Change Impact Analysis with Stochastic Dependencies

Sunny Wong, Yuanfang Cai, and Michael Dalton
Drexel University
Philadelphia, PA, USA
{sunny, yfcai, mcd45}@cs.drexel.edu

ABSTRACT

Researchers have shown that software change impact analysis (IA) can be improved by investigating logical coupling embodied in revision histories. However, prevailing history-based IA techniques do not take the temporal dimension of revision history into consideration to account for design evolution. In this paper, we formalize logical coupling as a stochastic process using a Markov chain model. By varying the length and the smoothing function of the model, we define a family of *stochastic dependencies*. Each member of this family weights historical data differently according to their recency and frequency. To assess its utility, we conduct IA on 86 releases of five open source systems by using 16 members of the stochastic dependency family and compare with the result of several existing approaches. Our results show that in four out of the five systems, our stochastic-based IA technique is the most accurate.

Categories and Subject Descriptors

D.2.10 [**Software Engineering**]: Design—*change impact analysis*

General Terms

probability theory, change impact analysis, change propagation, stochastic process

Keywords

impact analysis, stochastic dependency, logical dependency, Markov chain

1. INTRODUCTION

During software maintenance, changes are often introduced to fix bugs and accommodate new requirements. Understanding the impact scope of changes is a basic step of estimating the effort required. Traditionally, change impact analysis[1] (IA) [4] is often performed based on the software dependency structure. One inefficiency of conducting impact analysis based on software structure is that semantically coupled components may not structurally depend on each other. Recently, researchers have leveraged revision histories to more effectively identify semantically coupled components (called logical or evolutionary coupling) by checking how components historically change together.

Although historical data has been shown to be a useful resource for IA, a questions that arises is, *how long of a history is enough?* If taking a very long history into consideration, the software may have changed so that the coupling discovered in the distant past may not be valid anymore [13, 19]. If the history is too short, the revision data may not be sufficient to reveal semantic coupling. The answer to this question also depends on the nature and maturity of the specific system. The length of history needed to accurately predict 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 impact analysis and address the problem that existing history-based IA techniques do not account for the temporal dimension or the evolution of software architectures, we contribute a formalized generalization of logical dependencies based on probability theory, which we call *stochastic dependencies*. This formalization defines a family of dependency types. Each member of the family weights history data differently according to their recency and frequency. Our stochastic dependency family can be used to augment prevailing change impact analysis by selecting the member of the family that fits the project.

Our stochastic dependency framework is based on the Markov chain model [12], a probabilistic model often used in artificial intelligence for predicting the expected result of a random variable/event. Our assumption is that each transaction[2] in revision history can be modeled as a random variable. To address the temporal issue of changing dependencies, 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.

[1]Change impact analysis is also called change propagation analysis and change scope analysis.

[2]A transaction is defined as an atomic commit in a version control repository (e.g. Subversion). For repositories that do not natively support the concept of transactions (e.g. CVS), heuristics and techniques have been developed to reconstruct transactions.

Submitted to the 33rd International Conference on Software Engineering (ICSE).

From the Markov chain, we can compute the probability that two components are logically dependent (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 parameters to our framework, that is, 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 86 releases of five open source systems by varying the selection of parameter values of stochastic dependencies. Our results show that the stochastic-based change impact analysis outperforms two traditional structure-based IA approaches, and outperforms existing history-based IA approaches in four out of the five systems. The only system, where the stochastic-based approach is not more accurate, is Eclipse, which has shown be more stable than average software systems [17]—hence, its historical data remains valid for a long time.

The rest of this paper is organized as follows: Section 3 formally defines stochastic dependencies and how to use them for impact analysis. Section 4 presents our evaluation method and empirical results. Section 5 discusses threats to validity, and future work. Section 6 concludes.

2. RELATED WORK

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

2.1 Impact Analysis

Software change impact analysis [4] is the method of determining what software elements[3] are likely to change when, or after, a certain set of elements (called the starting impact set) changes. Numerous IA techniques have been proposed. Next, we discuss several representative categories of IA approaches.

Structure-based Impact Analysis.

