JOURNAL OF SOFTWARE: EVOLUTION AND PROCESS
J. Softw. Evol. and Proc. 2015; **00**:1–29
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/smr

Error leakage and wasted time: sensitivity and effort analysis of a requirements consistency checking process

Wenbin Li[1], Jane Huffman Hayes[1]* Giulio Antoniol[2], Yann-Gaël Guéhéneuc[2], Bram Adams[2]

[1]*University of Kentucky, USA*
[2]*Ecole Polytechnique de Montreal, Canada*

SUMMARY

Several techniques are used by requirements engineering practitioners to address difficult problems such as specifying precise requirements while using inherently ambiguous natural language text and ensuring the consistency of requirements. Often, these problems are addressed by building processes/tools that combine multiple techniques where the output from one technique becomes the input to the next. While powerful, these techniques are not without problems. Inherent errors in each technique may leak into the subsequent step of the process. We model and study one such process, for checking the consistency of temporal requirements, and assess error leakage and wasted time. We perform an analysis of the input factors of our model to determine the effect that sources of uncertainty may have on the final accuracy of the consistency checking process. Convinced that error leakage exists and negatively impacts the results of the overall consistency checking process, we perform a second simulation to assess its impact on the analysts efforts to check requirements consistency. We show that analysts effort vary depending on the precision and recall of the sub-processes and that the number and capability of analysts affect their effort. We share insights gained and discuss applicability to other processes built of piped techniques. Copyright © 2015 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: Sensitivity analysis; error propagation; requirements engineering; consistency checking; information retrieval; natural language processing; classification; semantic role labeling; error leakage; process model; machine learning; genetic algorithms; search-based software engineering; work flow process

1. INTRODUCTION

Challenges loom large as we endeavour to develop software systems. Many of these challenges are related to requirements engineering (RE) and its subareas of process, elicitation, specification, analysis, and validation [1]. In eliciting requirements, we face several obstacles as we attempt to ensure that all stakeholders are represented. In striving for thoroughness and exactness, we contend with customers and end users who do not understand formal languages or specifications. We also struggle to specify requirements that are complete and consistent because we deal with the inherent ambiguity of natural language text. Biggerstaff et al. labeled this the concept assignment problem [2]. As researchers, we strive to understand these and other issues and to develop approaches alleviating these challenges for practitioners.

We can address requirements challenges by various techniques, such as information retrieval, natural language processing, classification, clustering, to name a few. Complex requirements

*Correspondence to: Jane Huffman Hayes, University of Kentucky. E-mail: hayes@cs.uky.edu problems may be addressed by a collection of such techniques. For example, classifying a requirement as functional or non-functional requires this sequence of steps: (1) to perform pre-processing on the text of all of the requirement elements (remove stop words, spell out acronyms, etc.); (2) to tag each token in each requirement element with its part of speech; (3) to develop a training set to be used with a given dataset; and, (4) to use a classification or clustering technique to determine the labels for each requirement element (functional or non-functional requirement).

The aforementioned techniques (as any others) are powerful but not perfect. When researchers or practitioners build complex processes or tools from these non-perfect techniques, errors may leak into subsequent process steps (analogous to defects leaking into subsequent software development phases). For example, it is well known that typical information retrieval techniques used in tracing requirements achieve high recall but only low to moderate precision. Errors due to the imperfect and imbalanced precision and recall may seem insignificant, particularly if a process uses only one technique. But more complex requirements engineering processes require multiple techniques and, as in the functional/non-functional classification example given above, it is often the case that the output of one technique is the input to another technique, e.g., the output of parts of speech tagging becomes the input to the classification technique. Errors that leak from one technique to the next will impact the accuracy of the final result and will result in wasted time due to false positives introduced into the process.

Errors in the techniques have been the subject of much work but researchers focused on the quality of individual techniques, such as improving the recall and precision of information retrieval techniques used for feature location. To the best of our knowledge, they did not study sequences of these techniques and error leakage. In this paper, we only consider fully-automated processes, in which analysts do not intervene; future work includes modelling the impact of the analyst-in-the-loop [3].

