A Lightweight Approach to Technical Risk Estimation via Probabilistic Impact Analysis

Robert J. Walker, Reid Holmes, Ian Hedgeland
Department of Computer Science
University of Calgary
Calgary, Alberta, Canada
{rwalker, rtholmes, ianh}@cpsc.ucalgary.ca

Puneet Kapur, Andrew Smith
Chartwell Technology Inc.
Calgary, Alberta, Canada
{pkapur, asmith}@chartwelltech.com

Technical report 2006-817-10

2 February 2006

ABSTRACT

An evolutionary development approach is increasingly commonplace in industry but presents increased difficulties in risk management, for both technical and organizational reasons. In this context, technical risk is the product of the probability of a technical event and the cost of that event. This paper presents a technique for more objectively assessing and communicating technical risk in an evolutionary development setting that (1) operates atop weakly-estimated knowledge of the changes to be made, (2) analyzes the past change history and current structure of a system to estimate the probability of change propagation, and (3) can be discussed vertically within an organization both with development staff and high-level management. A tool realizing this technique has been developed for the Eclipse IDE.

1. INTRODUCTION

Making good decisions is key to successful development, yet remains a difficult task in an evolutionary development setting. Many organizations struggle to consider both "managerial" and "technical" factors on an objective basis. "Managerial" factors that must be considered include predictions of market forces, conflicting stakeholder interests, and budgetary constraints [32]; "technical" factors include the ease with which proposed extensions can be accommodated by the current software structure [4]. The organizational difficulties arise from the fact that those with the decision making roles typically have the least access to detailed technical knowledge [4, 30], while those with the detailed technical knowledge have the least ability to influence decisions about the direction of development [25]. To bridge this gap, a means for assessing the technical risk of proposed changes is needed that can be audited, for the sake of objectivity, and can serve as the basis for vertical communication within an organization.

Some authors emphasize the risk of introducing flaws into software [17] or causing the failure of a software project [4]. While these are clearly significant threats, we wish to consider the risk of any modification task in a more general sense: the risk of an event is defined as the product of the probability of the event and the cost of the event should it happen. Thus, we can see that even unlikely events with very high costs can result in unacceptably high risks. We expect that an analyst must have at least an approximate sense of key points within a software system that are likely to change. The task then becomes one of determining to what extent these key changes are likely to cause other changes in a cascading sequence: probabilistic change impact analysis. A variety of approaches to change impact analysis have been proposed in the past, but none is appropriate to our context. Many of these techniques expect to be provided with implementations of the initial changes in order to perform their analyses [19, 12, 3]; this is not practicable at an early planning stage. Other change impact techniques exist that try to support decision-making at early phases; some of these expect complete and accurate documentation to be available for analysis [34], others require detailed grammars to be specified as an input to the analysis [18], and still others depend solely on the qualitative judgment of a set of experts [14, 28]. None of these techniques copes well with the assessment and communication of technical risk within an evolutionary development and within an organization in which decision making tends to be separated from detailed, technical knowledge.

Instead, we propose a decision support technique (1) that supports simple entry and update by an analyst of even weakly-estimated knowledge of likely changes, (2) that automatically performs change impact estimation based on such "educated guesses," and (3) that can be used as a basis for vertical communication within an organization. Our technique works from three inputs: a structural dependence graph that is automatically extracted from a software project; change history data for that project that is automatically extracted from a CVS repository; and an indication by an analyst of the key components where a proposed feature is expected to result in a definite change. The technique then uses the structural dependence graph and change history data to estimate the probability that these definite changes will propagate to the rest of the system. Our algorithms are defined in such a way that changing the indication of the definite changes is simple and the results recomputed in real time. The technique has been implemented as a plugin to the Eclipse IDE and deployed to our industrial collaborators for an initial evaluation.

The remainder of the paper is structured as follows. Section 2 describes our concrete tool implemented as an Eclipse plugin. Section 3 outlines the data extraction steps used to drive the technical risk estimation algorithms. Section 4 describes and analyzes the theoretical model on which our technique is based. An preliminary, informal industrial evaluation has been performed and is described in Section 5. An analysis of the potential weaknesses of this approach, future work, and remaining issues are discussed in Sec-

tion 6. Section 7 considers related work. The contributions of this paper are a theoretical model for lightweight, probabilistic change impact estimation and a discussion of how this model can be used for decision-support in an industrial context.

2. THE TRE TOOL

Technical risk estimation is a process of specifying starting points for changes and estimating the likely propagation of those changes to the rest of the system. We have implemented a tool, named TRE, to support this process as a plugin to the Eclipse integrated development environment. The TRE tool analyzes structural dependencies between Java files within a project, plus historical data from a CVS repository regarding old versions of those files, to perform its estimations. Analysis is performed at the granularity of types; we consider alternatives in Section 6.

Four steps are needed to perform technical risk estimation: (1) extraction of dependency structure from the project source; (2) extraction of change history data from a CVS repository; (3) creating a conditional probability (CP) graph model (or loading an existing CP graph model); and (4) interacting with the CP graph model to make technical risk estimates. The first three steps are supported via Eclipse wizards; Figure 1 shows an example.

Extraction of dependency structure operates on one or more Eclipse projects residing in the workspace, as specified by the analyst. Currently, dependency analysis is performed only on Java files. The result of this extraction is an XML file representing the dependency graph for the types declared within the project. Dependencies on types external to the project are noted as such. External types are considered immutable for the purposes of technical risk estimation. Our interpretation of "dependency" is explained further in Section 3.1.

Extraction of change history data operates on a module in the CVS repositories that the workspace records (these can be viewed and modified via the standard CVS Repositories view of Eclipse). The result of this extraction is an XML file representing the inferred atomic change sets that have occurred in the repository or repositories specified by the analyst. The inference process performed by the tool is described in Section 3.2.