Various IA techniques have been developed using software dependency structures. For example, Briand et al. [7] and Elish and Rine [14] have proposed IA techniques based on the metrics of Chidamber and Kemerer [11]. Robillard [23] and Rajlich [22] presented algorithms based on dependency graph structures. Although Robillard's algorithm is not specifically designed for impact analysis, it can be used for IA based on the weights assigned to each element. To capture impact that cannot be easily identified through syntactic analysis, some IA approaches (e.g. Arisholm et al. [3], Apiwattanapong et al. [2]) use dynamic (instance-based rather than class-based) analysis to measure coupling. While impact analysis techniques based on structural coupling can be effective, some semantic relationships between software elements can only be revealed through analysis of revision history.

[3]We use the term *software element* to refer to software components as various abstraction levels (e.g. class, method, module, set of files).

History-based Impact Analysis.

Similar to our approach, Ying et al. [27] and Zimmermann et al. [28] also leverage revision histories to perform IA based on association rules that identify logical (or evolutionary) coupling [15] between software elements. An association rule, of the form $a \Rightarrow b$, predicts 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.

Logical 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. [5] and Ceccarelli et al. [9]. Different from our approach of using Markov chains, they use techniques to infer cause-effect relationships from the temporal data, rather than to filter out obsolete history data.

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 the elements continues to exist. To address this issue, the confidence heuristic can be used in conjunction with support. However, 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.

Probablistic Approaches.

Recently, several impact analysis techniques have been proposed that are based on probability theory. For example, Tsantalis et al. [25] define probabilities of impact based on structural relations between software elements in object-oriented designs. Abdi et al. [1] use a Bayesian network (a probabilistic model) with structural coupling metrics to construct the inference model. Mirarab et al. [18] 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.

2.2 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 [24]) and various other computing fields (e.g., web search engine algorithm [12]). Markov processes have also been applied to software engineering (e.g., generate test inputs [26], classify software behavior [6], predict component reliability [10]). To the best of our knowledge, Markov processes have not yet been applied to computing component dependencies for IA.

3. STOCHASTIC DEPENDENCY FAMILY

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

3.1 Definitions and Background

Let $E = \{e_i\}_{i=1}^{N}$ be the set of software elements of the system under consideration, where N is the total number of elements.[4] 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 $T^{(e)} = \{t_i^{(e)} \in T \mid e \in t_i^{(e)}\}$ 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. Let $M^{(e)}$ be the length of $T^{(e)}$.

Given two elements $a, b \in E$, let $C^{(a,b)} = \{X_i^{(a,b)}\}_{i=1}^{M^{(a)}}$ 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 $\Pr(X_\tau^{(a,b)} = 1)$. 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 of range $[0, 1]$. 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 probabilistic value of the stochastic dependencies, we can reason about which elements are expected to be in the 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 the maintainer. For example, we can define a minimum probability threshold and predict all elements above that threshold to be in the impact scope. Alternatively, we can use randomized rounding [21] to select the likely impacted elements. Next, we formally define a method for computing stochastic dependency values.

3.2 History-based 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 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 the evolution of software, we consider only the latest k transactions involving the starting impact set 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 affects of various k values in our evaluation in Section 4.

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 [20] 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 history-based *stochastic dependencies* that is also based on a linear combination of previous states:

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

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 predicted 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,[5] λ, to account for software evolution and weigh more recent transactions more heavily than older transactions. Table 1 shows several functions that can be used as the smoothing function, λ. 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 traditional logical dependency definition. The second column shows a linear function, indicating that the usefulness of a transaction in terms of determining stochastic dependency

[4] 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.

[5] 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.

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 logical dependencies that only use confidence to detect such changes.

Our stochastic dependency definition is a generalization of traditional logical 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 logical dependency, we can compute the logical 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 logical dependency (b, a) is said to exist at time τ if:

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

3.3 Example

We use a concrete, hypothetical example to demonstrate how to conduct change impact analysis based on stochastic dependencies. Suppose we have the following sequence of transactions in our revision history: $\{a, b\}$, $\{a, c\}$, $\{d\}$, $\{a, b\}$, $\{a\}$, $\{a, b, c\}$, $\{b, d\}$, $\{a\}$, $\{a, d\}$, $\{c\}$, $\{a, c\}$, $\{a\}$. Now we want to perform IA for a anticipated change to c. If using a linear λ sequence with $k = 5$, 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 determine if any of the other files $\{a, b, d\}$ are expected to be in the impact scope of changing c by applying the stochastic dependency formula:[6]