Practitioners often have a number of techniques from which to choose at the various steps of a requirements engineering process. For example, in the functional/non-functional requirement categorization example, they can select one among a multitude of clustering techniques. It seems intuitive that they should select the technique with the highest quality, i.e., the lowest error rate or error leakage rate. However, quality has different definitions and values depending on the technique. Also, quality is influenced by the characteristics of the artifacts and/or the domain of the artifacts and, consequently, quality has a level of uncertainty. For example, the vector space model (VSM) with term frequency-inverse document frequency weighting (tf-idf) is used often for requirements tracing. In a comparison of studies on requirement tracing, we observed that VSM (not enhanced by thesauri or similar) achieved recall of 83% with precision of 54% [4]. In a different setting, using different datasets written using different vernacular, recall was merely 25.4% with precision of only 11.4% [5, 6]. This uncertainty must be taken into account. Further, it is well known that the outputs of such a requirements engineering process will not be accepted at face value [6, 7, 8] . Rather, a human analyst will vet the results. These vetted results will constitute the final output that will be used to support subsequent activities in the software development lifecycle.

For many years, researchers assumed that a process resulting in high accuracy output would lead to a high quality final result (after the human analyst vetted). In fact, the study of methods assumed a perfect analyst when modeling the human in the loop (analyst would always make correct decisions when evaluating results from an automated tool) [6]. About five years ago, this was shown to be an incorrect assumption [7, 8, 9]. For trace recovery, such as between a requirements specification and a design document, studies showed that human analysts vetting the output of an automated process tended to make the results worse than what the process output (that is, they lowered the recall and precision of the results given to them for vetting) [7]. Interestingly enough, human analysts tended to make the best decisions and improve results when the process output was not so accurate. In fact, analysts who were given output of 100% recall, 100% precision actually returned final results that were much worse than analysts given much lower quality outputs.

One study found a "sweet spot" for the quality of a process output in terms of recall and precision that lead to analysts making the most improvement to the results and resulting in the best final output. The sweet spot was in the range of 25-62% recall, 25-50% precision [7]. Work also indicated that human analysts are much better at vetting results given to them than at going to look for elements that were not retrieved by an automated process. Further work also indicated that when vetting results, analysts are correct roughly 75% of the time [10].

Thus, it is not sufficient for a practitioner to simply find the "best" techniques (in terms of recall and precision) to assemble a process that will result in the best final output for use in subsequent activities. A practitioner can be confronted with many degrees of freedom in putting together a process: what techniques shall be selected (based on recall, precision, accuracy), in what order shall these techniques be run, how shall the process be "configured" (here we limit ourselves to the notion of optional versus non optional technique usage-configuration in the broader sense is not within the scope of this paper). While it may seem intuitive that higher quality techniques will lead to the best processes, validation is still required. Pioneers in empirical software engineering such as Vic Basili and Mary Shaw have emphasized the need for empirical validation [11, 12]. We argue that search-based software engineering can support the decision process by guiding selection of techniques and selection or ordering of techniques to minimize error leakage and wasted time. This paper, the first to study this problem, is a step toward that goal.

Using the consistency checking of temporal requirements with Temporal Action Language (TeAL) [13] as our running example, we formulate the problem as a fully automated process - a piped sequence of techniques with the output of each serving as the input to the next. We perform exploratory sensitivity analyses, using published data on accuracy of techniques, simulated data, and data collected by applying TeAL on a real world system. As expected, we found that retrieval rates and precision values of the techniques influence the quality of the overall process.

Then, following the sensitivity analysis, we study the impact of error leakage and wasted time on analysts efforts when they check the consistency of temporal requirements at each step of a checking process. Reusing our previous example, we model an objective function using three terms: a term modelling the analysts work times, a term modelling the overall time that it takes analysts to vet all requirements at each step of the process, and a term accounting for analysts wasted time. Using a genetic algorithm, we compute this objective function in four scenarios with different precision and recall values of each sub-process, as well as perfect and imperfect analysts, and show that error leakage leads to wasted time but that the overall time is decreased. We also show that error leakage causes the decrease of the overall time at the cost of errors introduced into the process.