A CP graph model can be generated from any structural dependency graph and any change history data. The CP graph is effectively a structural dependency graph annotated with probabilities on the dependency edges; each of these represents the conditional probability that a change to the target node will result in a change to the source node. The way in which these conditional probabilities are computed is described in Section 3.3. The CP graph is also written to an XML file; this file may be reloaded at later times to continue analysis.

Finally, the technical risk graph model may be used to estimate the risk of performing proposed changes to a project. The analyst indicates which types declared in the project will be the seed points for change. TRE then estimates the technical risk by propagating these seed points through the remainder of the CP graph according to the conditional probabilities annotating the dependencies. Details of the algorithms used are described in Section 4. The risk of changing a type is defined as the product of the probability of changing that type and the cost of changing that type. The total risk of a proposed change is the sum of the risks of changing all the types in the project. Currently, the cost of changing any one type is defined uniformly as 1 for the sake of simplicity. Various alternatives are possible; we consider this issue further in Section 6.

An Eclipse perspective has been implemented to simplify the analyst's interaction with the TRE tool during technical risk estimation. In Figure 2, we see an analysis being performed on a risk

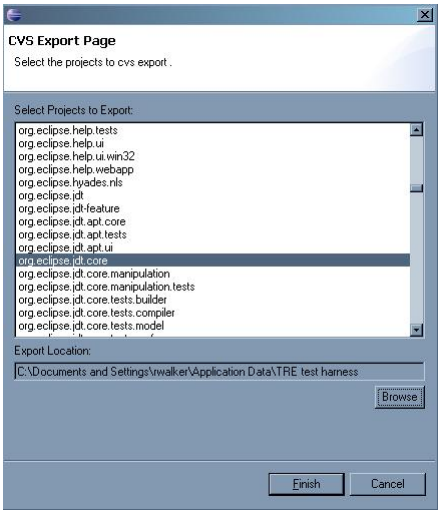

Figure 1: The Eclipse wizard used to select CVS projects for extracting change history data. Here, change history data is about to be extracted from the `org.eclipse.jdt.core` project.

graph representing a portion of `org.eclipse.jdt.core`. The perspective provides two views: on the left is the *Risk Graph Nodes* view, and on the right is the *Technical Risk Results* view. Four buttons are present on the tool bar for the Risk Graph Nodes view: *Extract Structure*, *Extract History*, *Create CP Graph*, and *Select CP Graph*. The first three correspond to the first three steps in the process described above; the fourth allows an existing CP graph to be reloaded.

The final, interactive step of the technical risk estimation process proceeds as the analyst selects or unselects the types displayed in the Risk Graph Nodes view. In the example, we see that `org.eclipse.jdt.core.IField` and `org.eclipse.jdt.core.IMethod` have been selected by the analyst. The computation performed by the tool is displayed both in the Technical Risk Results view and on the status line. In the Technical Risk Results view is a list of the types in the project along with the risk of that type changing and the uncertainty in that risk calculation. The types are sorted and coloured according to the calculated risk: the most intense red (resp. darkest grey in a greyscale rendering) and the highest risk is at the top, shading to white and the lowest risk at the bottom. Note that the risk of the selected types changing is currently defined as 1, and so these appear at the top of the list.

3. DATA PREPARATION

Issues involving the design of the basic data extraction and preparation steps of the tool are considered in this section. We begin with structural dependency extraction in Section 3.1, continue with change history data extraction in Section 3.2, and end with the construction of a simple conditional probability model annotating the structural dependencies in Section 3.3.

3.1 Structural Dependency Extraction

To perform structural dependency extraction, TRE begins by requesting that an AST be constructed for the selected project(s). A subclass of `org.eclipse.jdt.core.jdom.ASTVisitor` determines dependencies by visiting the nodes in this AST. Type `A` is considered dependent on type `somepkg.B` if and only if: (a) the

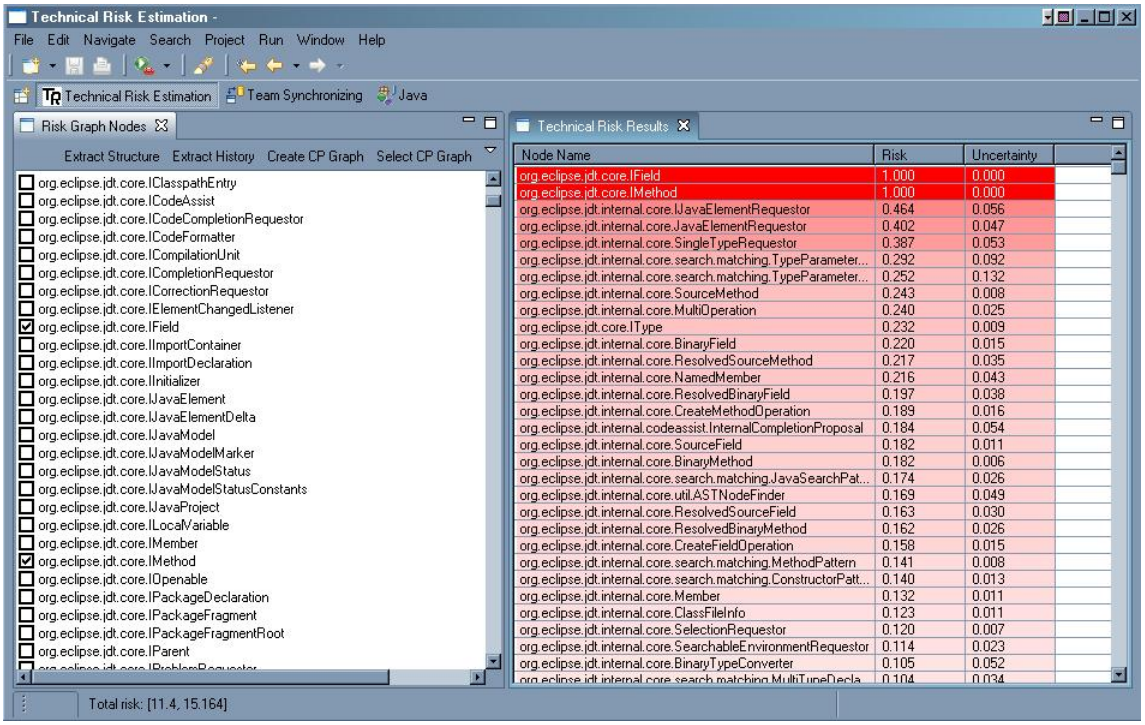

Figure 2: Technical Risk Perspective in Eclipse, showing interaction involving technical risk estimation on JDT core.