	$\{a, c\}$	$\{c\}$	$\{a, b, c\}$	$\{a, c\}$	Pr
a	$\lambda_1 \cdot 1$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 1$	$\lambda_4 \cdot 1$	0.6667
b	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 1$	$\lambda_4 \cdot 0$	0.2
d	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 0$	$\lambda_4 \cdot 0$	0

If we use a threshold-based rounding scheme with the minimum threshold to be 0.5, 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 stochastic dependency formula to compute the probabilities:

	$\{a\}$	$\{a, c\}$	$\{a, d\}$	$\{a\}$	$\{a, b, c\}$	Pr
b	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 0$	$\lambda_4 \cdot 0$	$\lambda_5 \cdot 1$	0.0667
c	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 1$	$\lambda_3 \cdot 0$	$\lambda_4 \cdot 0$	$\lambda_5 \cdot 1$	0.3333
d	$\lambda_1 \cdot 0$	$\lambda_2 \cdot 0$	$\lambda_3 \cdot 1$	$\lambda_4 \cdot 0$	$\lambda_5 \cdot 0$	0.2667

Using the same minimum threshold of 0.5, we would expect the change to a to not impact any other elements in this system.

[6]As a performance consideration, we could directly ignore d because it does not occur in any transactions with c.

4. EVALUATION

To evaluate the effectiveness of using stochastic dependencies to conduct change impact analysis, 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 logical-coupling-based IA techniques. Our evaluation aims to answer the following questions:

- Q1: Are stochastic-based IA approaches always more accurate than traditional structure-based IA techniques? We compare the accuracy of each stochastic dependency type to the accuracy of structure-based techniques for each subject system.
- Q2: Are stochastic-based IA approaches always more accurate than prevailing logic-based IA techniques, which do not take temporal effects into consideration? As we mentioned before, the prevailing logical coupling definition is a special case of our stochastic dependency family, and we conduct multiple dimensional comparisons in the similarly way of answering Q1.
- 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 too long or too short of a history both will negatively impact the IA performance.
- Q4: How does the selection of a decreasing 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.

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

4.1 Subjects

We select five open source projects with different sizes and from different domains as the subjects of our evaluation.

Log4J Log4J[7] is a popular logging framework for the Java programming language.

Hadoop Common Hadoop is a Java-based distributed computing framework; Hadoop Common[8] provides the shared components and functionality used by other Hadoop sub-projects.

JEdit JEdit[9] is a text editor that supports syntax highlighting of source code and the ability to write macros in various scripting languages.

Apache Ant Ant[10] is an automated software build tool for Java programs.

Eclipse JDT Eclipse JDT[11] is an AST analysis toolkit in the Eclipse IDE.

Table 2 shows some basic information of the systems that we analyzed.

[7]http://logging.apache.org/log4j/
[8]http://hadoop.apache.org/common/
[9]http://www.jedit.org
[10]http://ant.apache.org
[11]http://www.eclipse.org/jdt/

Table 1: Example Lambda Sequences

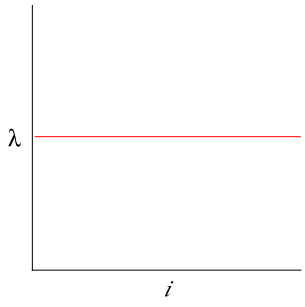
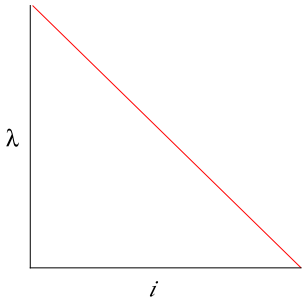
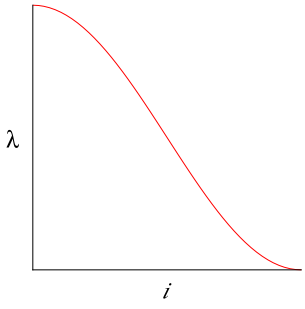
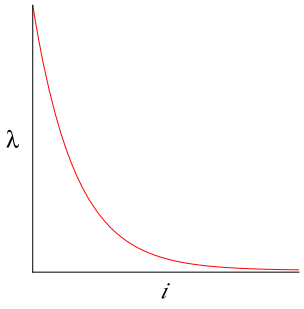
Constant	Linear	Sinusoidal	Exponential
			