This paper is an extension of the paper "Error Leakage and Wasted Time: Sensitivity Analysis of a Requirements Consistency Checking Process" [14]. The major extension contains two parts. One is a sensitivity analysis of the impact of varying sub-process orders and optionality in a process of consistency checking of temporal requirements. The other is a study that uses a genetic algorithm to analyze the impact of error leakage.

The paper is organized as follows. We present related work in Section 2 before modeling error leakage and wasted time and collecting data about the accuracy of techniques through a literature review in Section 3. We present the sensitivity analysis in Section 4. Section 5 presents the analysts effort model as a genetic algorithm as well as analysis and results. We discuss the threats to the validity of our results in Section 6. Finally, we conclude and present future work in Section 7.

2. RELATED WORK

This paper is an extension of the previous paper "Error Leakage and Wasted Time: Sensitivity Analysis of a Requirements Consistency Checking Process" [14], which studies error leakage in a sequence of analysis steps and its impact on accuracy and wasted effort. This section presents prior related work and shows that, although researchers studied bias in various kinds of data and processes in software engineering, they did not study error leakage in this way.

2.1. *Sensitivity and Bias*

Previous work on software engineering processes typically focused on the quality of individual techniques, not of the full process. For example, Le et al. [15] built a classifier model to predict whether or not a given fault localization approach would be successful for a given fault.

The classifier model does not provide feedback about accuracy or wasted effort, only a yes/no classification. More generally, sensitivity analysis on the impact of certain input variables on an approach or algorithm is commonplace in software engineering research (e.g., [16, 17]).

The closest work to ours is the work on error leakage due to bias in software engineering process **data** (not the actual **techniques**), i.e., the imperfect sampling or composition of a data set can have negative consequences on the subsequent analyses. Many kinds of bias have been identified, together with their impact on the successive phases of analysis, such as defect or effort prediction. Antoniol et al. [18, 19] analyzed tagging bias caused by interpreting all reports in a project's issue tracking system as bug reports, whereas analysts often use issue tracking systems to track other kinds of issues, such as tasks, decisions, and enhancements. Hence, the number of bug reports is overestimated and can impact the accuracy of defect prediction models.

Linkage bias [20, 21] corresponds to the observation that low severity bugs have a higher chance of being tagged with the source code changes in the version control system that fixed them and that experienced analysts are more likely to add such tags for the bugs that they resolve than less experienced analysts. Again, any subsequent analysis step will be impacted by the biased data set. Many other kinds of bias exist, such as code changes that bundle multiple bug fixes and features into one code change [22], limitations and risks in development data recoverable from repositories using Git [23] and GitHub [24], and the size of (a possibly biased) data set [25]. While our paper also focuses on bias, the bias that we analyze applies to the **techniques** that are being applied (and their order) and, hence, is complementary to bias in the **data**.

Closer to the bias that we analyze is the work of Dit et al. [26] and Binkley et al. [27] on the impact of various kinds of preprocessing operations of textual data on the effectiveness of feature location and traceability recovery from source code. Both papers compared the impact on feature location algorithms of using different approaches for splitting identifiers. They found that only for some source code projects feature location approaches based on information retrieval techniques do benefit from cleanly split source code identifiers.

Binkley et al. [28] and Haiduc et al. [29] studied how different ways of pre-processing and modeling queries for information retrieval impact the performance of traceability techniques when applied on issue reports and source code. Binkley et al. [28] found large differences in performance depending on (1) the parts of the issue report included in the analysis, (2) the use of different identifier splitting techniques, (3) inclusion or not of synonyms, and (4) choice of likelihood-based information retrieval models instead of LSI-based models. They reported that stemming of words did not have any impact. Haiduc et al. [29] proposed a model to recommend the best query rewriting approaches for a given information retrieval query. Instead of the impact of bias on information retrieval queries or concept location, we focus on consistency checking of temporal requirements.

2.2. *Search-based Software Engineering*

Search-based software engineering (SBSE) is a discipline that uses search-based optimization techniques to solve software engineering-related problems [30]. Any problem that can be formulated in terms of minimizing a cost function can be attempted using SBSE. Once a cost function is established (and a suitable representation of the problem is found), traditional search techniques, like linear programming, or meta-heuristic approaches, like genetic algorithms, can be applied. For example, Baker et al. [31] studied the selection of components to be released in a next version as a search-based problem, with net revenue of the selected components as cost.