name `B` (or `somepkg.B`) is explicitly mentioned by `A` and this definitely resolves to type `somepkg.B`; or (b) `A` contains an expression that definitely resolves to type `somepkg.B` and the result of this expression is used within a larger expression, e.g., the result type of a method invocation resolves to `somepkg.B` and this is used as the prefix to another method invocation. Other definitions of dependency could have been chosen [24]; this one was convenient.

A structural dependency graph $S = (T, D)$ is defined as the result of this process, where T is the set of types declared within the selected project(s) and D is the set of dependencies between types as described above. Self-dependencies are ignored. More formally,

$$D \subseteq T \times T \setminus \{(t,t) | t \in T\}. \quad (1)$$

3.2 Change History Extraction

To perform change history extraction, TRE traverses the specified module within a CVS repository. For each file, the log entries are retrieved (instances of `org.eclipse.team.internal.ccvs.core.ILogEntry`), and information on the author, comment, and timestamp stored for the file revision are recorded.

Atomic change sets are inferred by comparing log entry data. Two files that share the same author and comment, and where the timestamps of adjacent check-ins differ by less than three minutes [16], are considered members of the same atomic change set.

The change history is then recorded as an XML file consisting of a sequence of atomic change sets recording the author and comment, and the earliest of the timestamps, plus the set of files that were modified.

3.3 CP Graph Construction

To construct a conditional probability (CP) graph, the structural dependency graph is initially annotated with data from the change history. The number of revisions v_i to each type t_i is recorded at each node in the structural dependency graph. For each edge $e_{i,j} = (t_i, t_j)$ in the graph, the number of times that t_i and t_j occur in the same atomic change set (noted $v_{ij} = v_{ji}$) is recorded at that edge. The conditional probabilities of a change propagating across each edge $e_{i,j}$ may then be calculated as:

$$\Pr(t_i|t_j)|_{e_{i,j}} \approx v_{ij}/v_i. \quad (2)$$

To deal with the discreteness of the data and its occasional poor quality, the results of Equation 2 must be adjusted. This equation makes two assumptions: (1) that $v_i \geq v_{ij}$; and (2) that $v_i > 0$. If either is false, the result will be an invalid probability, so the result must be clamped to fall within the unit interval $[0, 1]$.

To deal with sparse data, we track an interval of conditional probabilities. In the case of few or no revisions, both the numerator and denominator can be 0; nothing can then be said other than that the probability of a change propagating across an edge is somewhere between 0 and 1. Similarly, we wish to account for the difference in quality between evaluating, e.g., $1/10$ and $100/1000$; while each would result in a calculated conditional probability of 0.1, our confidence in the latter would be much stronger. We borrow a simple approach involved in estimating the accuracy of physical measurements: we consider the error in the numerator and denominator to be ± 0.5. The results are again clamped to the unit interval.

Combining these adjustments, we arrive at the following equations:

$$\min_{\Pr}(t_i|t_j)|_{e_{i,j}} = \quad (3)$$

$$\begin{cases} 0 & \text{if } v_i = v_j = 0, \\ \max\left\{0, \dfrac{v_{ij} - 0.5}{v_i + 0.5}\right\} & \text{otherwise;} \end{cases}$$

$$\max_{\Pr}(t_i|t_j)|_{e_{i,j}} = \quad (4)$$
$$\begin{cases} 1 & \text{if } v_i = 0, \\ \min\left\{1, \frac{v_{ij}+0.5}{v_i-0.5}\right\} & \text{otherwise.} \end{cases}$$

(Continuing our simple example, 1/10 would result in a conditional probability range [0.048,0.158] while 100/1000 would result in a conditional probability range [0.099,0.101]).

The conditional probability graph G is then recorded in an XML file as a structural dependency graph $S = (T, D)$ annotated with the conditional probability intervals for each edge, computed according to Equations 3 and 4. More formally,

$$G = (S, \pi_C : D \mapsto \mathcal{I}), \text{ where} \quad (5)$$
$$\mathcal{I} = \{[m, n] | m \geq 0 \wedge n \leq 1 \wedge m \leq n\}. \quad (6)$$

4. THEORETICAL MODEL

In this section, we consider details of the algorithms underlying the TRE tool and their formal basis. The technical risk estimation step proceeds from a set of types marked as seed points for change by the analyst. The probability of these types changing is defined as 1, although the algorithms below could make use of any constant probabilities attached to these seeds.

In Section 4.1, we describe how the probabilistic change impact can be computed from a conditional probability graph and a set of seed types to provide a probabilistic change impact model. In Section 4.2, the final step of estimating the technical risk is considered.

4.1 Probabilistic Change Impact Analysis

Assume that each modification to a type is due either to an immediate modification (i.e., the type is in the seed set) or to a propagation across a sequence of direct dependencies stemming from such an immediate modification.

