UWSDL has 3 Operations	60%
AWS	3 out of 23 operations are selected leading to retrieving 3 out of 23 TC templates.	DWSDL has 3 Operation	86.95%
EucalyptusCC	2 operations are added leading to 2 out of 26 TC templates being retrieved.	DWSDL has 2 Operation	92.31%
	3 out of 26 WS code operations are changed leading to 3 out of 26 TC templates being retrieved.	UWSDL 3 has Operation	88.5%

and BibleWS to retrieve two different regression test suites each containing 60% of the original tests.

For simplicity when explaining the remaining two experiments, which used *AWS* and *EucalyptusCC*, we assume that there is one test case per operation. In AWS, when the user selects 3 of its 23 operations, there is an 86.95% reduction in the size of the regresion test suite. In *EucaluptusCC*, two of the operations were changed in two versions of the Eucalyptus WSDL. Using the resulting DWSDL that captures these two changes, AWSCM reduces the size of the test suite by 92.31%. AWSCM retrieved three operations as changed at the code level for *EucalyptusCC*. Thus, producing an 88.5% reduction in the number of test cases.

C. Service Evolution Metric Study

We present a study of the two cloud services that are two large-scale evolving distributed systems: Eucalyptus Cluster Controller (Eucalyptus-CC) and Amazon Web Service – Elastic Compute Cloud (AWS-EC2). To study our four service evolution metrics, we use the repositories available from Github [61]–[63]. The number of lines, parameters, lines of WS code, and WSDL operations are given in Table VI. The counts in the table are used to calculate the service evolution metrics whose values are shown in Fig. 7.

We deduce the following from Fig. 7. First, the number of lines per operation in WSDL of Eucalyptus-CC is slightly larger than that of AWS-EC2. In contrast, AWS-EC2 has more parameters per operation as compared to Eucalyptus-CC. Both projects have two messages per operations for all versions. Eucalyptus-CC has almost 200 lines of code per operation, which is found in the file *handlers.c* inside module "cluster."

To keep experimental result simple, external code dependencies are not considered. All the graphs suggest that both projects are growing at a steady pace. In general, the metric values can help a manager to take decisions regarding a cloud service evolution task such as regression testing and service monitoring.

In a time series graph of a service evolution metric, a sudden change (glitch) may denote an anomaly such as an incorrect upgrade. In this case, the project manager can go for validating the reason of the anomaly, which can help to evaluate the project risk. Therefore, in Fig. 7, all the glitches in the time series graphs are the point of deviation from normal trend.

For example, consider the following three changes shown in Table VI. First, in August of 2006, public beta of Amazon EC2 was released.[1] This is reflected in the first row (June-06). In October of 2007, two new instance types were added[2] to EC2, Large and Extra-Large. Thereafter, in May of 2008, EC2 added[3] two more instance types, High-CPU Medium and High-CPU Extra Large. These four additions are reflected in Table VI in the second row (January-07), third row (Feb-08), fourth row (July-09), and fifth row (November-09). These changes are evident in the PO_i time series of AWS-EC2 shown in Fig. 7. To begin with the initial point at (June-06). Second is the almost constant growth between (January-07) and (Feb-08). Third is the sudden increase (glitch) from (Feb-08) to (July-09) and (November-09).

In summary, the change classification enables AWSCM to construct a WSDL slice that captures the operations changed from an old to a new version. The evolution mining provides information about an entire version series, which helps to compare the two services empirically. Collectively, the two mining techniques retrieve different information, which helps in service evolution analysis.

V. RELATED WORK AND DISCUSSION

This section discusses related contributions. To begin with, work related to *software analysis* [27] and *mining software repositories* [28] in the context of *change impact analysis* includes the following. For example, Ying *et al*. [29] presented an approach to determine change patterns and evaluated their work on the code repositories for Eclipse and Mozilla. Likewise, we evaluate our work by analyzing the repositories of *Eucalyptus-CC* and *AWS-EC2*. More recently, Negara *et al*. [30] identified previously unknown frequent code change patterns from an old repository. Thereafter, they analyzed some of the mined unknown code change patterns and some high-level program transformations.

[1][Online]. Available: Jeff Barr, (August 25, 2006). "Amazon EC2 Beta". https://aws.amazon.com/blogs/aws/amazon_ec2_beta/. Link accessed on February 2019.