$\frac{1}{k}$	$\frac{2(k-i+1)}{k^2+k}$	$\frac{\cos\left(\pi k^{-1}(i-1)\right)+1}{k+1}$	$2^{-i}+\frac{2}{k2^{k+1}}$

Table 2: Subject System Information

Subject	# Vers	Repository Dates	KSLOC	# Trans
Log4J	11	12/14/00–6/10/07	10–15	3550
Hadoop Common	20	2/3/06–12/18/09	13–36	9319
JEdit	12	9/2/01–2/19/10	63–98	17339
Apache Ant	20	1/13/00–8/5/10	9–125	14554
Eclipse JDT	23	5/2/01–7/15/10	137–278	61110

4.2 Evaluation Procedure

For each subject system, we extracted transaction information from its Subversion (SVN) repository. Since Eclipse JDT uses CVS instead of SVN, we used the *cvs2svn*[12] tool to convert the repository to SVN and obtained transaction information. For each transaction in revision history, we performed impact analysis on it using one of the files as the starting impact set. Assuming each transaction is a single change task, an ideal IA approach would be able to predict 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. [28], we ignore transactions with more than 30 files as they are likely branch or merge operations in the revision history and not meaningful change tasks. Since we are interested in computing impact scope from a starting impact set, we also ignore transactions with a single file.

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 comparison.

[12]http://cvs2svn.tigris.org/

Next we introduce the existing IA techniques against which we compare our stochastic-based IA technique.

$$
\begin{aligned}
precision &= \frac{\#\ correct}{\#\ predicted} \\
recall &= \frac{\#\ correct}{transaction\_size} \\
F_1 &= \frac{2 \times precision \times recall}{precision + recall}
\end{aligned}
$$

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. [8]. 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 comparison, 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 measure as the *static* dependency approach. The other approach we compare against is the work of Robillard [23]. 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 a dependency graph. In this section, we refer to this measure as the *topology* dependency approach.

To perform IA on a starting impact set using these measures, we first compute the dependency value 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.

Logic-based Impact Analysis.

To answer the second evaluation question, we compare the IA accuracy of stochastic dependency against that of traditional logical dependencies [28]. We perform logic-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 prior to processing the transactions for each software release.

Stochastic-based Impact Analysis.

We use the four λ sequences presented in Table 1 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 logical 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 at large as the threshold. Also similar to the structural dependency approaches, we find the best minimum threshold for each software release.

4.3 Results

Tables 4, 5, 6, 7, and 8 are the results of performing IA using the 16 stochastic dependency members on the five subjects. Table 3 shows the average F_1 scores for each subject system using the logic-based (column *Logic*), topology-based (column *Topology*), static dependency-based (column *Static*), and stochastic-based (with $k = 10$) IA techniques respectively. The shaded cell in each row highlights the approach with the best average F_1 score over all the software releases.

Q1: Comparing with structure-base IA.

These tables show that the accuracy of both structure-based IA (columns *Topology* and *Static*) for all the subjects are lower than any member of the stochastic-based IA approaches, and are also lower than traditional logic-based impact analysis. We use Apache Ant as an example to elaborate more details.

For 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. Figure 1 shows the average F_1 score for each release of Ant, using several dependency types. This figure shows that using the stochastic dependency with sinusoidal λ sequence and a k value of 10 achieves highest accuracy of all the other dependency types in all the releases we studied.

Q2: Comparing with traditional logic-based IA.

Table 3 show that out of the five subject systems, using stochastic-based IA significantly improved the accuracy in three of the systems (Log4J, Hadoop, and Ant) over logical dependencies. Stochastic-based and logic-based IA achieved

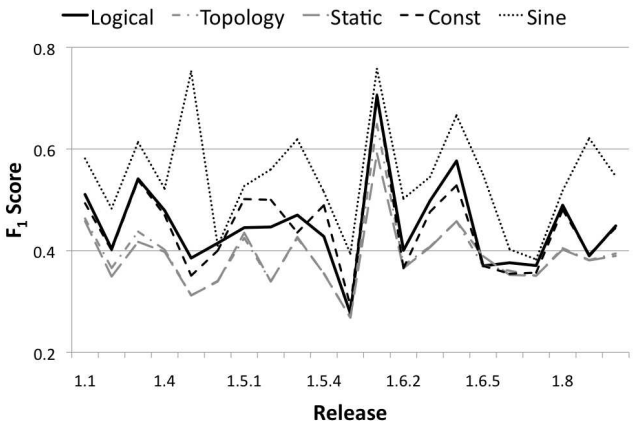


Figure 1: Average F_1 per Release: Apache Ant

comparable IA accuracies with JEdit. With Eclipse JDT, the logical dependencies achieved significantly better accuracy than all the other approaches. The results suggest that the stochastic-based IA *often* perform as well or better than traditional logical dependencies, but not always. Gathering additional empirical evidence for these questions remains ongoing work.