Begin with a conditional probability graph G as defined in Section 3.3. Let $\Delta_0 \subseteq T$ be a set representing the seed types that the analyst assumes will be immediately modified. We wish to determine the probable change impact to the remainder of the types in the project, i.e., for every type $t \in T$, we wish to determine the probability that t will be modified given that every type in Δ_0 is modified. We can consider this task to involve the construction of a fuzzy set $\Delta = (T, \mu)$ where $\mu : T \mapsto [0, 1]$ is a membership function indicating the probability that each type will change [36].

From the definition of conditional probability we have that

$$\Pr(t_1 \cap t_2 \cap t_3) = \Pr(t_1|t_2 \cap t_3) \cdot \Pr(t_2|t_3) \cdot \Pr(t_3)$$

for any three types t_1, t_2, t_3. We have the assumption that a modification to a given t can occur only either because $t \in \Delta_0$ (in which case this probability is 1) or because a type upon which t depends has changed. There must exist a path through G from some type δ in Δ_0 to t for t to have changed. For every path $\delta, t_1, t_2, \ldots, t_n, t$, the probability that t must change is:

$$\Pr\left(t \cap \delta \cap \bigcap_{i=1}^{n} t_i\right) = \Pr\left(t \left| \delta \cap \bigcap_{i=1}^{n} t_i \right.\right) \cdot \Pr\left(t_1 \left| \delta \cap \bigcap_{i=2}^{n} t_i \right.\right) \cdot \Pr\left(t_2 \left| \delta \cap \bigcap_{i=3}^{n} t_i \right.\right) \cdot \ldots \cdot \Pr(t_n|\delta) \cdot \Pr(\delta).$$

This equation simplifies significantly because a given type in the path will change only if its predecessor changes; hence, each intersection collapses to the type at the destination of the path:

$$\Pr(t)|_\delta = \Pr(t|t_1) \cdot \Pi_{i=1}^{n-1} \Pr(t_i|t_{i+1}) \cdot \Pr(t_n|\delta) \cdot \Pr(\delta). \quad (7)$$

There may be more than one path that leads from δ to t, and there may be many types in Δ_0 from which paths lead to t. Each path itself yields a fuzzy set Θ_i indicating the probability that a change to its start will propagate to parts on that path. The fuzzy set Δ that we are interested in determining is simply the union over every Θ_i that is yielded from a path beginning at some element of Δ_0. We consider the probability that t will change to be the maximum of the probabilities calculated along all possible paths from a part in Δ_0 to t, which is consistent with the standard definition of the union of fuzzy sets [36].

We can see that the probability of a change propagating from a source type s to a target type t is analogous to finding the longest path between two vertices in a graph. Because the probability will either remain constant or decrease at each step, infinite paths due to cycles do not cause us difficulties. We proceed with a variation on a modified Dijkstra's algorithm [6]. In this variation, probability is analogous to a reciprocal of distance, requiring the calculation of maxima to replace minima, 0 to replace ∞, etc. In the following algorithm, G is a conditional probability graph as defined in Equation 5, and $s \in T$ is a distinguished source type; the output is a function $\rho_s : T \mapsto \mathcal{I}$. The lower bound $\check{\rho}_s$ and upper bound $\hat{\rho}_s$ of ρ_s for each value of T will be calculated separately. Furthermore, we define $D_t = \{q|(q, t) \in D\}$ be the types directly dependent on type t.

CHANGE-PROBABILITY(G, s)

```
1   for every type t ∈ T
2       ρ̌_s(t) := 1, W := T
3       for q ∈ T \ {t}
4           ρ̌_s(q) := 0
5       while W ≠ ∅
6           find some w ∈ W such that ρ̌_s(w) is maximal
7           W := W \ {w}
8           for v ∈ W ∩ D_w
9               ρ̌_s(v) := max{ρ̌_s(v), ρ̌_s(w) × Pr_min(v|w)}
10  for every type t ∈ T
11      ρ̂_s(t) := 1, W := T
12      for q ∈ T \ {t}
13          ρ̂_s(q) := 0
14      while W ≠ ∅
15          find some w ∈ W such that ρ̂_s(w) is maximal
16          W := W \ {w}
17          for v ∈ W ∩ D_w
18              ρ̂_s(v) := max{ρ̂_s(v), ρ̂_s(w) × Pr_max(v|w)}.
```

The proof that CHANGE-PROBABILITY computes the probability that f will change given that s will change is largely identical to that for Dijkstra's algorithm. The direction of inequalities is reversed, and the sums are replaced with products, which does not alter the argument. Thus, $\rho_s(f)$ represents the maximum product of the input conditional probabilities (for a lower bound traversal or for an upper bound traversal) over any path from s to f. Given this and that $\Pr(s) = 1$, $\rho_s(f) = \Pr(f)|_s$ by Equation 7.

An implementation of Dijkstra's algorithm that uses an unsorted working set has running time in $O(|T|^2)$; a more rational priority queue implementation based on Fibonacci heaps reduces this to $O(|T| \log(|T|) + |D|)$ [5]. The changes introduced by CHANGE-PROBABILITY do not alter these arguments.

Because, in general, we will want to know the probability of changing each type given that some set of source types will change, we can consider some alternatives. Our first choice, and the one that we have chosen to implement, repeats the computation of CHANGE-PROBABILITY for each of the types $\delta \in \Delta_0$. We could choose to pre-compute CHANGE-PROBABILITY for each δ, running in total time $O(|T|^2 \log(|T|) + |T||D|)$ for the Fibonacci heap implementation. For cases of large $|T|$, this can impact interactive speeds when most of these computations will never be used. Instead, we compute each only on demand, and cache the results. The best choice is likely to run a background process that uses spare cycles to compute other paths when the analyst has not asked for a specific Δ_0 to be computed. Other alternatives for computing all-pairs shortest paths include the Floyd-Warshall algorithm [9] with a running time in $\Theta(|T|^3)$, and Johnson's algorithm [13] with a running time in $O(|T|^2 \log(|T|) + |T||D|)$; neither of these represent an improvement over the repeated use of CHANGE-PROBABILITY.