[2]Jeff Barr, (October 16, 2007). "Amazon EC2 Gets More Muscle". Link accessed on February 2019.

[3][Online]. Available: Jeff Barr, (May 29, 2008). "More EC2 power". https://aws.amazon.com/blogs/aws/more-ec2-power/. Link accessed on February 2019.

TABLE VI
INFORMATION ABOUT EUCALYPTUS-CC AND AMAZON WEB SERVICE-EC2 TO CALCULATE SERVICE EVOLUTION METRICS

Time Sequence	Eucalyptus Cluster Controller (CC)						AWS-Elastic Compute Cloud (EC2)		
Time for last commit	Version	# Commits	# Lines of WSDL	# Parameter	# Code lines	# Operations	# Lines of WSDL	# Parameter	# Operations
June-06	The project was not published on GitHub during this period.						865	96	14
January-07							1157	130	19
Feb-08							1560	181	26
July-09							3568	523	65
November-09							4565	714	81
March-10	1.0	2612	912	50	3272	15	Data is not available for these months.		
Feb-09	1.5	512	848	45	2086	14			
June-10	2.0	4054	1087	59	3663	18	4077	762	87
Nov-10	Data is not available for these months.						4536	890	95
Nov-11							5637	1116	119
Feb-12	3.0	13647	1464	83	4822	24	Data is not available for this month.		
June-12	3.1.0	14252	1684	101	4870	28	6513	1359	137
August-12	3.1.1	14310	1465	84	4870	24	7021	1455	144
Feb-13	3.2.1	16770	1580	98	5606	26	7252	1501	149
August-13	3.3.1	19974	1684	101	6438	28	7392	1535	151
Oct-13	3.4	20621	1783	102	6672	30	Data is not available for this month.		
Feb-14	Data is not available for this month.						7612	1579	156
May-14	4.0	22130	1836	102	6869	31	Data is not available for these months.		
May-15	4.1.1	23208	1840	104	7066	31			
Dec-16	4.3.1		1905	105	6346	32			