Genetic algorithms (GAs) are one of the most popular search algorithms for SBSE [26], with applications ranging from test input data generation [32], project planning [33], refactoring [34] to web service composition [35], just to name a few. However, to the best of our knowledge, GAs have not been used before in SBSE to perform a sensitivity analysis. Among the various works using GAs, more similarity can be found with the paper of Di Penta et al. [33] in that we also seek for a robust solution. This work and the work by Baker et al. [31] inspired us to use search-based techniques to explore the optimal configuration of techniques to use in a requirement traceability process, with the precision and recall of the final results as cost function. Panichella et al. [36] also presented a study which uses a genetic algorithm to explore configurations of different IR steps in typical IR processes for software engineering tasks and find an optimal configuration. This method can obtain up to 50% better precision than a default configuration, close to ideal, basically by measuring the quality of clusters of configurations (silhouette coefficient).

2.3. *Process Modeling*

Alegria et al. [37] described a tool for building software process models, Avispa, which is a graphical tool for analyzing Software Process Engineering Metamodel (SPEM) 2.0 process models [1]. The authors discussed the notion of error patterns that are commonly encountered in process modelling, such as Isolated roles (node not connected with an edge), Multiple Purpose tasks (too many output work products), and Overloaded Roles (role involved in too many tasks). However, they did not model or simulate the impact of these error patterns on the final results of the process.

Anan et al. [38] examined the number of errors by phase of the software development lifecycle, using various models to estimate errors. For example, they examined software reliability models, such as Goel-Okumoto, to examine failures during testing to estimate remaining undetected errors. They did not discuss the impact of the different techniques used during each phase on the overall errors and final results of the software development process.

Other research work modeling software engineering processes includes that of Pandey, Suman, and Ramani [39], of Krusche, Alperowitz, Bruegge, and Wagner [40], and of Sadiq, Mohd, and Hassan [41]. Although these previous works modelled various processes and provided evaluation of these processes, they did not study the impact of errors at each step of the processes and the propagation of these errors through the whole processes and their impact on the final results.

2.4. *Consistency Checking*

As the subject of this study, we chose the problem of requirement consistency checking because it is a well-known and difficult problem in requirements engineering and because proposed solutions are processes made of several piped techniques. Typically, this problem arises because inconsistencies among software requirements must be removed to ensure the quality of software projects. Consistency checking tasks are time-consuming and error-prone if performed manually. Therefore, researchers proposed alternatives to manual checking using a formal system, such as Linear Temporal Logic [42], Answer Set Programming [43, 44], and Temporal Action Logic [45]. Once researchers provided formal representations of requirements, they could use automated reasoning tools for the automated detection of inconsistencies.

Recently, a promising semi-automated approach was proposed to address consistency checking in the case of temporal requirements. The approach is based on the Temporal Action Language TeAL [13], developed by several of the co-authors of this paper, and on tools that process natural language requirements. A tracing tool, RETRO [46], is used to extract temporal requirements and some non-temporal related requirements from the requirement document. The extracted requirements are parsed and the key elements of the system that they specify are identified using the Stanford Parser [47, 48, 49]. Based on this information, an “almost TeAL” approximation of the sought TeAL representation is produced by a tool GenerateTeAL. The analyst must finally correct, complete, and verify the output of GenerateTeAL. This final task has been demonstrated to be significantly easier for analysts than that of generating the TeAL theory directly from the requirements.

In the following, we use consistency checking of requirements using the Temporal Action Language as subject of our simulation and sensitivity analysis to show that errors leak from one technique to the next and that they impact the final results of the requirement consistency checking process.

3. MODELING ERROR LEAKAGE AND WASTED TIME

We now describe our overall approach to modeling error leakage and wasted time. We then present the formulation of the problem of designing an automated sequence of techniques for consistency checking of temporal requirements.

