Generalizing Evolutionary Coupling with Stochastic Dependencies

Sunny Wong
Siemens Healthcare
Malvern, PA, USA
sunny.wong@siemens.com

Yuanfang Cai
Dept. of Computer Science
Drexel University
Philadelphia, PA, USA
yfcai@cs.drexel.edu

***Abstract*—Researchers have leveraged evolutionary coupling derived from revision history to conduct various software analyses, such as software change impact analysis (IA). The problem is that the validity of historical data depends on the recency of changes and varies with different evolution paths—thus, influencing the accuracy of analysis results. In this paper, we formalize evolutionary coupling as a stochastic process using a Markov chain model. By varying the parameters of this model, we define a family of *stochastic dependencies* that accounts for different types of evolution paths. Each member of this family weighs historical data differently according to their recency and frequency. To assess the utility of this model, we conduct IA on 78 releases of five open source systems, using 16 stochastic dependency types, and compare with the results of several existing approaches. The results show that our stochastic-based IA technique can provide more accurate results than these existing techniques.**

***Index Terms*—impact analysis, stochastic dependency, evolutionary coupling, Markov chain**

I. Introduction

Researchers have leveraged revision history to identify *evolutionary coupling* between components by checking how components historically change together [1]–[3]. The effectiveness of conducting various software analyses based on evolutionary coupling has been demonstrated. For example, Cataldo et al.'s empirical studies [4] suggest that evolutionary dependencies provide more accurate estimates of coordination requirements than structural dependencies. Software change impact analysis[1] (IA) [5] exemplifies another analysis technique where evolutionary dependencies often outperform structural dependencies. Different from traditional change impact analysis techniques that are often performed based on static dependency structure, evolutionary-coupling-based IA captures semantically coupled components that may not structurally depend on each other.

Although historical data has been shown to be a useful resource for various analyses, the validity of historical data depends on the recency of changes and varies with different evolution paths. For example, the amount of history needed to accurately estimate change impact would be different for a relatively new system that is frequently refactored than for a long established system with a stable architecture.

To improve the accuracy of history-based analysis techniques, we present a formalized generalization of evolutionary dependencies based on probability theory, which we call *stochastic dependencies*. This formalization defines a family of dependency types. To account for different evolution paths, each member of the family weighs historical data differently according to their recency and frequency. Our stochastic dependency family can be used to augment prevailing history-based analysis techniques by selecting a member of the family that fits the project in consideration.

Our stochastic dependency framework is based on the Markov chain model [6], a probabilistic model often used in artificial intelligence for determining the expected result of a random variable/event. Our assumption is that each transaction (i.e., atomic commit) in revision history can be modeled as a random variable. To address the temporal issue of changing dependencies that invalidate historical data, our approach first limits the length of historical data used for analysis. This limited sequence of transactions create a sliding window that resembles a Markov chain (discrete k-th order Markov process) in which each state of the chain is a transaction from revision history.

From the Markov chain, we can compute the probability that two components are coupled (i.e., they will both be changed in the next transaction). This probability value can be used to reason about whether a component will be in the impact scope of a change. To account for the importance of recent history over distant history, our model for computing this dependency probability uses a smoothing function to control how much each transaction contributes to the predicted probability. The two parameters to our framework (i.e., the length of history to use and the type of smoothing function) allow for the definition of a family of stochastic dependency types.

To evaluate our approach, we conduct change impact analysis on 78 releases of five open source systems by varying the values of the two stochastic dependency parameters. Our results show that the stochastic-based change impact analysis can provide more accurate results than two traditional structure-based IA approaches and an existing history-based IA approach for all five systems.

[1]Change impact analysis (also called change propagation analysis and change scope analysis) is the method of determining what software elements (components, files, etc.) are likely to change (called the change scope or the impact scope) when, or after, a certain set of elements (called the starting impact set) changes [5].

978-1-4577-1639-3/11/$26.00 ©2011 IEEE

The rest of this paper is organized as follows: Section II presents related work. Section III uses an example to overview the stochastic dependency computation for change impact analysis. Section IV formally defines the family of stochastic dependencies. Section V presents our evaluation method and empirical results. Section VI discusses threats to validity and future work. Section VII concludes.

II. Related Work

In this section, we review related work and differentiate our stochastic based technique from existing approaches.

A. Impact Analysis

Numerous IA techniques have been proposed. Below, we discuss several representative categories of IA approaches.

1) Structure-based Impact Analysis: Various IA techniques have been developed using software dependency structures. For example, Briand et al. [7] and Elish and Rine [8] proposed IA techniques based on the metrics of Chidamber and Kemerer [9]. Robillard [10] and Rajlich [11] presented algorithms based on dependency graph structures. To capture impact that cannot be easily identified through syntactic analysis, some IA approaches (e.g. Apiwattanapong et al. [12]) use dynamic (i.e., instance-based rather than class-based) analysis to measure coupling.

2) History-based Impact Analysis: Ying et al. [3] and Zimmermann et al. [2] leverage revision histories to perform IA based on association rules that identify evolutionary (or logical) coupling [1] between software elements. An association rule, of the form $a \Rightarrow b$, states that if a is changed then b will likely also be changed. These rules are prioritized based on the heuristics of *support* and *confidence*. *Support* is defined as the number of transactions that a and b occur together in revision history. *Confidence* is defined as *support* divided by the number of times that a occurs (with or without b). With minimum support and confidence thresholds, association rules are selected to predict the impact scope of a change starting from a.

Evolutionary dependency approaches often consider the transactions in the overall revision history to be a multiset and disregard the temporal sequence (i.e., ordering) of the transactions. Two approaches that include temporal information in IA include the work of Bouktif et al. [13] and Ceccarelli et al. [14]. Different from our approach of using Markov chains, their purpose is to infer cause-effect relationships from the temporal data, rather than to filter out obsolete history data.

The Evolution Radar [15] is another approach that considers the temporal sequence of transactions. It allows for the tracking of evolutionary dependencies over time by dividing revision history into time intervals. However, its goal is different from our approach in that it is meant for the interactive visualization of dependency changes to aid the detection architecture decay.