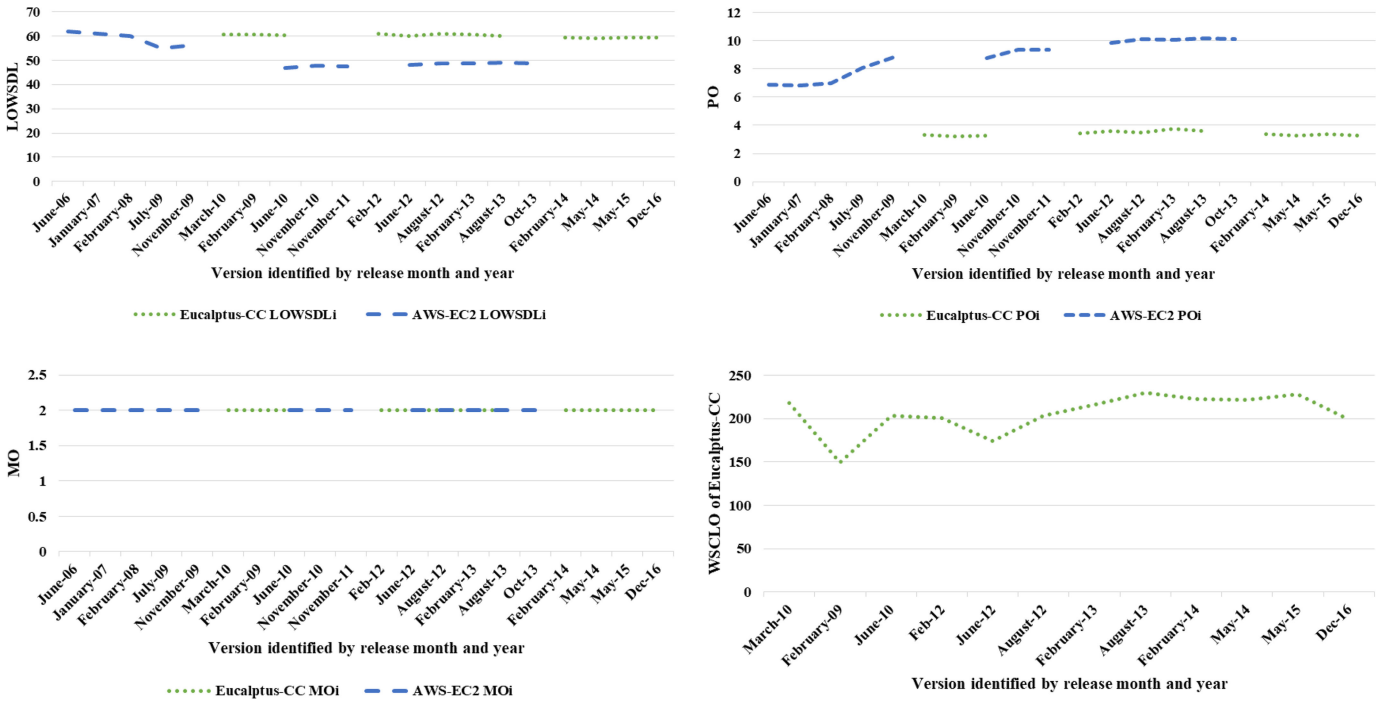

Fig. 7. Four time-series graphs to show four service evolution metrics for AWS-EC2 and Eucalyptus-CC.

Work related to change analysis, mining, and learning of an evolving service repository includes that of Xiao *et al*. [31] who proposed a formula to quantify the cost of a business process change in a service-oriented architecture. Dam *et al*. [32] analyzed change propagation using extensions of the well-known UML framework in SOA as SoaML. Papazoglou *et al*. [33], [34] present a theoretical framework for controlling the evolution of services using distinguished shallow and deep changes. Alam *et al*. [35] presented a literature survey of 60 relevant studies of change propagation in SOA and Business Process Management (BPM) based on research questions. Romano and Martin [36] proposed a tool WSDLDiff for gathering fine-grained changes between subsequent versions of a service; then this information is used to analyze the evolution of four web services. Recently, Karn *et al*. [37] proposed a methodology for the selection and tuning of a machine-learning model, which is applied to dynamic changing environments for achieving higher accuracy in prediction of security attacks.

Intuitively, our approach is useful in both static and dynamic program analysis for testing and verification [38]. For example, in support of mobile app testing [39], Facebook researchers have developed CT-Scan [40], which can also aid in regression testing [41]. Li *et al*. [42] formalized the semantic slicing of software versions histories and proposed CSLICER to identify the set of semantically related commits. An application of change impact analysis based interface slicing, such as WSDL slicing, was used in regression testing of web services [17].

The work related to software and service evolution metrics includes the following. Constantinou *et al*. [43] presented six metrics to measure the architectural stability and evolution of software projects for reuse. Wang *et al*. [44] proposed a service evolution model based on four service evolution patterns used to analyze service dependencies, identify changes on services, and estimate impact on consumers. Recently, Kohar *et al*. [45] proposed a service evolution metric, a service client-code evolution metric, and service usefulness evolution metric. Similarly, we consider service evolution analysis using the four novel service evolution metrics, which produced useful interpretations (e.g., see Fig. 7 and Table VI).

Fokaefs *et al*. [46] presented WSDL evolution analysis through the empirical study of their VTracker algorithm for XML differencing. The algorithm can recognize and analyze changes between subsequent versions of various WSDL web-services. Thereafter, Fokaefs *et al*. [47] introduced WSDarwin, a specialized differencing method for WSDL and WADL (Web Application Description Language), which produces a set of rules. Li *et al*. [48] reported an empirical study on web API evolution and then drew conclusions for application developers. In contrast to these works, our tool AWSCM can use change information to generate interface (WSDL) slices, which can execute subsets of a cloud/web service. Additionally, AWSCM can also retrieve cloud/web service evolution information that can lead to the extraction of facts about the evolution of an evolving distributed system.

Mohamed *et al*. [49] presented SaaS evolution platform based on Software Product Lines (SPLs) for a multitenant single instance; the platform provides evolution rules based on feature modeling to govern evolution decisions. Jamshidi *et al*. [50] reviewed cloud migration research and include a comparative study between SOA and Cloud migration. Ghosh *et al*. [51] proposed SelCSP framework to facilitate selection of a trustworthy and competent service provider. Recently, Noor *et al*. [52] described a trust management framework named as CloudArmor, which is based on reputation and credibility model. Similarly, our service evolution metrics generate empirical reports that may be helpful in the Cloud/SOA migration and in the selection of a trustworthy service provider. Our approach can also be helpful in the change and evolution maintenance of services supporting Blockchain management [53], [54] and Cloud management [55], [56].