We use as running example a consistency checking process that is composed of multiple sub-processes, each implemented by one technique (we limit our scope to single techniques and will study processes composed of composite sub-processes in future work, i.e., sub-processes using several techniques **in parallel**). Based on our industry experience and work with industrial partners, the results of such an automated process are not accepted at face value. Rather, some level of checking or review is performed by an analyst before the results are used. It is intuitive that an automated process that retrieves only relevant elements will result in the easiest review task for the analyst. On the contrary, an automated process that fails to retrieve relevant elements and only retrieves irrelevant elements will require too much work and result in frustration for the analyst. Since reality lies somewhere in between these two extremes [50], what aspects of these techniques most influence the quality of the overall process?

We seek to understand what most influences the quality of the overall consistency checking process. For this, we undertake two studies: sensitivity analysis (Sections 4), and problem analysis using a genetic algorithm (Section 5). We address the sensitivity analysis first, and follow a four-step approach: 1) We develop a model of the overall consistency checking process that computes a set of output factors from a set of input factors. We combine the output factors into one objective function for the overall consistency checking process. 2) We determine the distributions for the input factors and randomly generate data according to the distributions, as shown in Figure 1. We use three of the four sources suggested by Wagner [51]: scientific literature, expert opinion where needed, and a controlled experimental study of our Temporal Action Language (TeAL) process [13]. 3) We generate scatterplots of the input factors to determine their influence. Finally, 4) we model the process with optionality and varying order of sub-processes and perform a sensitivity analysis to study the impact of these factors.

3.1. Model of the Consistency Checking Process

For our discussion, we assume that we work with the set $R = \{r_1, \ldots, r_n\}$ of n requirements specifying a software system. We denote by T the subset of R consisting of all temporal requirements in R and assume that $T = \{t_1, \ldots, t_m\}$.

We assume that to check the consistency of the temporal requirements in R, we use a process P comprising sub-processes to be performed in a sequential order: the input to P is sent as the input to the first process, its output is sent as input to the next process, etc. We write $P =< P_1, \ldots, P_k >$ to indicate the sub-processes forming P, as well as the order in which they are performed. We further assume that each step P_i, $i = \{1, \ldots, k\}$, can be accomplished by any of the several available j_i techniques in the set $C_i =< p_1, \ldots, p_j >$. We denote as p_k the technique chosen to fulfill the sub-process k.

Each technique has a retrieval rate ar. This is the recall of the technique, where recall is defined as the percentage of relevant elements that are retrieved over all retrieved elements. There is also an error rate er associated with the technique; this represents non-retrieved or "lost" elements. These elements can also be thought of as false negatives or Type 2 errors. Together they satisfy the identity $er + ar = 1$. Thus, if an information retrieval technique, such as LDA, has recall of 0.9, its error rate is 0.1. If a noun-verb classifier has retrieval rate of 0.8, its error rate is 0.2. With ar and er, we limit our interest to lost elements, which likely will no longer be considered in the subsequent sub-processes due to the imperfect recall of upstream techniques. We denote error leakage as el; for a given sub-process el = $number\ of\ relevant\ elements \times er$.

Each technique has a wasted time cost rate cr due to irrelevant elements retrieved that must be processed in this and subsequent sub-processes. These elements can also be thought of as false positives or Type 1 errors. For techniques that are retrieving elements from a collection, the precision of the technique, or pr, is defined as the percentage of retrieved elements that are relevant over the retrieved elements. For classification techniques, pr measures how well elements are labelled or categorized. In either case, pr can be used to find cr. Together they satisfy the identity $cr + pr = 1$. Thus, if an information retrieval technique has precision of 0.6, cr is 0.4. We are interested in the false positive elements because these connote possible wasted time/extra effort in subsequent sub-processes. We denote wasted time as wt; for a given sub-process, wt = $number\ of\ relevant\ elements \times cr$.

The objective function *process-cost* that we will use in performing the sensitivity analysis combines these error leakages and wasted times. We quantify the extra work for an analyst examining the output of a given process as the time that must be invested to find lost elements el plus time that must be invested to examine elements that should not have been retrieved wt:

$$process\text{-}cost \ = \ el(p_k) \ + \ wt(p_k).$$

Running Example: In our running example, we examine the four sub-processes of consistency checking of temporal requirements, i.e., $i = 4$. Let us assume a document R describing 500 requirements, $R = \{r_1, \ldots, r_{500}\}$, 100 of which, denoted $t_1, \ldots, t_{100}$, are temporal, i.e., $T = \{t_1, \ldots, t_{100}\}$. We apply a four-step consistency checking process P to analyze R, where the process $P =< P_1, P_2, P_3, P_4 >$. Let us assume that each step Pi, $i = \{1, 2, 3, 4\}$, can be accomplished by a technique p_i with accuracy rate ar_i, cost cr_i, and precision pr_i. We model and study the quality of the output of P_4, that is, the output of the entire process P. This quality is expressed by our objective function, which depends on the parameters of the techniques selected for each step.