While these existing history-based techniques are effective at identifying semantic dependencies between software elements, refactorings that remove these dependencies may cause the approaches to overestimate the impact scope. Since the support heuristic never decreases, once support passes the minimum threshold, a dependency between two elements continues to exist. Although the confidence heuristic can be used in conjunction with support, as the number of transactions becomes large, confidence only minimally changes with each transaction. Our stochastic dependency framework addresses this issue by limiting the length of historical information analyzed and uses a smoothing function to emphasize more recent history over distant history.

Cataldo [16] explored the question of how many months of revision history are enough to accurately compute evolutionary dependencies and coordination requirements. He constructed task dependency matrices on monthly increments to find when a matrix no longer significantly differs from a previous matrix. Although he found a fix point (19 months) when the dependencies stabilized, the result is hard to generalize to other software systems. Recent architectural refactorings can significantly alter the structure of dependencies, invalidating previous revision history.

3) Probabilistic Impact Analysis: Recently, several impact analysis techniques have been proposed that are based on probability theory. For example, Tsantalis et al. [17] define probabilities of impact based on structural relations between software elements in object-oriented designs. Abdi et al. [18] use a Bayesian network (a probabilistic model) with structural coupling metrics to construct the inference model. Mirarab et al. [19] also use a Bayesian network and combine structural coupling with historical data in constructing the network. While our approach also uses a probabilistic model (Markov chain), stochastic dependencies use the temporal sequence of transactions in revision history to account for changes to the dependency structure over time.

B. Markov Processes

Markov processes (of which Markov chains are a specific type) are probabilistic models that are widely used in artificial intelligence (e.g. reinforcement learning [20]) and various other computing fields (e.g. web search engine algorithm [6]). Markov processes have also been applied to software engineering (e.g. generate test inputs [21], classify software behavior [22], predict component reliability [23]). To the best of our knowledge, Markov processes have not yet been applied to computing evolutionary coupling.

III. Framework Overview

In this section, we present a high-level overview of our stochastic dependency framework, using concrete (but hypothetical) examples to demonstrate how to compute stochastic dependencies and how to conduct IA using them. The next section provides more rigorous, formal definitions and explanations for the concepts discussed here. For all the examples in this section, we consider that we have the following sequence of transactions in our revision history: $\{a, b\}$, $\{a, c\}$, $\{d\}$, $\{a, b\}$, $\{a\}$, $\{a, b, c\}$, $\{b, d\}$, $\{a\}$, $\{a, d\}$, $\{c\}$, $\{a, c\}$, $\{a\}$; where $\{a\}$ is the most recent transaction.

Intuitively, a stochastic dependency (β, α) is the *probability* that, if the next transaction includes α, it will include β. To compute this probability, we look at the previous transactions that include α. Each one of these transactions that also includes β gives us evidence that a dependency exists (i.e., β depends on α). To account for different evolution paths and changing design structures, we use two parameters in computing the stochastic dependency value: a limit on history length and a smoothing function. The history length parameter k defines the number of transactions to look at; thereby, ignoring transactions that occurred in the distant past. The smoothing function λ allows us to give more weight to transactions that occurred recently over those from the distant past.

We first consider an example of limiting to the last five transactions ($k = 5$) and using a linear λ smoothing function (i.e., the usefulness of historical data decays linearly with time, as shown in Table I). To compute the stochastic dependencies on c, we first find the last five transactions that involve c: $\{a, c\}$, $\{a, b, c\}$, $\{c\}$, $\{a, c\}$. Since c has only been involved in four transactions so far, we only use the four available transactions. Then we compute the stochastic dependencies as shown below:

	$\{a,c\}$	$\{c\}$	$\{a,b,c\}$	$\{a,c\}$	Pr
a	$\lambda_1 \cdot 1$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 1$	$\lambda_4 \cdot 1$	0.67
b	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 1$	$\lambda_4 \cdot 0$	0.20
d	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 0$	$\lambda_4 \cdot 0$	0

From these stochastic dependency values, we can perform IA and estimate the impact of changing c. There are various methods to use these values for IA. The simplest way is to use a threshold-base rounding scheme. If we use 0.5 as the minimum threshold, then we would predict that only a is likely to be in the impact scope of c.

As another example, we consider analyzing the impact of changing a. The last five transactions that involve a are $\{a, b, c\}$, $\{a\}$, $\{a, d\}$, $\{a, c\}$, $\{a\}$. Again we use the same linear smoothing function:

	$\{a\}$	$\{a,c\}$	$\{a,d\}$	$\{a\}$	$\{a,b,c\}$	Pr
b	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 0$	$\lambda_4 \cdot 0$	$\lambda_5 \cdot 1$	0.07
c	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 1$	$\lambda_3 \cdot 0$	$\lambda_4 \cdot 0$	$\lambda_5 \cdot 1$	0.33
d	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 1$	$\lambda_4 \cdot 0$	$\lambda_5 \cdot 0$	0.27

Using the same minimum threshold of 0.5, we would expect that changing a will not impact any other elements.

IV. Formalization

In this section, we first present the definitions and mathematical notations to formally define the *stochastic dependency*. After that, we describe a method for computing the dependency value.

A. Definitions and Background

Let $E = \{e_i\}_{i=1}^{N}$ be the set of software elements in the system, where N is the total number of elements.[2] Let $T = \{t_i\}_{i=1}^{M}$ be the sequence of revision history transactions, each involving a subset of software elements (i.e., $\forall t_i \in T : t_i \subseteq E$), where M is the length of T. We can view T as a stochastic process, where each t_i is a discrete random variable with domain 2^E. For any element $e \in E$, let $T^{(e)} = \left\{ t_i^{(e)} \in T \mid e \in t_i^{(e)} \right\}$ be the subsequence of T involving e. Without loss of generality, we use separate indexing sequences for T and $T^{(e)}$—in other words, $t_1^{(e)}$ is not necessarily the first element of T, $t_2^{(e)}$ is not necessary the second element of T, etc. Then let $M^{(e)}$ be the length of $T^{(e)}$.