Since the stochastic-based and structure-based IA achieved comparable accuracy and all performed worse than the logic-based IA for Eclipse JDT, we further examined the Eclipse results for an explanation. Using logical dependencies, we obtained an average precision of 0.4494 and average recall of 0.4850. Using stochastic-based IA (all family members achieved very similar results), we obtained an average precision of 0.7871, which is much higher than that of the logic-based approach, and an average recall of 0.2243, which is much lower.

This differences in these measures indicate that the semantic coupling within the system remains stable for a long period of time. We use an example to explain the differences. For example, suppose files `A` and `B` are semantically coupled, and are changed together frequently in the early stages of revision history. Then for a period of time after that, `A` is modified without changes to `B`. During this time, the logic-based IA would continually predict that `B` is in the change scope of `A` because the support value for `B` never decreases.

In comparison, a stochastic-based IA approach would initially also predict that `B` is in the impact scope of `A`, but after some time, it would (incorrectly) infer that the dependency between `A` and `B` has been refactored, and stop predicting `B` in the impact scope. Hence, if later changes to `A` impact `B` again, the stochastic-based approach would suffer a low recall. The continual prediction of `B`, by the logic-based approach, during the period that `A` is modified without `B` decreases the precision of the logic-based approach, as we see with Eclipse. Hence, we hypothesize that the stability of dependencies in Eclipse is why logic-based IA outperforms the other techniques.

This observation implies that in order to achieve more accurate IA results, different types of dependencies should be applied to different types of systems. The stochastic-based IA approach is shown to be better for software systems that are more volatile and often subject to refactoring.

Table 3: Results

Subject	Average F_1 Score						
	Logic	Topology	Static	Stochastic ($k = 10$)			
				Const	Line	Sine	Expon
Log4J	0.3733	0.3090	0.3048	0.3666	0.4119	0.4178	0.4158
Hadoop Common	0.4652	0.3668	0.3717	0.4420	0.4854	0.4873	0.5229
JEdit	0.3844	0.3512	0.3506	0.3950	0.3993	0.3951	0.3943
Apache Ant	0.4355	0.3760	0.3726	0.4324	0.4900	0.4900	0.4828
Eclipse JDT	0.3754	0.3311	0.3310	0.3313	0.3313	0.3313	0.3313

Q3: Effects of k.

Reading horizontally in Tables 4, 5, 6, 7, and 8, we can see how the F_1 results change with the increase in k. Again, Eclipse JDT appear to be different from other systems in that the F_1 score does not change with the changes to k. Again, this is consistent with our previous analysis that Eclipse is much stabler than average software systems.

Table 4: F_1 Scores for $\langle \lambda, k \rangle$ Pairs: Apache Ant

λ \ k	5	10	20	40
Constant	0.4578	0.4323	0.4109	0.3978
Linear	0.4833	0.4900	0.4574	0.4347
Sinusoidal	0.4824	0.4900	0.4583	0.4356
Exponential	0.4817	0.4822	0.4822	0.4822

For all other four systems, the accuracy first increases with the increase of k and then decreases as k continues increasing. For example, in Apache Ant, using a linear λ sequence with $k = 5$ (second row, first column in Table 4), we achieved an average F_1 of 0.4833. Increasing the length of history to $k = 10$ (second row, second column), we improved the accuracy to 0.4900. However, further increasing the length of history to 20 and 40, caused the accuracy to begin decreasing. This increase in accuracy and then decrease in accuracy from increasing k was consistent in the four subject systems, but the point of decrease varied among the systems. For most of the systems, the accuracy began decreasing at $k = 10$, and for JEdit, the accuracy began to decrease at $k = 20$. These changes in accuracy allows us to positively answer evaluation question Q3: changing the amount of historical data analyzed (k) *does* affect the accuracy of stochastic-based IA in an unstable system.