We define a *risk graph* R to be a structural dependence graph augmented with the change propagation probability functions ρ_s for all $s \in T$:

$$R = ((T, D), \{\rho_s : T \mapsto \mathcal{I} | s \in T\}). \quad (8)$$

Attempting to access one of these functions that has not been cached results in its computation.

For a given risk graph R and seed set Δ_0, our task is now to determine the fuzzy set (Δ, μ) representing the probability that a change will spread from the seed set. The membership function μ must also be computed with lower ($\check{\mu}$) and upper ($\hat{\mu}$) bounds, for each type. The algorithm below provides this step.

FUZZY-CHANGE-SET(R, Δ_0)

```
1  Δ := T
2  for every type t ∈ T
3      find some maximal ρ̌_δ(t) ∈ {ρ̌_d(t) | d ∈ Δ_0}
4      μ̌(t) := ρ̌_δ(t)
5      find some maximal ρ̂_δ(t) ∈ {ρ̂_d(t) | d ∈ Δ_0}
6      μ̂(t) := ρ̂_δ(t)
```

For an unordered set $\{\rho_s(t) | s \in T\}$, lines 3 and 5 in FUZZY-CHANGE-SET require running time in $O(|T|)$ while the for-loop in line 2 iterates through every type in the graph. An alternative is to store them in a Fibonacci heap, where lines 3 and 5 could each be performed in $O(1)$. (Each of the $|T|$ runs of CHANGE-PROBABILITY would result in a separate value that would require storage; inserting these into Fibonacci heaps for retrieval of maxima at lines 3 and 5 of FUZZY-CHANGE-SET would require $O(|T|)$ running time which would not alter the complexity of CHANGE-PROBABILITY should this be added as a final step.) The storage of $\mu(t)$ at lines 4 and 6 could be performed as an unsorted set, resulting in a constant running time if such storage would suffice for later purposes. In fact, there is no reason to keep μ in an order different than T itself, so performing the for-loop in the order in which T is stored permits us to avoid needing to sort μ, and lines 4 and 6 would remain with a constant running time. The total running time for FUZZY-CHANGE-SET would thus be in $O(|T|^2)$ for the unordered set implementation, or $O(|T|)$ for the Fibonacci heap implementation.

4.2 Estimation of Technical Risk

If we can assume that there exists a cost-of-change function $\kappa : T \mapsto \Re^+$ that is independent of paths through the risk graph, and given the fuzzy set (Δ, μ), we can compute the total technical risk τ of performing a change Δ_0 as:

$$\tau|_{\Delta_0} = \sum_{t \in T} \kappa(t) \cdot \mu(t)|_{\Delta_0}. \quad (9)$$

Ignoring the cost of calculating κ, Equation 9 consists of a simple sum of products. The functions μ and κ can each be recorded in a set sorted in the same order as T, thus yielding $O(1)$ lookup times. The total running time to compute Equation 9 will thus be in $O(|T|)$. As a starting point, we have used the simplistic notion that $\kappa \equiv 1$; we consider more realistic options in Section 6.

5. PRELIMINARY EVALUATION

As a preliminary step in evaluating the efficacy of the approach, a study was undertaken by our industrial partners on the use of the tool to plan change tasks on their codebase. This study is necessarily informal at this stage, as (un)usability issues inherent with a prototype can easily mask effects of (un)usefulness. For the sake of better understanding how our partners would approach the use of the TRE tool, they were given the freedom to conduct the study in a manner that made sense to them.

The tool was assessed by a team of four individuals: one in a technical decision making role with little current knowledge of the fine-grained code structure; one in a technical lead role involving small-scale design and implementation decisions; and two key front-line developers. This team considered a variety of tasks: three code optimizations, three bug fixes, and three changed requirements; specific details of the tasks are hidden to protect intellectual property of our industrial partners. For each task, the team was first asked to give an estimate of the risk involved, in terms of likely number of files that would need modification. They then used the TRE tool to estimate the risk. And finally, the actual number of files that were modified were compared against the estimates. The team was given a written explanation of the operation of the TRE tool, and encouraged to discuss it as a group.

A serious issue was raised by the team: the granularity at which the tool expects its input is too fine-grained for individuals with technical decision making roles but who do not develop code on a daily basis. People in such roles understand the software at an abstract level, often architectural. But such knowledge is difficult to manually translate into a selection of a set of files. Without the ability to formulate a very coarse model, downwards communication within an organization will not be facilitated by the tool. Fortunately, improving this situation can be achieved through better user interface and workflow design that we discuss further in Section 6.

The team summarized their view of the TRE tool thus: "The tool seems to be most useful with certain types of changes. For bug fixes and requirement changes the results using the tool seem reasonably close. However for optimizations it was not very useful and seems to over-estimate the changes required."

The data that they collected and reported is summarized in Table 1. When asked for more detail regarding what these numbers mean, they explained the process that they used: the team-estimated risk represents how many files they considered likely to change, while the tool-estimated risk were how many files had a 50% or greater probability of changing.

In two of the three optimization tasks, the actual change corresponded to the range of technical risk reported; in the third case, our interpretation is that the current structural dependency model did not account for a propagation either due to a subtle relationship (such as those that Ying and colleagues [35] consider) or due to one that a better model could account for, e.g., the need to propagate an interface change through the type hierarchy. For bug fix task 1, the

Task	Team-est. risk	Tool-est. risk	Actual change
Optimization 1	5–10	10–17	10
Optimization 2	2–4	2–13	5
Optimization 3	3–5	1	4
Bug fix 1	1–2	1–2	2
Bug fix 2	11	11	14
Bug fix 3	3	1	3
Changed requirements 1	1–2	11–31	2
Changed requirements 2	1–2	1	3
Changed requirements 3	1–2	5–9	2

Table 1: Data reported by the industrial team for their change tasks.

tool appears to have performed well; however, for bug fix tasks 2 and 3, the tool reportedly underestimated the risk. The report for bug fix task 2 may be a data entry error, since the tool in its present form is unlikely not to report a range of risks. For changed requirements tasks 1 and 3, the tool seems to have overestimated the risk; task 2 seems to be suffering from the same structural dependency model issues discussed above.