Given two elements $a, b \in E$, let $C^{(a,b)} = \left\{ X_i^{(a,b)} \right\}_{i=1}^{M^{(a)}}$ be a stochastic process that models whether b is in the transactions that involve a:

$$X_i^{(a,b)} = \begin{cases} 1 & \text{if } b \in t_i^{(a)} \\ 0 & \text{otherwise} \end{cases}$$

We define the stochastic dependency (b, a) at time τ as $\Pr\left(X_\tau^{(a,b)} = 1\right)$. The method for computing this probability varies among the different types of stochastic dependencies. Unlike some existing dependency definitions, in which a dependency either exists or does not, we define that an element is stochastically dependent upon another with a continuous probability. Given a starting impact set a, we compute its stochastic dependents from all other elements $b \in E \backslash \{a\}$ to determine if b will be in the impact scope of a. In this paper, we use the terms change scope and impact scope interchangeably.

With the probability values of stochastic dependencies, we can reason about which elements are expected to be in an impact scope. While we can simply sort the software elements by decreasing probability values and present the list to a maintainer, this continuous range also allows for various techniques to reduce the number of elements presented to a maintainer. For example, we can define a minimum probability threshold and consider all elements above that threshold to be in the impact scope. Alternatively, we can use randomized rounding [24] to select the likely impacted elements. For simplicity, we use a minimum threshold strategy for our evaluation in Section V. Next, we formally define our method of computing stochastic dependencies.

B. Computing Stochastic Dependencies

Given a sequence of $\tau - 1$ transactions in revision history involving an element a, we have defined the probability that another element b will be involved in the next transaction involving a as the *stochastic dependency* from b to a at time τ. For each of the $\tau - 1$ transactions involving a, let x_i be the value of $X_{\tau-i}^{(a,b)}$ (i.e., x_i indicates whether b and a occurred together in the $(\tau - i)$-th transaction involving a). Then we

[2] Although software elements may be added and deleted over time, we use E to refer to the set of elements in the software at the time of interest.

define the probability that b is stochastically depend on a when the τ-th transaction involving a occurs as follows:

$$\Pr\left(X_{\tau}^{(a,b)}=1\right) \equiv$$
$$\Pr\left(X_{\tau}^{(a,b)}=1 \mid X_{\tau-1}^{(a,b)}=x_1 \wedge \cdots \wedge X_{1}^{(a,b)}=x_{\tau-1}\right)$$

As a first step in accounting for different evolution paths of software, we consider only the latest k transactions involving a for analysis. We emphasize that these are the last k transactions that software element a is involved in, but not necessarily the latest k transactions in the revision history. Selecting an appropriate value for k can affect the accuracy of impact analysis. If k is too large then dependencies may have been removed during evolution. On the other hand, if k is two small then semantic dependencies between components may not be detected. We investigate the effects of various k values in our evaluation reported in Section V.

Only considering the latest k transactions creates a sliding window that resembles a discrete k-th order Markov process (Markov chain), which leads to our method of computing the stochastic dependency probability. A Markov chain is a stochastic process with a property that the next state depends only on the current state and a finite number of previous states, but not the entire history of states. A k-th order Markov chain depends on the current state and the $k-1$ previous states—or formally, $\Pr(Y_i \mid Y_{i-1}, \ldots, Y_1) \equiv \Pr(Y_i \mid Y_{i-1}, \ldots, Y_{i-k})$.

A prevailing model [6] for high order Markov chains defines the probability for a next state to be based on a linear combination from the previous states. Based on this prevailing model, we derive a method to compute *stochastic dependencies* that is also based on a linear combination of previous states:

$$\begin{aligned} \Pr\left(X_{\tau}^{(a,b)}=1\right) \quad &\equiv \quad \sum_{i=1}^{k} \lambda_i X_{\tau-i}^{(a,b)} \\ \text{where} \quad & \quad \lambda_i \in [0,1] \\ \text{and} \quad & \quad \sum_{i=1}^{k} \lambda_i = 1 \end{aligned}$$

Based on this model, every time b changes with a in a transaction, we gain evidence that b depends on a and this evidence contributes to the computed probability. Intuitively, if b often changes with a recently, then it is more likely that it will change with a in the near future. On the contrary, if b only changes with a in the distant past, but not recently, it is more likely that this dependency has been removed during evolution. To capture this temporal phenomenon, we use a monotonically decreasing smoothing function,[3] λ, to account for software evolution and weigh more recent transactions more heavily than older transactions. Table I shows several functions that can be used as the smoothing function λ.

[3]Technically, $\lambda = \{\lambda_i\}_{i=1}^{k}$ is a sequence of k values but it can be intuitively understood as a function that maps to the appropriate sequence value. Hence, we interchangeably refer to it as either a sequence or a function.

The first column shows a constant function in which all the transactions are weighed the same regardless of how long ago they occurs, as with the traditional evolutionary dependency definition. The second column shows a linear function, indicating that the usefulness of a transaction in terms of determining stochastic dependency decreases linearly over time. The third column shows a sinusoidal function that weighs the most recent transactions similarly, and older transactions less and less. The last column shows an exponentially decaying function, which models a rapid decreasing of the usefulness of a transaction over time. These last three functions weigh past transactions less heavily than recent transactions.

By using a limited history k and a decreasing λ sequence, our stochastic dependencies potentially can recover more quickly from refactorings, which may significantly alter the dependency structure of a design, than traditional evolutionary dependencies that only use confidence to detect such changes. Our stochastic dependency definition is a generalization of traditional evolutionary dependencies because they represent a specific k value and λ sequence for stochastic dependencies. For example, to use a minimum support value of σ and minimum confidence of χ in determining traditionally-defined evolutionary dependency, we can compute the evolutionary dependency (b, a) at the time when τ-th transaction involving a occurs using our stochastic dependency model by letting $k = \tau$ (considering all the existing transactions involving a) and $\lambda = \{1/k\}_{i=1}^{k}$ (treating these transactions equally). Then a evolutionary dependency (b, a) is said to exist at time τ if the probability exceeds both the minimum support and confidence:

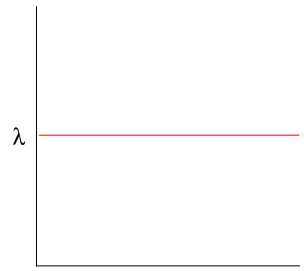
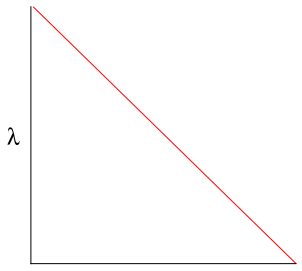
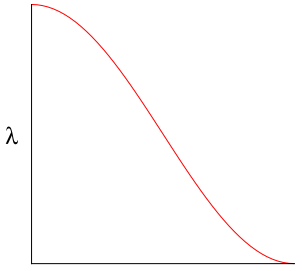
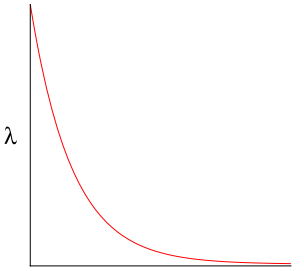
$$\Pr\left(X_{\tau}^{(a,b)}=1\right) \geq \max\left\{\chi, \frac{\sigma}{\tau}\right\}$$

V. Evaluation

To evaluate the effectiveness of our stochastic model, we conduct change impact analysis using the stochastic dependencies defined in the previous section. We compare the accuracy of IA using 16 members of the stochastic dependency family with different combinations of k (length of history) and λ (smoothing function). We also compare the results with two traditional structure-based IA techniques and the prevailing evolutionary-coupling-based IA techniques, aiming to answer the following questions:

- Q1: Are stochastic-based IA approaches more accurate than traditional structure-based IA techniques? For each subject system, we compare the accuracy of each stochastic-dependency-based IA with structure-based IA techniques.
- Q2: Are stochastic-based IA approaches more accurate than prevailing history-based IA techniques, which do not take different types of evolution into consideration? As we mentioned before, the prevailing evolutionary coupling definition is a special case of our stochastic dependency family.
- Q3: How does the selection of k affect the accuracy of impact analysis? We investigate the hypothesis that the IA will achieve peak performance with a particular choice of k and that the length history can impact IA performance.

TABLE I
EXAMPLE LAMBDA SEQUENCES

Constant	Linear	Sinusoidal	Exponential
			
$\frac{1}{k}$	$\frac{2(k-i+1)}{k^2+k}$	$\frac{\cos\left(\pi k^{-1}(i-1)\right)+1}{k+1}$	$2^{-i}+\frac{2}{k2^{k+1}}$

Q4: How does the selection of a λ sequence affect the accuracy of impact analysis? We investigate the hypothesis that, in general, the usefulness of a transaction decreases over time. We consider that the hypothesis is true if we observe increasing IA performance with a decreasing λ. We also hypothesize that the best λ function for each subject system will be different because, as independent projects, their evolution paths should be different.

In this section, we first describe the five software systems to which we apply our approach. Then we describe the evaluation procedure and present the results.

A. Subjects

Table II shows some basic information of the following five open source systems that we chose as the subjects of our evaluation. These projects were chosen because they have different sizes and are from different domains.

Log4J:[4] a popular logging framework for the Java programming language.

Hadoop Common:[5] a Java-based distributed computing framework; Hadoop Common provides the shared components and functionality used by other Hadoop sub-projects.

JEdit:[6] a text editor that supports syntax highlighting of source code and the ability to write macros.

Apache Ant:[7] an automated software build tool for Java applications.

JBoss:[8] one of the most widely used Java application servers in the world. It provides a platform for running enterprise Java applications based on the Java EE standards.

B. Evaluation Procedure

For each subject system, we extract transaction information from its Subversion (SVN) repository. For each transaction, we randomly select a file that is involved in the transaction as the starting impact set, and performed impact analysis on it. Assuming each transaction is a single change task, an ideal IA approach would be able to identify all files in the transaction as being in the impact scope. We perform IA on each transaction up to five times (fewer if the transaction has fewer than five files), with a different random starting impact file each time. The same starting impact files for each transaction are used for all the approaches for unbiased comparison. Consistent with the work of Zimmermann et al. [2] we ignore transactions with more than 30 files as they are unlikely to be meaningful. These large blanket operations on the code base are often changes to licensing information that need to be updated in the comment section of all files; or other trivial housekeeping activities on the code base.

We use the standard information retrieval measures of *precision*, *recall*, and F_1 for assessing the accuracy of IA. Precision measures how much of the predicted impact scope is correct, while recall measures how much of the actual impact scope was predicted. The F_1 measure/score combines precision and recall into a single number for ease of comparison. Next we introduce the existing IA techniques against which we compare our stochastic-based IA technique.

$$precision = \frac{\#\ correct}{\#\ predicted}$$

$$recall = \frac{\#\ correct}{transaction\ size}$$

$$F_1 = \frac{2 \times precision \times recall}{precision + recall}$$

TABLE II
SUBJECT SYSTEM INFORMATION

Subject	# Vers	History	KSLOC	# Trans
Log4J	11	12/00–6/07	15	3550
Hadoop Common	20	2/06–12/09	36	9319
JEdit	12	9/01–2/10	98	17339
Apache Ant	20	1/00–8/10	125	14554
JBoss	15	10/99–8/10	534	107501

[4]http://logging.apache.org/log4j/
[5]http://hadoop.apache.org/common/
[6]http://www.jedit.org/
[7]http://ant.apache.org/
[8]http://www.jboss.org/

1) Structure-based Impact Analysis: To answer the first evaluation question, we compare our stochastic IA accuracy with that of two traditional structure-based IA techniques. The first is from the work of Briand et al. [25]. They proposed a structural coupling measure that combines whether classes from different inheritance hierarchies interact (CBO'), whether a class (directly or indirectly) aggregates another class (INAG), and the number of method calls between classes (PIM). To ease analysis, we normalize their coupling measure into the range of $[0, 1]$ by dividing by the largest measure value. In this section, we refer to this technique as the *static* dependency approach. The other structure-based approach we compare against is the work of Robillard [10]. Robillard defines an algorithm that, given a starting impact set, assigns a weight in the range $[0, 1]$ to other software elements based on analyzing the structure/topology of the dependency graph. In this section, we refer to this technique as the *topology* dependency approach.