Table 5: F_1 Scores for $\langle \lambda, k \rangle$ Pairs: Hadoop

λ \ k	5	10	20	40
Constant	0.4662	0.4420	0.4276	0.4084
Linear	0.5099	0.4854	0.4582	0.4335
Sinusoidal	0.5094	0.4873	0.4601	0.4340
Exponential	0.5188	0.5299	0.5227	0.5227

Q4: Effects of λ.

Reading vertically in these tables, we can see how the λ sequences affect accuracy. For example, in Apache Ant, with a constant λ and $k = 5$, we achieved an average F_1 of 0.4578. Keeping the same length of history, but switching to a sinusoidal λ sequence, we increased the accuracy to 0.4824. Switching to a linear sequence slightly increased the accuracy to 0.4883. For Apache Ant, the linear and sinusoidal sequences both achieved similar accuracies with the best combination being a sinusoidal sequence and 10 transactions (shown as a shaded cell in the table). This trend is consistent with all the system except for Eclipse JDT, in which the λ does not seems to be making any effects due to its stability over time.

Table 6: F_1 Scores for $\langle \lambda, k \rangle$ Pairs: JEdit

λ \ k	5	10	20	40
Constant	0.3775	0.3950	0.3954	0.3820
Linear	0.3892	0.3993	0.4096	0.3989
Sinusoidal	0.3890	0.3951	0.4056	0.3975
Exponential	0.3935	0.3943	0.3943	0.3943

Table 7: F_1 Scores for $\langle \lambda, k \rangle$ Pairs: Log4J

λ \ k	5	10	20	40
Constant	0.3845	0.3666	0.3539	0.3320
Linear	0.4134	0.4119	0.4039	0.3558
Sinusoidal	0.4169	0.4178	0.3956	0.3570
Exponential	0.4113	0.4158	0.4158	0.4158

Among the other four systems we evaluated, the sinusoidal and exponential λ sequences were the best performing stochastic dependency types. In addition, the decreasing λ sequences always outperformed the constant λ sequence. Hence, we can positively answer evaluation question Q4: using a decreasing λ sequence to emphasize more recent historical data *does* improve the accuracy of stochastic-based IA when the system is not stable.

5. DISCUSSION

This section discusses some threats to validity to our framework and evaluation. We also describe some possible future work with stochastic dependencies.

Although logical dependencies are a type of stochastic de-

Table 8: F_1 Scores for $\langle \lambda, k \rangle$ Pairs: Eclipse

λ \ k	5	10	20	40
Constant	0.3313	0.3313	0.3313	0.3313
Linear	0.3313	0.3313	0.3313	0.3313
Sinusoidal	0.3313	0.3313	0.3313	0.3313
Exponential	0.3313	0.3313	0.3313	0.3313

pendencies that use a constant λ sequence, not all stochastic dependencies with constant λ sequences are logical dependencies. For a stochastic dependency type to predict the same impact scope as logical dependencies, we need to have a constant λ sequence *and* k must be the number of transactions that involves the 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 logic-based IA. To achieve the same results as logic-based IA, we would have needed to vary the value of k for each starting impact set in each transaction.

5.1 Threats to Validity

Since we only applied our approach to five Java-based, object-oriented subject 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.

As with any technique that derives software dependencies from transactions in revision history, our approach assumes that the transactions represent cohesive work units. In other words, if developers only commit to the revision control system once a week, 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 logical dependencies indicate that transaction often are these cohesive work units and often do indicate semantic coupling of software elements.

Our stochastic dependency 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 accurate impact analysis results.

We only compared the stochastic-based IA to a few existing approaches, and there are many other types of IA. Our study does show that the temporal dimension of history data does impact the accuracy of a change impact analysis technique. Our experiment also shows that systems with different evolution characteristic and stability needs different types of dependencies to effectively conduct IA.

5.2 Future Work

Exploring methods to improve stochastic-based IA accuracy in general is an ongoing work. In particular, we are looking into techniques to improve the IA accuracy in systems like Eclipse. Precisely characterizing the evolution pattern of different systems and applying the best IA methods accordingly is also future work.

The parameters of stochastic dependency allow for a large number of specific dependency types to be defined. 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. Exploring techniques to automatically construct λ and k values for a given software system is also a possible future work.