Despite what seems to objectively be slightly worse performance for the bug fix tasks and much worse for the changed requirements tasks, the team considered the tool more useful for the bug fix tasks and the changed requirements tasks than for the optimization tasks. One conjecture is that the summary data does not fully represent the reality of the situation, e.g., perhaps the correlation of the tool-estimated risks and the actual changes is coincidence and the individual files reported as risky do not correspond to the ones that actually require change. This conjecture must be confirmed or refuted in a future, more controlled study.

The team gravitated towards their natural tendency to threshold the information and treat even a 50–50 chance of change as a prediction of a definite file change and to disregard any lower probabilities; in some circumstances, this will not provide an accurate interpretation of the estimated risk. Also, whenever the change history contains no data, we are able to provide no real information on the probability of change propagation (i.e., 0.5 ± 0.5). This seems to have been an especially troublesome artifact, but without data, no analysis of the change history can be performed.

Ultimately, usability issues did limit the evaluation of the usefulness of the approach. Nevertheless, at least three lessons can be taken from this exercise: (1) the detailed risk data needs to be reported in a fashion that takes people's tendency to threshold into better account, and that deals with uncertainty more clearly; (2) a more iterative approach to building these models and communicating their contents upwards and downwards through an organization is needed, where those with less detailed knowledge can view the data at a more abstract level; and (3) a more comprehensive structural dependency model is needed.

6. DISCUSSION AND FUTURE WORK

As a result of our preliminary evaluation and of our original plans, a number of extensions and issues remained to be considered and possibly realized in our tool implementation. We summarize these in the following subsections: Section 6.1 considers the difficulties inherent in evaluating the probabilistic results of the theoretical model, and the need for further *in situ* study; Section 6.2 considers the potential sources of error in the data extraction and modelling which might be better addressed; and issues surrounding workflow and organizational communication are discussed in Section 6.3.

6.1 Evaluation of model and tool

Currently, the model and tool we have described make a number of assumptions that may be violated in the real world. Our preliminary evaluation suggests that issues that we had considered to be minor points of usability would actually be too severe for the tool to be adopted in the manner intended. An iterative approach to *in situ* evaluation will continue to be necessary to ensure that additional such points do not intrude in future.

Aside from such practical issues, the question as to whether the model that underlies the tool produces results that are accurate remains unanswered. The TRE tool reports the technical risk of a proposed change task as the combination of the probability that a change will propagate to a given type and the cost of changing that type should the propagation occur. Its probabilistic predictions are valid only for a single trial; while one might expect that a set of predictions will become very inaccurate after only a few trials, testing convergence to a probability will require a very large number of trials. Similarly, strict correlation between a thresholded probabilistic prediction (e.g., a prediction above 50% means that the file is predicted to definitely change) and actual outcomes would be incorrect: improbable events will occasionally happen and very likely events occasionally will not. Instead, we have determined a means for correlating the actual profile of a large number of individual probabilistic predictions and single trials with a theoretical profile. In this manner, we hope to vary some of the assumptions of the TRE model, such as the form of the structural dependency model, and compare the results of these variations quantitatively. We are in the process of implementing the infrastructure to collect the data for such evaluations. Details of the theory behind it are left until results from its application can also be reported.

6.2 Sources of error

A number of issues in the way we collect and model data are potential sources of error within our analyses.

An often reported issue is that of the presence of transformations within the codebase, ranging from the renaming of types to large-scale refactorings. We are in the process of applying the technique of origin analysis [10] as a means to better combine data arising before and after such transformations. A finer-grained representation of structural dependencies combined with origin analysis is likely to improve the accuracy of our predictions.

Such transformations are but one impediment to the larger issue of inferring causality that our model ultimately involves. While one can determine atomic change sets either because the repository system supports them explicitly or they can be inferred, approaches (including ours) that depend on atomic change sets as the basis for causality inference are at the mercy of the repository commit style applied by the developers of a given project. For example, are entire feature sets committed at once or are individual files checked in on a rather ad hoc basis? Instead, a stronger model for recovering causality is needed. Utilizing data from change request repositories to bridge the gaps between atomic change sets is one possibility (e.g., [8, 31]).

Of particular interest to our industrial partners is the ability to cope with software that involves multiple languages. Our current approach for extracting structural dependencies would need to support specific combinations of languages to understand how they interact. If the number of languages to be simultaneously supported is small and invariant, the brute force method of providing syntactic and semantic analysis support is likely practicable. However, in situations where configuration details effectively result in a proliferation of small, special purpose languages, the cost of providing such support becomes prohibitive. Lightweight approaches involv-

ing lexical analysis [20] or island grammars [18] must be considered as a more practicable solution.

Our current assumption that the cost of changing any given file is constant (and "1") regardless of any other factors is clearly unrealistic, but serves as a simple starting point. Utilizing data on the size of changes or the error rates connected with previous changes (e.g., [17, 31, 22]) are more sophisticated approaches that we will investigate in future.

6.3 Workflow and organizational issues

A number of issues involving workflow and usability of the tool have arisen, some of which are straightforward to address while others are not.