To perform IA using these measures, we first compute the dependency values of the files based on the software structure in the most recent release. Given a minimum threshold value, we select all files whose dependency value is at least the threshold value to be included in the impact scope. Prior to performing IA on a given software release, we first identify the minimum threshold value that maximizes the F_1 measure. Essentially, we compare against the best accuracy of these structure-based IA.

2) History-based Impact Analysis: To answer the second evaluation question, we compare the IA accuracy using stochastic dependency against traditional evolutionary dependency [2], [3]. We perform traditional history-based IA on a transaction by analyzing all transactions from the beginning of the revision history to the most recent release (in order to be fair to the structural dependency experiments). Similarly to the structural dependency experiments, we determine the best minimum support and confidence values that maximize F_1 prior to processing the transactions for each software release.

3) Stochastic-based Impact Analysis: We use the four λ sequences presented in Table I as the smoothing function of each stochastic dependency family member. For each λ sequence, we use the values of 5, 10, 20, and 40 for k (maximum number of transactions to consider). As a result, we consider 16 members of the stochastic dependency family in total. Like with evolutionary dependencies, we consider the transaction for the most recent software release to be the latest transaction available for analysis. Although various rounding techniques are possible for our stochastic dependency value to determine whether a file is in the impact scope, we follow the same strategy as with the structural dependencies—given a minimum threshold, a file is considered in the impact scope if its stochastic dependency value is at least as large as the threshold. Also similar to the other two types of IA approaches, we find the best minimum threshold for each software release.

C. Results

Table III shows the results from our evaluation. For each subject system and IA technique, we show the average F_1 score in the column labeled $\bar{F}_1$. The average F_1 score is computed over the IA predictions for all transactions. For example, with Apache Ant, we ran a total of 4306 impact analyses on 1313 transactions with each dependency type. We compute the average F_1 score for each dependency type, by adding the F_1 score for each of the 4306 IA analysis results and dividing the total by 4306. Due to space constraints, we only show the F_1 values for the stochastic dependencies where $k = 10$. F_1 values for other k values are discussed below.

a) Q1. Comparing with Structure-based IA: Table III shows that the accuracy, by F_1 measure, of both structure-based IA for all the subjects are lower than any member of the stochastic-based IA approaches, and are also lower than traditional history-based impact analysis. Take Hadoop Common for example, the topology approach achieved an average F_1 score of 0.5163 and the static approach achieved an average F_1 score of 0.5174. In contrast, all the stochastic dependencies had average F_1 scores of above 0.55.

To confirm our intuition based on the average F_1 measures, we apply the Wilcoxon signed rank test [26] to compare each of the stochastic dependency types against static dependency types. The null hypothesis H_0 for each statistical test is that the F_1 value (per impact analysis on each transaction) achieved by the existing approach is the same as the F_1 value achieved by the stochastic dependency. The alternative hypothesis H_1 is that the stochastic dependency achieved greater F_1 value than the existing approach. The resulting W-scores and p-values from performing the statistical tests, with a 95% confidence interval, are also shown in Table III.

The results of the Wilcoxon signed rank test corroborate our intuition that the stochastic dependencies outperformed the structure-based IA approaches. All the p-values from the statistic tests were $< 2.2 \times 10^{-16}$ (the smallest p-value possible in our statistics software: R[9]). These p-values, which are statistically quite significant, indicate that the null hypothesis should be rejected and the alternative hypothesis should be accepted (i.e., the stochastic dependencies provide more accurate predictions than the static dependencies). From these results we can affirmatively answer the first evaluation question that stochastic dependencies are more accurate than traditional structural dependencies at IA.

b) Q2. Comparing with Traditional History-based IA: Looking at the results from Table III, we see that for any of the five subject system, three out of the four types of the stochastic dependencies yield a higher average F_1 score than traditional evolutionary dependencies, and that the differences are statistically significant. Take Log4J as an example: we observe that the average F_1 for the constant stochastic dependency is 0.3492, which is lower than that of evolutionary dependencies, 0.3641. The statistical test results (W-score of 38457.5 and p-value of 0.9989) confirm that the stochastic dependency does

[9]http://www.r-project.org/

TABLE III
WILCOXON SIGNED RANK TEST RESULTS FOR F_1 SCORES ($k = 10$)

Subject	Stochastic		Evolutionary			Topology			Static		
	λ	$\bar{F}_1$	$\bar{F}_1$	W	p	$\bar{F}_1$	W	p	$\bar{F}_1$	W	p
Log4J	Const	0.3492	0.3641	38457.5	0.9989	0.3082	66804.5	$<$ 2.2e-16	0.2992	77051	$<$ 2.2e-16
	Line	0.3913		62979.5	7.133e-6		95164.5	$<$ 2.2e-16		131060.5	$<$ 2.2e-16
	Sine	0.3974		82792.5	1.57e-7		119081	$<$ 2.2e-16		158798.5	$<$ 2.2e-16
	Expon	0.3935		94266.5	1.552e-4		90482	$<$ 2.2e-16		116364	$<$ 2.2e-16
Hadoop	Const	0.5534	0.5658	795625.5	1.0	0.5163	809766	$<$ 2.2e-16	0.5174	880037.5	$<$ 2.2e-16
	Line	0.5761		1094091	2.654e-5		1488936	$<$ 2.2e-16		1584656	$<$ 2.2e-16
	Sine	0.5780		1131795	8.406e-7		1540090	$<$ 2.2e-16		1648998	$<$ 2.2e-16
	Expon	0.6034		1988273	$<$ 2.2e-16		2075044	$<$ 2.2e-16		2217333	$<$ 2.2e-16
JEdit	Const	0.3950	0.3796	14149902	5.043e-6	0.3525	4962412	$<$ 2.2e-16	0.3525	4962412	$<$ 2.2e-16
	Line	0.3993		14404088	1.054e-6		3555722	$<$ 2.2e-16		3555722	$<$ 2.2e-16
	Sine	0.3951		14207902	0.001198		2117458	$<$ 2.2e-16		2117458	$<$ 2.2e-16
	Expon	0.3943		14337290	0.2036		1445818	$<$ 2.2e-16		1445818	$<$ 2.2e-16
Ant	Const	0.4203	0.4265	295247	0.9938	0.3733	625191.5	$<$ 2.2e-16	0.3701	606184.5	$<$ 2.2e-16
	Line	0.4690		1209382	$<$ 2.2e-16		1988838	$<$ 2.2e-16		1988610	$<$ 2.2e-16
	Sine	0.4691		1194846	$<$ 2.2e-16		1914936	$<$ 2.2e-16		1908747	$<$ 2.2e-16
	Expon	0.4662		1071560	$<$ 2.2e-16		1216914	$<$ 2.2e-16		1179197	$<$ 2.2e-16
JBoss	Const	0.6335	0.6500	1030575	1.0	0.6145	928520.5	$<$ 2.2e-16	0.6145	928520.5	$<$ 2.2e-16
	Line	0.6574		2183468	4.339e-4		3208790	$<$ 2.2e-16		3208790	$<$ 2.2e-16
	Sine	0.6576		2167306	2.895e-4		3045315	$<$ 2.2e-16		3045315	$<$ 2.2e-16
	Expon	0.6751		3027636	$<$ 2.2e-16		2444432	$<$ 2.2e-16		2444432	$<$ 2.2e-16