VI. Conclusion

This article described approaches for change mining and evolution mining of an evolving distributed system. The first approach used a *service change classifier* algorithm to assign change labels to a service's operations and then extracts a *WSDL slice*. The second approach used four *service evolution metrics* to study cloud service evolution and to deduce facts from a version series. This article also described a Service Evolution Analytics model and tool (AWSCM) that supported change and evolution mining of an evolving distributed system. We performed four case studies to construct WSDL slices of four web services. We also demonstrated the service change classifier, our tools ability to reduce regression testing cost, and service evolution metrics using two well-known cloud services: Eucalyptus-CC and AWS-EC2. The results demonstrated how the tool can be used to study the evolution of cloud services. Our Service Evolution Analytics model was also helpful in subset regression testing, which in turn helps to maintain the QoS required by a SLA. In the future, we plan to apply change and evolution mining to other APIs.

References

[1] S. A. Frost and M. J. Balas, "Evolving systems and adaptive key component control," in *Aerospace Technologies Advancements*. Norderstedt, Germany: Books on Demand, 2010, ch. 7, p. 115.

[2] T. Mens, A. Serebrenik, and A. Cleve, *Evolving Software Systems*. Berlin, Germany: Springer, 2014.

[3] M. Efatmaneshnik, S. Shoval, and L. Qiao, "A standard description of the terms module and modularity for systems engineering," *IEEE Trans. Eng. Manage.*, vol. 67, no. 2, pp. 365–375, May 2020.

[4] E. Fricke and A. P. Schulz, "Design for changeability (DfC): Principles to enable changes in systems throughout their entire lifecycle," *Syst. Eng.*, vol. 8, no. 4, pp. 342–359, 2005.

[5] A. M. Ross, D. H. Rhodes, and D. E. Hastings, "Defining changeability: Reconciling flexibility, adaptability, scalability, modifiability, and robustness for maintaining system lifecycle value," *Syst. Eng.*, vol. 11, no. 3, pp. 246–262, 2008.

[6] M. A. Chaumun, H. Kabaili, R. K. Keller, F. Lustman, and G. Saint-Denis, "Design properties and object-oriented software changeability," in *Proc. 4th Eur. Conf. Softw. Maintenance Reeng.*, 2000, pp. 45–54.

[7] H. P. Breivold, I. Crnkovic, and P. J. Eriksson, "Analyzing software evolvability," in *Proc. 32nd Annu. IEEE Int. Comput. Softw. Appl. Conf.*, 2008, pp. 327–330.

[8] S. V. Loo, G. Muller, T. Punter, D. Watts, P. America, and J. Rutgers, "The Darwin project: Evolvability of software-intensive systems," in *Proc. IEEE Int. Workshop Softw. Evolvability*, 2007, pp. 48–53.

[9] M. Böttcher, F. Höppner, and M. Spiliopoulou, "On exploiting the power of time in data mining," *ACM SIGKDD Explorations Newslett.*, vol. 10, no. 2, pp. 3–11, 2008.

[10] M. Boettcher, "Contrast and change mining," *Wiley Interdisciplinary Rev.: Data Mining Knowl. Discovery*, vol. 1, no. 3, pp. 215–230, 2011.

[11] M. Takaffoli, F. Sangi, J. Fagnan, and O. R. Zäıane, "Community evolution mining in dynamic social networks," *Procedia—Social Behavioral Sci.*, vol. 22, pp. 49–58, 2011.

[12] B. Wu, X. Pei, J. Tan, and Y. Wang, "Resume mining of communities in social network," in *Proc. 7th IEEE Int. Conf. Data Mining Workshops*, 2007, pp. 435–440.

[13] T. Srivastava, P. Desikan, and V. Kumar, "Web mining—Concepts, applications and research directions," in *Foundations and Advances in Data Mining*. Berlin, Germany: Springer, 2005, pp. 275–307.

[14] H. Chen and M. Chau, "Web mining: Machine learning for web applications," *Annu. Rev. Inf. Sci. Technol.*, vol. 38, pp. 289–330, 2004.

[15] A. Chaturvedi, "Subset WSDL to access subset service for analysis," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, 2014, pp. 688–691.