sub-process P_1 determines the set T consisting of temporal requirements in R. It maps any set of requirements R to its subset $o_1 = P_1(R)$. Ideally, o_1 will consist of all temporal requirements in R. However, in performing this mapping, some temporal requirements may not be retrieved from R, and we quantify this error with the error rate er_1, while some non-temporal requirements will be retrieved, which we quantified by the precision pr_1. In our notation and because $ar = 1 - er$ and $cr = 1 - pr$ by definitions, we have:

$$el(p1) = n \times er_1$$
$$wt(p1) = n \times cr_1$$

The goal of the second sub-process P_2 is to identify in R non-temporal requirements that are related to the temporal ones and so might affect the consistency of the temporal requirements. That sub-process takes as input the original set R of requirements less the set o_1 of requirements identified as temporal in Step 1. The set o_1 is only an approximation of T. In this example, sub-process P_2 outputs the set o_2.

In performing this transformation, some related requirements may not be retrieved due to the error er_2 of the sub-process P_2, and some unrelated ones may be depending on the precision pr_2. Taking these rates into account and applying them to o_1 and R, we have:

$$el(p_2) = (cardinality(R) - cardinality(o_1)) \times er_2$$
$$wt(p_2) = (cardinality(R) - cardinality(o_1)) \times cr_2$$

sub-process P_3 is applied to each of the requirements in the sets o_1 and o_2 to tag their parts of speech. The output consists of a set o_3 of tags. In performing this transformation, we use only the retrieval rate ar_3 for this sub-process and set pr_3 to 0 because one and only one tag is assigned per part of speech by definition of the requirement consistency checking problem. We thus assume that all tags in a requirement are correct or all are incorrect. This assumption is based on the internal cohesion of requirements: if one tag is correct, than majority (if not all) of the tags will be correct. So, given a collection of 10 requirements that are tagged with 0.8 ar_3, we assume that 8 requirements are correctly tagged and 2 are not. We thus have:

$$el(p_3) = (cardinality(o_2) + (cardinality(R) - cardinality(o_2))) \times er_3$$
$$wt(p_3) = wt(p_2) + pr_3$$

Finally, sub-process P_4 is applied to the tagged temporal and related requirements retrieved from P_3, o_3, to identify the semantic roles in each requirement (such as action, agent). P_4 transforms the output of P_3 to a set of roles for each requirement tagged in P_3. In performing this transformation, some roles may not be retrieved due to er_4, some may be retrieved that should not be, because of