not outperform the evolutionary dependencies in this scenario. We highlight the cells of Table III where the statistical test results indicate that our stochastic dependencies did not outperform the existing dependency type.

Although the stochastic dependencies do not always outperform the evolutionary dependencies, we affirmatively answer our second evaluation question because, for each subject system, most stochastic dependency types can provide more accurate IA results than traditional evolutionary dependencies. Due to the difference in the evolution paths of subject systems, it is not surprising that some stochastic dependency types can provide more accurate IA results. Below, we further explore how the parameters that define stochastic dependencies (k and λ) influence analysis accuracy.

c) Q3. Effects of k: In determining the effects of k (length of history) on the accuracy of analysis, we again apply the Wilcoxon signed rank test. Given a fixed λ smoothing function, we vary the value of k and compare the F_1 values. Table IV shows the average F_1 scores for Hadoop Common as a representative example.

TABLE IV
AVERAGE F_1 SCORES FOR $\langle\lambda, k\rangle$ PAIRS: HADOOP COMMON

λ \ k	5	10	20	40
Const	0.5679	0.5534	0.5415	0.5289
Line	0.5919	0.5761	0.5597	0.5429
Sine	0.5922	0.5780	0.5608	0.5443
Expon	0.6005	0.6034	0.6032	0.60302

From Table IV we see that increasing the value of k generally decreases the accuracy of analysis (except in the case of the exponential smoothing function where increasing k from 5 to 10 improves the accuracy). To corroborate these observations, we apply the Wilcoxon signed rank test with a null hypothesis that two different k values are equally accurate. The alternative hypothesis is that one of the k values is more accurate than the other. For example, comparing $k = 5$ and $k = 10$ for the linear smoothing function of Hadoop, we obtain a W-score of 863620.5 and p-value of 1.749×10^{-9}, which confirms that $k = 5$ is statistically more accurate than $k = 10$. Comparing $k = 10$ and $k = 20$ for the same smoothing function yields a W-score of 687905 and a p-value of 9.938×10^{-14}, which indicates that $k = 10$ is more accurate than $k = 20$.

We applied this statistical test to all pairs of k values in each subject system and found consistently that changing the k value does affect the accuracy of analysis. Due to space restrictions, we do not elaborate on the statistical test results for the other subject systems. These results allow us to affirmatively answer our third evaluation question—that the value of k does influence the accuracy of analysis, and that both too long and too short of a history will negatively impact the IA results.

d) Q4. Effects of λ: To evaluate the effects of different λ sequences, we compare the accuracy of applying the four λ sequences for each given subject system. For example, Table V shows the average F_1 values for JBoss under different combination of λ and k. For example, when $k = 10$, the constant smoothing function yields lower accuracy than the

other functions, and the exponential smoothing function yields highest accuracy. To conduct the Wilcoxon signed rank test, we define null and alternative hypotheses to compare different λ pairs under the same k.

As an example, we define a null hypothesis as the linear and constant smoothing functions are equally accurate and the alternative hypothesis as the linear function is more accurate. For JBoss with $k = 10$, we obtain a W-score of 2062498 and a p-value of $< 2.2 \times 10^{-16}$. This statistically significant result confirms that the linear λ performs better than constant λ function. Comparing the exponential and sinusoidal smoothing functions, we obtain a W-score of 2159677 and a p-value of 1.27×10^{-12}, confirming that the exponential function is more accurate than the sinusoidal function. On the other hand, comparing the linear and sinusoidal smoothing functions yield a W-score of 74374.5 and a p-value of 0.8096, indicating no significant difference in accuracy. These results suggest that JBoss follows an evolution path with high refactoring activity, since the historical data quickly becomes inaccurate in performing IA.

TABLE V
AVERAGE F_1 SCORES FOR $\langle \lambda, k \rangle$ PAIRS: JBOSS

λ \ k	5	10	20	40
Const	0.6508	0.6335	0.6154	0.6107
Line	0.6694	0.6574	0.6367	0.6193
Sine	0.6696	0.6576	0.6366	0.6195
Expon	0.6754	0.6751	0.6752	0.6752

We obtain similar results with other subjects: different λ provides different IA accuracy in different projects: the exponential smoothing function is best for Hadoop and JBoss; with Log4J, sinusoidal smoothing slightly outperforms others; for Apache Ant, the linear and sinusoidal functions are equally best; for JEdit, the linear and constant functions are equally best. So we can positively answer the last evaluation question: in most cases, a decreasing λ sequence increases accuracy and the best λ for a project differs due to different evolution paths where the speeds at which historical data becomes irrelevant can vary.