[16] A. Chaturvedi, "Automated web service change management AWSCM—A tool," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, 2014, pp. 715–718.

[17] A. Chaturvedi and D. Binkley, "Web service slicing: Intra and inter-operational analysis to test changes," *IEEE Trans. Serv. Comput.*, to be published, doi: 10.1109/TSC.2018.2821157.

[18] A. Chaturvedi, "Change impact analysis based regression testing of web services," 2014, *arXiv:1408.1600*.

[19] A. Chaturvedi and A. Gupta, "A tool supported approach to perform efficient regression testing of web services," in *Proc. IEEE 7th Int. Symp. Maintenance Evol. Serv.-Oriented Cloud-Based Syst.*, 2013, pp. 50–55.

[20] A. Chaturvedi, "Reducing cost in regression testing of web service," in *Proc. CSI 6th Int. Conf. Softw. Eng.*, 2012, pp. 1–9.

[21] Y. Du, L. Wang, G. Mu, and X. Li, "Dynamic monitoring of service outsourcing for timed workflow processes," *IEEE Trans. Eng. Manage.* , vol. 66, no. 4, pp. 715–729, Nov. 2019.

[22] Z. Li, G. Nan, and M. Li, "Advertising or freemium: The impacts of social effects and service quality on competing platforms," *IEEE Trans. Eng. Manage.* , vol. 67, no. 1, pp. 220–233, Feb. 2020.

[23] G. Canfora and M. Di Penta, "Testing services and service-centric systems: Challenges and opportunities," *IT Professional*, vol. 8, pp. 10–17, 2006.

[24] D. S. Linthicum, "The evolution of cloud service governance," *IEEE Cloud Comput.*, vol. 2, no. 6, pp. 86–89, Nov./Dec. 2015.

[25] D. Binkley, "The application of program slicing to regression testing," *Inf. Softw. Technol.*, vol. 40, no. 11/12, pp. 583–594, 1998.

[26] R. Chinnici, J.-J. Moreau, A. Ryman, and S. Weerawarana, "Web services description language (WSDL) version 2.0 part 1: Core language," W3C Recommendation, Jun. 2007. [Online]. Available:https://www.w3.org/TR/wsdl20/

[27] D. Binkley, "Source code analysis: A road map," in *Proc. Future Softw. Eng.*, 2007, pp. 104–119.

[28] A. E. Hassan, "The road ahead for mining software repositories," in *Proc. Frontiers Softw. Maintenance*, 2008, pp. 48–57.

[29] A. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll, "Predicting source code changes by mining change history," *IEEE Trans. Softw. Eng.*, vol. 30, no. 9, pp. 574–586, Sep. 2004.

[30] S. Negara, M. Codoban, D. Dig, and R. E. Johnson, "Mining fine-grained code changes to detect unknown change patterns," in *Proc. 36th Int. Conf. Softw. Eng.*, 2014, pp. 803–813.

[31] H. Xiao, J. Guo, and Y. Zou. "Supporting change impact analysis for service oriented business applications," in *Proc. Int. Workshop Syst. Develop. SOA Environ.*, 2007, p. 6.

[32] D. H. Khanh and A. Ghose, "Supporting change propagation in the maintenance and evolution of service-oriented architectures," in *Proc. Asia Pacific Soft. Eng. Conf.*, 2010, pp. 156–165.

[33] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, "Managing evolving services," *IEEE Softw.*, vol. 28, no. 3, pp. 49–55, May/Jun. 2011.

[34] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou, "On the evolution of services," *IEEE Trans. Softw. Eng.*, vol. 38, no. 3, pp. 609–628, May/Jun. 2012.

[35] K. A. Alam, R. Ahmad, A. Akhunzada, M. H. N. Md Nasir, and S. U. Khan, "Impact analysis and change propagation in service-oriented enterprises: A systematic review," *Inf. Syst.*, vol. 54, pp. 43–73, 2015.

[36] D. Romano and M. Pinzger, "Analyzing the evolution of web services using fine-grained changes," in *Proc. IEEE 19th Int. Conf. Web Serv.*, 2012, pp. 392–399.

[37] R. Karn, P. Kudva, and I. M. Elfadel, "Dynamic autoselection and autotuning of machine learning models for cloud network analytics," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 5, pp. 1052–1064, May 2019.