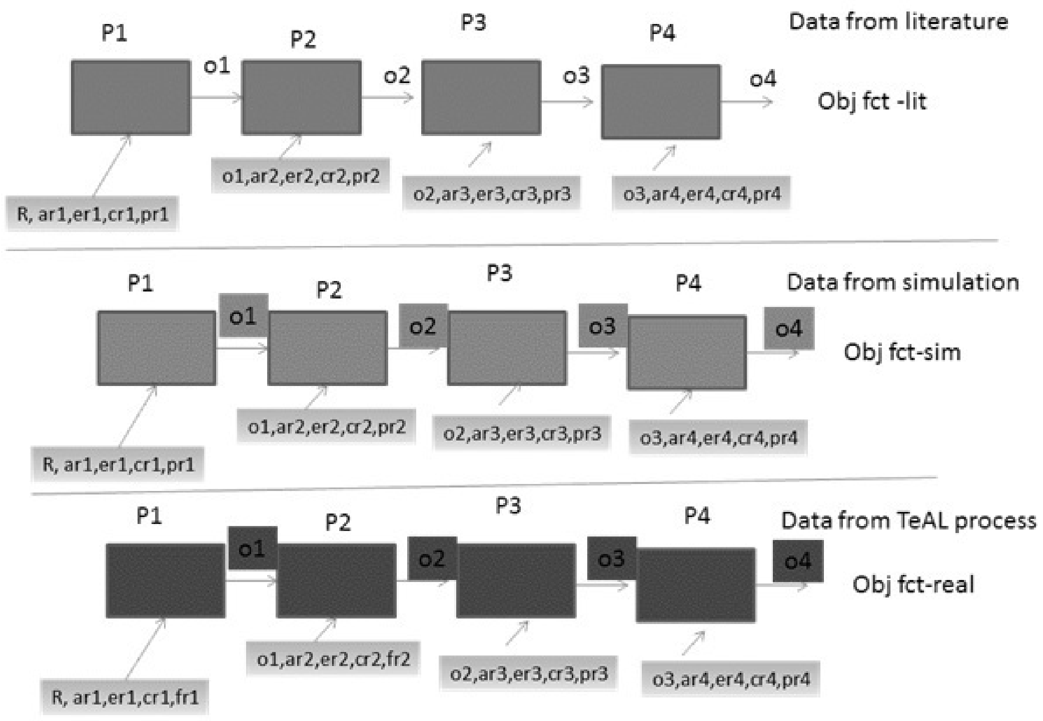

Figure 1. Consistency checking process with error leakage/wasted time measures from three sources

pr_4. We thus have:

$$el(p_4) = cardinality(o3) \times er_4$$
$$wt(p_4) = cardinality(o3) \times cr_4$$

Finally, the overall objective function is:

$$process\text{-}cost = el(p_4) + wt(p_4)$$

where higher values of *process-cost* indicate more work for the analysts and, consequently, lower *process-cost* values are desired.

3.2. Determining the Distributions for the Input Factors

Information retrieval (IR) techniques have been used in a variety of process/tools supporting requirements engineering and software engineering, most notably in requirements tracing. To model the distribution of the input variables for our model, i.e., the set of error and cost rates of the used techniques, we sought published quality levels for existing techniques. We undertook a limited literature review, using appropriate search terms in the Google search engine as well as ACM Digital Library to retrieve articles, URLs, and related sources of information on IR technique accuracy.

In Table 1, we show the values, summarized as lower and upper bounds for some techniques, as well as the source of the data and the dataset under evaluation when the measures were captured. For example, VSM with tf-idf weighting has a pr value of 0.276 with ar of 0.829 for the LEDA dataset. VSM has a lower bound for er of 0.000 and an upper bound of 0.746, while ar ranged from 0.254 to 1.000. The set of IR techniques have a lower bound of 0.110 and an upper bound of 0.540 for pr. Table I focuses on IR techniques because they are used to accomplish sub-processes P_1 and P_2 in our running example of consistency checking.

Parts of speech (POS) taggers have been used in a variety of process/tools supporting requirements engineering and software engineering, most notably in requirements analysis, in

Table I. Measures for IR techniques

	er	ar	cr	pr	Lit Source	Dataset
VSM tf-idf	0.746	0.254	0.886	0.114	[5]	MODIS
	0.000	1.000	0.862	0.138	[52]	Albergate
	0.500	0.500	0.567	0.433	[52]	Albergate
	0.276	0.724	0.829	0.171	[53]	LEDA
	0.100	0.900	0.830	0.170	[54]	eTour
[low,upp]	[0.000, 0.746]	[0.254, 1.000]	[0.567, 0.886]	[0.114, 0.433]		
LDA	0.400	0.600	0.890	0.110	[54]	eTour
LSI	0.000	1.000	0.836	0.164	[52]	Albergate
	0.140	0.860	0.788	0.212	[52]	Albergate
	0.000	1.000	0.882	0.118	[52]	LEDA
	0.167	0.833	0.460	0.540	[52]	LEDA
	0.100	0.900	0.830	0.170	[54]	eTour
[low,upp]	[0.000, 1.000]	[0.833, 1.000]	[0.460, 0.882]	[0.118, 0.540]		
Jensen Shannon	0.090	0.910	0.830	0.170	[54]	eTour
IR summary values [low,upp]	[0.000, 0.746]	[0.254, 1.000]	[0.460, 0.890]	[0.110, 0.540]		