D. Assessing Precision and Recall

While the F_1 score provides a simple, single quantity measure, the respective importance of precision and recall measures varies with applications. For the purpose of impact analysis, the recall measure is considered more important than precision because developers do not want to miss important files that need to be changed. In this subsection, we provide further elaboration on the precision and recall values, using JBoss, the largest subject systems as a representative example. In Table VI, we present the quartile values of precision and recall scores of each type of IA analysis on JBoss.

Table VI shows that the static and topology dependencies have very high precision but low recall compared to stochastic dependencies. The high precision is not surprising because these techniques identify components that are highly structurally coupled and hence they are most likely to be impacted by changes. The low recall is also not surprising: many semantically coupled components do not have structural dependencies. By leveraging historical data, the evolutionary and stochastic dependencies can detect semantic coupling and improve the recall scores, but at the cost of slightly decreased precision values. The large increase in recall compensates for the small lost in precision, and improves the overall F_1 accuracy.

Similarly, Table VI shows that the exponential smoothing function achieves the same precision quartile values as the evolutionary dependencies, even though the average (arithmetic mean) precision is slightly lower (not shown). The table also shows that comparing with traditional evolutionary dependencies, exponential stochastic dependency achieves significant improvement in the recall: in the first quartile (Q1), the recall increases from 0.25 to 0.5 and the median (Q2) recall increase from 0.6 to 1. This means that more than half the impact analyses conducted by the exponential stochastic dependencies had perfect recall.

TABLE VI
QUARTILE VALUES OF IA ACCURACY ON JBOSS ($k = 10$)

Approach	Measure	Min	Q1	Q2	Q3	Max
Static	Precision	0	1	1	1	1
	Recall	0	0.2	0.5	1	1
Topology	Precision	0	1	1	1	1
	Recall	0	0.2	0.5	1	1
Evolutionary	Precision	0	1	1	1	1
	Recall	0	0.25	0.6	1	1
Const Stochastic	Precision	0	1	1	1	1
	Recall	0	0.25	0.5	1	1
Line Stochastic	Precision	0	0.5	1	1	1
	Recall	0	0.33	1	1	1
Sine Stochastic	Precision	0	0.6	1	1	1
	Recall	0	0.33	1	1	1
Expon Stochastic	Precision	0	1	1	1	1
	Recall	0	0.5	1	1	1

We make similar observations from the other subject systems: comparing with traditional evolutionary dependencies, our stochastic dependencies consistently have slightly lower precision values, but have significantly better recall values, and thus significantly higher F_1 values. Table III shows that the F_1 values for the other subject systems are lower than those for JBoss, but this precision/recall tradeoff occurs with all the subjects.

VI. DISCUSSION

In this section, we discuss the evaluation results, threats to validity, and possible future work.

Although stochastic dependency is a generalization of evolutionary dependency, not all stochastic dependencies with constant λ sequences are evolutionary dependencies due to the k parameter. In order to identify the same impact scope as evolutionary dependencies, we need to have a constant

λ sequence *and* k must be the number of *all* the transactions involving the starting impact set. That is why, in our evaluation, the stochastic dependency types with a constant λ sequence did not perform the same as traditional history-based IA. To achieve the same results as existing history-based IA, we would have needed to vary the value of k for each transaction.

The low IA accuracy for some of the evaluation subject systems may seem alarming at first (the average F_1 measure dips down to $1/3$ for some systems). However, the accuracy in actual usage may be significantly higher. This low accuracy in the evaluation is due to two factors: the directional nature of impact analysis and the selection of a random starting impact set. For example, consider a transaction that contains a base class `A` and its two subclasses `B` and `C`, and assume that it is only possible for changes to `A` to affect `B` and `C`, but changes to `B` or `C` never affect `A`. Since we do not know that `A` is the correct starting impact set, our evaluation approach considers each file to be a starting impact set. Hence, even a perfect IA approach would only achieve a 0.33 average F_1 score ($F_1 = 1$ with `A` as starting impact set, $F_1 = 0$ with `B` or `C` as starting impact set). Identifying the actual starting impact set of a transaction or modification request is an active area of research (e.g. Antoniol et al. [27]) that is orthogonal to our stochastic approach and can be used to improve IA accuracy in the future.

In our evaluation, we found the optimal minimum threshold values to compare each IA approach at its best accuracy. However, in practice, the best value is not known a priori. One strategy to address this problem is to use the optimal threshold from the previous release's transactions. After each software release, the minimum threshold can be recomputed and used for estimating during the next release.

e) Threats to Validity: Since we only applied our approach to five Java-based, object-oriented systems, we cannot conclude that the effectiveness of stochastic dependencies generalizes to all software systems; however, we did choose projects of various sizes and domains to begin addressing this issue. Similarly, we only applied the stochastic model to IA, but not other analysis techniques where historical data can be leveraged, such as bug triage prediction. Thus we cannot claim that the same improvement can be generalized to other history-based analysis techniques.

As with any technique that derives dependencies from transactions in revision history, our approach assumes that the transactions represent cohesive work units. In other words, if a developer only commits to the revision control system a batch of files once in a while, the files committed together may have no semantic relationship and are only coincidentally occurring in the same transaction. The accuracy of IA produced by stochastic dependencies and evolutionary dependencies indicate that transactions often are cohesive work units and generally do indicate semantic coupling of software elements.

Our framework assumes that the transactions of a revision control history form a stochastic process where the dependencies between software elements control the probability distribution of the random variables in this process. We use this assumption as the basis for building the Markov chain model. In reality, the probability distribution of these random variables (software elements that occur together in a transaction) can also be influenced by external factors and our assumption may not always be true. However, our evaluation shows that modeling the revision history as a stochastic process is a sufficient approximation for the actual behavior of development to yield more accurate IA results.

f) Future Work: The parameters of stochastic dependency allow for a large number of specific dependency types to be defined. In the study reported in this paper, we only used several k and λ values. Identifying additional, successful λ sequences is an ongoing work. For example, a possible λ sequence could be based on the how long ago (i.e., in terms of days, months, etc.) a transaction was committed. The best k value for different systems may also vary. Exploring techniques to automatically construct optimal λ and k values for a given software system is also a possible future work.