[38] M. Harman and P. O'Hearn, "From start-ups to scale-ups: Open problems and challenges in static and dynamic program analysis for testing and verification (keynote paper)," in *Proc. IEEE 18th Int. Working Conf. Source Code Anal. Manipulation*, 2018, pp. 1–23.

[39] J. Dunn, A. Mols, L. Lomax, and P. Medeiros, "Managing resources for large-scale testing," May 24, 2017. [Online]. Available:https://code.facebook.com/posts/1708075792818517/managing-resourcesfor-large-scale-testing/

[40] A. Reversat, "The mobile device lab at the Prineville data center," Jul. 13, 2016. [Online]. Available: https://code.facebook.com/posts/300815046928882/the-mobile-device-lab-at-the-prineville-data-center/

[41] Z. Mi, "Mobile performance: Tooling infrastructure at facebook," Apr. 10, 2015. [Online]. Available: https://code.facebook.com/posts/924676474230092/mobile-performance-tooling-infrastructure-at-facebook/

[42] Y. Li, C. Zhu, J. Rubin, and M. Chechik, "Semantic slicing of software version histories," *IEEE Trans. Softw. Eng.*, vol. 44, no. 2, pp. 182–201, Feb. 2018.

[43] E. Constantinou and I. Stamelos, "Architectural stability and evolution measurement for software reuse," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, 2015, pp. 1580–1585.

[44] S. Wang, W. A. Higashino, M. Hayes, and M. A. M. Capretz, "Service evolution patterns," in *Proc. IEEE Int. Conf. Web Serv.*, 2014, pp. 201–208.

[45] R. Kohar and N. Parimala, "A metrics framework for measuring quality of a web service as it evolves," *Int. J. Syst. Assurance Eng. Manage.*, vol. 8, no. 2, pp. 1222–1236, 2017.

[46] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and Alex Lau, "An empirical study on web service evolution," in *Proc. IEEE Int. Conf. Web Serv.*, 2011, pp. 49–56.

[47] M. Fokaefs and E. Stroulia, "WSDarwin: Studying the evolution of web service systems," in *Advanced Web Services*, New York, NY, USA: Springer, 2014, pp. 199–223.

[48] J. Li, Y. Xiong, X. Liu, and L. Zhang, "How does web service API evolution affect clients?" in *Proc. IEEE 20th Int. Conf. Web Serv.*, 2013, pp. 300–307.

[49] F. Mohamed, M. Abu-Matar, R. Mizouni, M. Al-Qutayri, and Z. A. Mahmoud, "SAAS dynamic evolution based on model-driven software product lines," in *Proc. IEEE 6th Int. Conf. Cloud Comput. Technol. Sci.*, 2014, pp. 292–299.

[50] P. Jamshidi, A. Ahmad, and C. Pahl, "Cloud migration research: A systematic review," *IEEE Trans. Cloud Comput.*, vol. 1, no. 2, pp. 142–157, Jul.–Dec. 2013.

[51] N. Ghosh, S. K. Ghosh, and S. K. Das, "SelCSP: A framework to facilitate selection of cloud service providers," *IEEE Trans. Cloud Comput.*, vol. 3, no. 1, pp. 66–79, Jan./Mar. 2015.

[52] T. H. Noor, Q. Z. Sheng, L. Yao, S. Dustdar, and A. H. H. Ngu, "CloudArmor: Supporting reputation-based trust management for cloud services," *IEEE Trans. Parallel Distrib. Syst.*, vol. 27, no. 2, pp. 367–380, Feb. 2016.

[53] G. Fortino, F. Messina, D. Rosaci, and G. M. L. Sarne, "Using blockchain in a reputation-based model for grouping agents in the internet of things," *IEEE Trans. Eng. Manage.*, to be published, doi: 10.1109/TEM.2019.2918162.

[54] M. Kuperberg, "Blockchain-based identity management: A survey from the enterprise and ecosystem perspective," *IEEE Trans. Eng. Manage.*, to be published, doi: 10.1109/TEM.2019.2926471.

[55] S. Singh and S. C. Misra, "Exploring the challenges for adopting the cloud PLM in manufacturing organizations," *IEEE Trans. Eng. Manage.*, to be published, doi: 10.1109/TEM.2019.2908454.