It is clear that the input required by the tool at present is too fine-grained to support its use by people with only abstract knowledge of the system under study. On the other hand, an even finer-grained input could be useful where more specific information is known. The theoretical model that we have described in this work does not require that seed points be specified at the granularity of types. Coarser-grained input could be supported by providing a tree-based view that collapses types into packages, for example, or that otherwise provides some form of architectural clustering (e.g., reflexion models [21]). Likewise, the change history data could be analyzed at a finer-granularity to allow input at the method- and field-level when desired.

The issue then becomes one of how to interpret seed markings at coarse granularities in terms of the underlying fine-granularity model. Two options would seem worth pursuing. Marking a coarse-grained item (such as a package) could be treated as equivalent to marking each of its individual constituent types (or methods). This could interact with organizational workflow in such a way that the technical decision maker's initial estimates of risk would be consistently higher than those of the front-line developers. What the social and organizational implications of such a phenomenon might be remains unclear. Alternatively, a fraction of the probability could be assigned to each of the underlying fine-grained items; note that Equation 7 remains valid for probabilities of seed points other than 1.

Other issues of practical importance include the ability to collect the data incrementally as changes happen over time, better user interface controls to adjust the way the results are displayed (e.g., sliders for adjusting thresholds of interest and of pessimism), basic search functionality in the user interface, and the ability to mark nodes with probabilities other than 1 (e.g., to specify "this is guaranteed not to change"). Differentiating the kind of change would also allow the model to propagate different kinds of change in different fashion. The details of such categorizations, the effect on the theoretical model, and the practicality of asking for additional input need to be considered further.

7. RELATED WORK

Various previous work has considered the meaning of "technical risk" and how it should be managed. Technical risk has been seen as a serious factor in industrial development for decades (e.g., [7]), where improved tools and process were often seen as the key means to mitigate it. Initially, technical risk was often equated with the risk that the proposed technology to be used in a development effort would not actually support the application, and so that development effort would fail (e.g., [2, 4]). Later, technical risk was generalized to aspects of managing technical personnel and technical projects (e.g., [30]), and a continued emphasis on process quality as the means of mitigation. This move towards separation of managerial decisions and technical decisions has been seen as emphasizing the organizational gulf between decision makers and technical personnel [4]. More recent work attempts to provide questionnaires to guide assessment of risk (e.g., [14]) or to leverage and combine the opinions of experts [27, 28]. The TRE tool is seen as providing input to the release planning process as envisioned by Saliu and Ruhe [28].

Various authors have considered which risk factors are most serious in software development, often based on surveys of project managers' opinions [26, 32]. Others have attempted more objective, quantitative approaches [23]. Still others provide survey based approaches to evaluate the risk in a given project

A large body of work emphasizes the risk of introducing defects into software as it is modified. Belady and Lehman provide a very early attempt at doing this quantitatively [1]. Mockus and Weiss provide a probabilistic model of software failures due to a variety of factors including developer experience and change propagation [17]; Schneidewind [29] provides an alternative model. A large body of work by Zeller and colleagues focusses on failure prediction based on historical data (e.g., [31, 22]).

Dependence analysis worked from a strong theoretic basis [11], and has long been seen as important in software modification [15]. Formal undecidability lead to approximate change impact analysis techniques [19, 12, 3]. However, all these techniques require detailed implementations as input. In contrast, Turver and colleagues define a technique for early change impact analysis [34]; unfortunately, it assumes complete and accurate documentation that is rarely available in the development setting that we consider. Moonen has proposed a lightweight impact analysis technique based on the creation of island grammars to describe the syntactic cues that an analyst seeks [18]; however, this approach remains too detailed for the needs of early decision support.

Many approaches try to evaluate the quality of designs based on quantitative measures. Tsantalis and colleagues most recently consider how such quantitative approaches can predict change [33].

Most significant for this workshop are the large number of approaches that use historical data to guide change tasks. Both Ying and colleagues [35] and Zimmermann and colleagues [37] consider how frequent patterns in past change propagation can be used to guide developers in propagating changes that they might otherwise miss. They use thresholding to filter suggestions that are unlikely to be important; in contrast, we must consider the presence of low probability/high cost events in evaluating risk. The approach we have presented in this paper is largely complementary to these, lying at the opposite end of the spectrum between coarse-grained planning and fine-grained implementation.

8. CONCLUSION

We have described a theoretical model for technical risk estimation and its concrete realization as the TRE tool. TRE has been designed to help organizations make more informed decisions about the risks associated with modifying software elements within their changing systems. By providing an abstract view about the potential costs of a particular change to managers, and more concrete data to developers, the tool facilitates communication between these two groups. Through supporting the decision making process, TRE allows organizations to ground their development plans both in the structural nature of their source code and the past development history that the system has undergone.

Although preliminary evaluation has shown a need to improve the tool in specific ways to support this vertical communication between management and developers, we believe that most of these issues can be addressed by small improvements in workflow-support and usability.

Ultimately we see the TRE tool as being complementary to many of the approaches that have come out of the repository mining community in recent years. Our work extends these approaches to provide an objective foundation for making decisions at various levels of industrial organizations about the technical risk of software modifications.

9. ACKNOWLEDGMENTS

The authors wish to thank Stefania Bertazzon for comments on a draft of this paper. This work was supported in part by the National Science and Engineering Research Council and in part by Chartwell Technology Inc.

10. REFERENCES

[1] L. A. Belady and M. M. Lehman. A model of large program development. *IBM Systems J.*, 15(3):225–252, 1976.

[2] B. I. Blum. Three paradigms for developing information systems. In *Proc. Int'l Conf. Softw. Eng.*, pages 534–543, 1984.

[3] S. Bohner. Software change impacts: An evolving perspective. In *Proc. Int'l Conf. Softw. Maintenance*, pages 263–271, 2002.

[4] C. Chittister and Y. Y. Haimes. Assessment and management of software technical risk. *IEEE Trans. Systems, Man and Cybernetics*, 24(2):187–202, 1994.