Exploring methods to improve the accuracy of stochastic-based software analysis in general is an ongoing work. In particular, we are exploring the use of more complex probabilistic models (e.g. hidden Markov models, dynamic Bayesian networks [20]) in place of the Markov chain model. Incorporating structural dependency information into these models to improve IA accuracy is also of interest.

VII. Conclusion

Using history-based evolutionary coupling to conduct various software analyses in software maintenance has gained popularity recently. However, differences in the speed of evolution of different software systems affect the validity of historical data used for analysis, and thereby the accuracy of the analysis results. Our stochastic dependency framework generalizes evolutionary coupling, using a length of history k and a smoothing function λ parameter to account for different types of evolution paths. By varying these two parameters, one can select the combination that best fits the system. Our experiment shows that performing IA using stochastic dependencies consistently outperforms structure-based IA techniques. Comparing with prevailing history-based IA techniques, our stochastic-based approach can provide better IA accuracy—in particular, significant improvement in recall for all five subject systems.

Acknowledgments

This work was supported in part by the National Science Foundation under grants CCF-0916891 and DUE-0837665.

References

[1] H. Gall, K. Hajek, and M. Jazayeri, "Detection of logical coupling based on product release history," in *Proc. 14th IEEE International Conference on Software Maintenance*, Nov. 1998, pp. 190–197.

[2] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller, "Mining version histories to guide software changes," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 429–445, Jun. 2005.

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

[4] M. Cataldo, A. Mockus, J. A. Roberts, and J. D. Herbsleb, "Software dependencies, work dependencies, and their impact on failures," *IEEE Transactions on Software Engineering*, vol. 35, no. 6, pp. 864–878, Jul. 2009.

[5] S. A. Bohner and R. S. Arnold, *Software Change Impact Analysis*. IEEE Computer Society, 1996.

[6] W.-K. Ching and M. K. Ng, *Markov Chains: Models, Algorithms, and Applications*, 2nd ed. Springer, 1996.

[7] L. C. Briand, Y. Labiche, L. O'Sullivan, and M. M. Sówka, "Automated impact analysis of UML models," *Journal of Systems and Software*, vol. 79, no. 3, pp. 339–352, Mar. 2006.

[8] M. O. Elish and D. C. Rine, "Investigation of metrics for object-oriented design logical stability," in *Proc. 7th European Conference on Software Maintenance and Reengineering*, Mar. 2003, pp. 193–200.

[9] S. R. Chidamber and C. F. Kemerer, "A metrics suite for object oriented design," *IEEE Transactions on Software Engineering*, vol. 20, no. 6, pp. 476–493, Jun. 1994.

[10] M. P. Robillard, "Topology analysis of software dependencies," *ACM Transactions on Software Engineering and Methodology*, vol. 17, no. 4, pp. 18:1–18:36, Aug. 2008.

[11] V. Rajlich, "A model for change propagation based on graph rewriting," in *Proc. 13th IEEE International Conference on Software Maintenance*, Oct. 1997, pp. 84–91.

[12] T. Apiwattanapong, A. Orso, and M. J. Harrold, "Efficient and precise dynamic impact analysis using execute-after sequences," in *Proc. 27th International Conference on Software Engineering*, May 2005, pp. 432–441.

[13] S. Bouktif, Y.-G. Guéhéneuc, and G. Antoniol, "Extracting change-patterns from CVS repositories," in *Proc. 13th Working Conference on Reverse Engineering*, Oct. 2006, pp. 221–230.

[14] M. Ceccarelli, L. Cerulo, G. Canfora, and M. D. Penta, "An eclectic approach for change impact analysis," in *Proc. 32nd International Conference on Software Engineering*, May 2010, pp. 163–166.

[15] M. D'Ambros, H. C. Gall, M. Lanza, and M. Pinzger, "Analyzing software repositories to understand software evolution," in *Software Evolution*. Springer, 2008.

[16] M. Cataldo, "Dependencies in geographically distributed software development: Overcoming the limits of modularity," Ph.D. dissertation, Carnegie Mellon University, 2007.

[17] N. Tsantalis, A. Chatzigeorgiou, and G. Stephanide, "Predicting the probability of change in object-oriented systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 7, pp. 601–614, Jul. 2005.

[18] M. K. Abdi, H. Lounis, and H. A. Sahraoui, "Predicting maintainability expressed as change impact: A machine-learning-based approach," in *Proc. 21st International Conference on Software Engineering and Knowledge Engineering*, Jul. 2009, pp. 122–128.

[19] S. Mirarab, A. Hassouna, and L. Tahvildari, "Using bayesian belief networks to predict change propagation in software systems," in *Proc. 15th IEEE International Conference on Program Comprehension*, Jun. 2007, pp. 177–188.

[20] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed. Prentice Hall, 2003.

[21] J. A. Whittaker and M. G. Thomason, "A markov chain model for statistical software testing," *IEEE Transactions on Software Engineering*, vol. 20, no. 10, pp. 812–824, Oct. 1994.

[22] J. F. Bowring, J. M. Rehg, and M. J. Harrold, "Active learning for automatic classification of software behavior," in *Proc. ACM SIGSOFT International Symposium on Software Testing and Analysis*, Jul. 2004, pp. 195–205.

[23] L. Cheung, R. Roshandel, N. Medvidovic, and L. Golubchik, "Early prediction of software component reliability," in *Proc. 30th International Conference on Software Engineering*, May 2008, pp. 111–120.

[24] P. Raghavan and C. D. Tompson, "Randomized rounding: A technique for provably good algorithms and algorithmic proofs," *Combinatorica*, vol. 7, no. 4, pp. 365–374, Dec. 1987.

[25] L. C. Briand, J. Wüst, and H. Lounis, "Using coupling measurement for impact analysis on object-oriented systems," in *Proc. 15th IEEE International Conference on Software Maintenance*, Aug. 1999, pp. 475–482.

[26] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, Dec. 1945.

[27] G. Antoniol, G. Canfora, G. Casazza, and A. D. Lucia, "Identifying the starting impact set of a maintenance request," in *Proc. 4th European Conference on Software Maintenance and Reengineering*, Mar. 2000, pp. 227–230.