[56] R. C. Basole and H. Park, "Interfirm collaboration and firm value in software ecosystems: evidence from cloud computing," *IEEE Trans. Eng. Manage.*, vol. 66, no. 3, pp. 368–380, Aug. 2019.

[57] T. Apiwattanapong, A. Orso, and M. J. Harrold, "JDiff: A differencing technique and tool for object-oriented programs," *J. Autom. Soft. Eng.*, vol. 14, no. 1, pp. 3–36, Mar. 2007.

[58] T. Apiwattanapong, A. Orso, and M. J. Harrold, "A differencing algorithm for object-oriented programs," in *Proc. 19th IEEE Int. Conf. Autom. Softw. Eng.*, 2004, pp. 2–13.

[59] "JDiff," Oct. 2019. [Online]. Available: http://javadiff.sourceforge.net/

[60] "Membrane SOA Model," Mar. 2017. [Online]. Available: http://membrane-soa.org/soa-model/

[61] "AWS EC2," Mar. 2017. [Online]. Available: https://github.com/chilts/awssum-amazon-ec2

[62] "AWS EC2 Release," Mar. 2017. [Online]. Available: https://aws.amazon.com/releasenotes/Amazon-EC2

[63] "Eucalyptus," Mar. 2017. [Online]. Available: https://github.com/eucalyptus/eucalyptus

Animesh Chaturvedi received the B.Eng. degree from IET, Devi Ahilya University, Madhya Pradesh, India, in 2011, and the M.Tech. degree from the Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, India, in 2014. He is currently working toward the Ph.D. degree with the Indian Institute of Technology, Indore, India.

He is a Postdoctoral Research Assistant in NetSys Group of King's College London, London, U.K., and working with The Alan Turing Institute, London, United Kingdom. He did research work with IIT-Kanpur, Motorola, and Arris. His research interest is in data mining, machine learning, and evolving systems.

He attended events including 29th IEEE International Conference on Software Maintenance 2013, IEEE CloudCom 2014, 36th IEEE/ACM International Conference on Software Engineering 2014, and Heidelberg Laurate Forum (HLF) 2019. He has been reviewer for IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTING, IEEE TRANSACTIONS ON BIG DATA, and IEEE Systems Journal.

Aruna Tiwari received the B.Eng. and M.Eng. degrees in computer engineering from SGSITS Indore, India, and the Ph.D. degree in computer science and engineering from RGPV Bhopal, India, in 1994, 2000, and in 2009, respectively.

She is an Associate Professor of Computer Science and Engineering at Indian Institute of Technology, Indore, India, since 2012. Her research interests include soft computing, neural network, fuzzy clustering, evolutionary computation, genome analysis, and health-care applications. Her research group is working on these techniques to accomplish goals of data mining, machine learning, big data analytics, and hardware realization. She has many publications in peer-reviewed journals (of IEEE Transactions, Elsevier, and Springer), international conferences, and book chapters. She has research collaboration with CSIR CEERI Pilani and Indian Institute of Soyabean Research (Indian Council of Agriculture & Research). She has been reviewer for many reputed journals and conferences.

Dave Binkley received a bachelor's in computer engineering from Case Western Reserve University, Cleveland, Ohio, and the doctorate degree from the University of Wisconsin–Madison, in 1991.

He is a Professor of Computer Science at Loyola University Maryland, Baltimore, United States where he has worked since receiving his doctorate. He has been a Visiting Faculty Researcher at the National Institute of Standards and Technology (NIST), worked with Grammatech Inc. on CodeSurfer development, and was a member of the Crest Centre at Kings' College London. His current research focuses on software product families and the application of information retrieval techniques in software engineering.

He recently co-chaired the program for 2014 Software Evolution Week combining WCRE and CSMR.

Shubhangi Chaturvedi received the B.Eng. degree from Rajiv Gandhi Proudyogiki Vishwavidyalaya, University, Madhya Pradesh, India, in 2013, and the M.Tech degree from National Institute of Technology Bhopal, Madhya Pradesh, India, in 2018. She is currently working toward the Ph.D. degree with the Indian Institute of Information Technology, Design and Manufacturing, Jabalpur, Madhya Pradesh, India.

She has been Assistant Professor of Computer Science and Engineering at National Institute of Technology Bhopal, Madhya Pradesh, India. Her research interest is in data mining and Big-Data analytics.