[5] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. McGraw-Hill, 1992.

[6] E. W. Dijkstra. A note on two problems in connection with graphs. *Numerische Mathematik*, 1:169–271, 1959.

[7] H. Fischer. Computer system simulation of an on-line interactive command and control system. In *Proc. Winter Simulation Conf.*, pages 333–340, 1971.

[8] M. Fischer, M. Pinzger, and H. Gall. Populating a release history database from version control and bug tracking systems. In *Proc. Int'l Conf. Softw. Maintenance*, pages 23–32, 2003.

[9] R. W. Floyd. Algorithm 97 (shortest path). *Commun. ACM*, 5(6):345, 1962.

[10] M. W. Godfrey and L. Zou. Using origin analysis to detect merging and splitting of source code entities. *IEEE Trans. Softw. Eng.*, 31(2):166–181, 2005.

[11] S. Horwitz, T. Reps, and D. Binkley. Interprocedural slicing using dependence graphs. *ACM Trans. Program. Lang. Sys.*, 12(1):26–60, Jan. 1990.

[12] M. Hutchins and K. Gallagher. Improving visual impact analysis. In *Proc. Int'l Conf. Softw. Maintenance*, pages 294–303, 1998.

[13] D. B. Johnson. Efficient algorithms for shortest paths in sparse networks. *J. ACM*, 24(1):1–13, 1977.

[14] J. Kontio, G. Getto, and D. Landes. Experiences in improving risk management processes using the concepts of the Riskit method. In *Proc. ACM SIGSOFT Int'l Symp. Foundations Softw. Eng.*, pages 163–174, 1998.

[15] J. P. Loyall and S. A. Mathisen. Using dependence analysis to support the software maintenance process. In *Proc. Conf. Softw. Maintenance*, pages 282–291, 1993.

[16] A. Mockus, R. T. Fielding, and J. Herbsleb. Two case studies of open source software development: Apache and Mozilla. *ACM Trans. Softw. Eng. Method.*, 11(3):1–38, 2002.

[17] A. Mockus and D. M. Weiss. Predicting risk of software changes. *Bell Labs Technical J.*, 5(2):169–180, 2000.

[18] L. Moonen. Lightweight impact analysis using island grammars. In *Proc. Int'l Wkshp. Program Comprehension*, pages 219–228, 2002.

[19] M. Moriconi and T. C. Winkler. Approximate reasoning about the semantic effects of program changes. *IEEE Trans. Softw. Eng.*, 16(9):980–992, 1990.

[20] G. C. Murphy and D. Notkin. Lightweight lexical source model extraction. *ACM Trans. Softw. Eng. Method.*, 5(3):262–292, 1996.

[21] G. C. Murphy, D. Notkin, and K. Sullivan. Software reflexion models: Bridging the gap between source and high-level models. In *Proc. ACM SIGSOFT Symp. Foundations Softw. Eng.*, pages 18–28, 1995.

[22] N. Nagappan, T. Ball, and A. Zeller. Mining metrics to predict component failures. In *Proc. Int'l Conf. Softw. Eng.*, 2006. To appear.

[23] D. E. Neumann. An enhanced neural network technique for software risk analysis. *IEEE Trans. Softw. Eng.*, 28(9):904–912, 2002.

[24] A. Podgurski and L. Clarke. A formal model of program dependences and its implications for software testing, debugging, and maintenance. *IEEE Trans. Softw. Eng.*, 16(9):965–979, 1990.

[25] K. S. Rajeswari and R. N. Anantharaman. Development of an instrument to measure stress among software professionals: Factor analytic study. In *Proc. SIGMIS Conf. Computer Personnel Research*, pages 34–43, 2003.

[26] J. Ropponen and K. Lyytinen. Components of software development risk: How to address them? A project manager survey. *IEEE Trans. Softw. Eng.*, 26(2):98–112, 2000.

[27] G. Ruhe and D. Greer. Quantitative studies in software release planning under risk and resource constraints. In *Proc. IEEE Int'l Symp. Empirical Softw. Eng.*, pages 1–10, 2003.

[28] O. Saliu and G. Ruhe. Software release planning for evolving systems. *Innovations in Systems and Softw. Eng.*, 1(2), 2005. To appear.

[29] N. F. Schneidewind. Predicting risk as a function of risk factors. *Innovations in Systems and Softw. Eng.*, 1(1):63–70, 2005.

[30] S. A. Sherer. The three dimensions of software risk: Technical, organizational, and environmental. In *Proc. Hawaii Int'l Conf. Systems Science*, pages 369–378, 1995.

[31] J. Śliwerski, T. Zimmermann, and A. Zeller. When do changes induce fixes? In *Proc. Int'l Wkshp. Mining Software Repositories*, pages 24–28, 2005.

[32] A. Tiwana and M. Keil. The one-minute risk assessment tool. *Commun. ACM*, 47(11):73–77, 2004.

[33] N. Tsantalis, A. Chatzigeorgiou, and G. Stephanides. Predicting the probability of change in object-oriented systems. *IEEE Trans. Softw. Eng.*, 31(7):601–614, 2005.

[34] R. J. Turver and M. Munro. An early impact analysis technique for software maintenance. *J. Softw. Maintenance: Res. and Pract.*, 6:35–52, 1994.

[35] A. T. T. Ying, G. C. Murphy, R. Ng, and M. C. Chu-Carroll. Predicting source code changes by mining change history. *IEEE Trans. Softw. Eng.*, 30(9):574–586, 2004.

[36] L. A. Zadeh. Fuzzy sets. *Information and Control*, 8(3):338–353, 1965.

[37] T. Zimmermann, P. Weißgerber, S. Diehl, and A. Zeller. Mining version histories to guide software changes. *IEEE Trans. Softw. Eng.*, 31(6):429–445, 2005.