Pure history-based stochastic dependency types represent only one sub-family of stochastic dependencies. For example, we are exploring ways to integrate structural dependency measures with history-based dependency to enable the uniform use of stochastic dependencies for IA, with systems where revision history data is limited or unavailable. In addition, we are also exploring the use of more complex probabilistic models (e.g. hidden Markov models, dynamic Bayesian networks [24]) in place of the Markov chain model. Defining and evaluating other sub-families of stochastic dependencies is a future work.

While efficient data structures and algorithms (e.g., frequent pattern (FP)-trees [16]) have been developed for mining of association rules for logical dependency analysis, such techniques are not yet available for our stochastic dependency framework. In our evaluation, the performance of stochastic-based IA for each transaction was a constant factor slower than the logic-based IA, not including the time for preprocessing logical dependencies (e.g., building the FP-tree). Developing data structures and algorithms to improve the performance of computing stochastic dependencies remains a future work.

6. CONCLUSION

Using history-based logical coupling to conduct change impact analysis in software maintenance has gained popularity recently. The temporal dimension of the historical data influences the accuracy of such analysis because data in distant history may not be valid, especially for a system that is volatile and subject to frequent refactoring. Our stochastic dependency family formally accounts for the characteristic of evolution history. By varying the smoothing function and the length of history, a developer can select the combination of parameters 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, the stochastic-based approach obtains better IA accuracy for all the subject systems except Eclipse, which has a more stable architecture in which distant history remains valid longer than other systems. The data suggests that stochastic dependencies are more effective in more volatile systems that undergo frequent refactoring.

7. ACKNOWLEDGEMENTS

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

8. REFERENCES

[1] 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*, pages 122–128, July 2009.

[2] 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*, pages 432–441, May 2005.

[3] E. Arisholm, L. C. Briand, and A. Foyen. Dynamic coupling measurement for object-oriented software. *IEEE Transactions on Software Engineering*, 30(8):491–506, Aug. 2004.

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

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

[6] 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*, pages 195–205, July 2004.

[7] L. C. Briand, Y. Labiche, L. O'Sullivan, and M. M. Sówka. Automated impact analysis of UML models. *Journal of Systems and Software*, 79(3):339–352, Mar. 2006.

[8] 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*, pages 475–482, Aug. 1999.

[9] 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*, pages 163–166, May 2010.

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

[11] S. R. Chidamber and C. F. Kemerer. A metrics suite for object oriented design. *IEEE Transactions on Software Engineering*, 20(6):476–493, June 1994.

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

[13] S. G. Eick, T. L. Graves, A. F. Karr, J. S. Marron, and A. Mockus. Does code decay? assessing the evidence from change management data. *IEEE Transactions on Software Engineering*, 27(1):1–12, Jan. 2001.

[14] 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*, pages 193–200, Mar. 2003.

[15] 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*, pages 190–197, Nov. 1998.

[16] J. Han, J. Pei, Y. Yin, and R. Mao. Mining frequent patterns without candidate generation: A frequent-pattern tree approach. *Data Mining and Knowledge Discovery*, 8:53–87, Jan. 2004.

[17] T. Mens, J. Fernández-Ramil, and S. Degrandsart. The evolution of Eclipse. In *Proc. 24th IEEE International Conference on Software Maintenance*, pages 386–395, Oct. 2008.

[18] 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*, pages 177–188, June 2007.

[19] D. L. Parnas. Software aging. In *Proc. 16th International Conference on Software Engineering*, pages 279–287, May 1994.

[20] A. E. Raftery. A model for high-order markov chains. *Journal of the Royal Statistical Society*, 47(3):528–539, 1985.

[21] P. Raghavan and C. D. Tompson. Randomized rounding: A technique for provably good algorithms and algorithmic proofs. *Combinatorica*, 7(4):365–374, Dec. 1987.

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

[23] M. P. Robillard. Topology analysis of software dependencies. *ACM Transactions on Software Engineering and Methodology*, 17(4):18:1–18:36, Aug. 2008.

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

[25] N. Tsantalis, A. Chatzigeorgiou, and G. Stephanide. Predicting the probability of change in object-oriented systems. *IEEE Transactions on Software Engineering*, 31(7):601–614, July 2005.

[26] J. A. Whittaker and M. G. Thomason. A markov chain model for statistical software testing. *IEEE Transactions on Software Engineering*, 20(10):812–824, Oct. 1994.

[27] 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*, 30(9):574–586, Sept. 2004.

[28] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Transactions on Software Engineering*, 31(6):429–445, June 2005.