Table II. Measures for POS tagging techniques

	ar	Lit Source	Dataset
OpenNLP	0.966	[55]	OpenNLP training data
Stanford Parser	0.973	[56]	Penn Treebank WSJ
CLAWS	[0.950, 0.960]	[57, 58, 59, 60, 61]	Penn Treebank WSJ
LBJ	0.966	[62]	Wall Street Journal corpus
Minipar	0.800	[63]	SUSANNE corpus
POS summary values [low, upp]	[0.800, 0.973]		

particular for consistency checking. We examined the literature and found the following information on the accuracy of a number of POS tagging techniques. In Table II, we show the lower and upper bounds for the measures as well as the source of the data and the dataset from which the values were obtained. For example, the CLAWS parser has a lower bound of 0.0400 for er and an upper bound of 0.050; a lower bound of 0.950 for ar and an upper bound of 0.960 when applied on the Wall Street Journal (WSJ) subset of the Penn Treebank dataset (recall that $er + ar = 1$). POS tagging techniques are used to accomplish sub-process P_3.

Semantic role labelers are used more and more frequently as researchers and practitioners alike realize that syntactic analysis of natural language text will not suffice for today's applications. These role labellers glean semantic information from natural language text by ascribing meaning, in a contextual setting, to tokens in texts. These techniques support processes/tools such as translation of requirements to LTL. We examined the literature and found the following information on the accuracy of a number of SRL techniques. In Table 3, we show the lower and upper bounds for the measure and the source of the data and the datasets used to collect the values. For example, Senna has a lower bound of 0.550 for er and an upper bound of 0.570; a lower bound of 0.430 for ar and an upper bound of 0.450 when applied on biomedical texts. SRL are used to accomplish sub-process P_4 during consistency checking.

Finally, consistency checking of temporal requirements using the TeAL technique was studied and applied to the 511 phone system in California [64]. During a previous study [65], we collected the same measures described above, the values are shown in Table 4. Our approach uses the techniques introduced above: VSM, Stanford parser, and Senna. The sub-process P_1, identifying temporal requirements, has 0.288 for er, 0.712 for ar, 0.516 for er, and 0.484 for pr. The sub-process P_2, identifying non-temporal related requirements, has 0.559 for er, 0.441 for ar, 0.516 for er, and 0.484 for pr. The sub-process P_3, tagging parts of speech, has 0.143 for er and 0.857 for ar. The sub-process P_4, identifying semantic roles (predicates and arguments), has 0.130 for er, 0.870

Table III. Measures for SRL techniques

	er	ar	cr	pr	Lit Source	Dataset
Senna [low, upp]	[0.550, 0.570]	[0.430, 0.450]	[0.320, 0.290]	[0.680, 0.710]	[66]	Biomedical texts
ISRL	0.368	0.630	0.324	0.666	[67]	WSJ from PropBank
NLPNet SRL	0.302	0.598	0.328	0.662	[68]	PropBank-Br
SRL summary [low, upp]	[0.302, 0.570]	[0.430, 0.630]	[0.320, [0.328]	[0.662, 0.710]		

Table IV. Measures for TeAL technique

	er	ar	cr	pr	Lit Source	Dataset
P_1-VSM	0.288	0.712	0.516	0.484	Unpublished notes	Phone 511
P_2-VSM	0.559	0.441	0.516	0.484	Unpublished notes	Phone 511
P_3-Stanford	0.143	0.857			[65]†	Phone 511
P_4-Senna [low, upp]	[0.130]‡ [0.385]‡	[0.870]‡ [0.615]‡	[0.167]‡ [0.294]‡	[0.833]‡ [0.706]‡	Unpublished notes	Phone 511

for ar, 0.167 for cr, and 0.833 for pr when we use Senna to find predicates, and has 0.385 for er, 0.615 for ar, 0.294 for cr, and 0.706 for pr when we use Senna to find arguments.

3.3. *Sensitivity Analysis*

A sensitivity analysis is used to assess the impact of variations in the inputs of a mathematical model on its outputs to understand the impact of uncertainty in the inputs on the results of the model as well as assessing the robustness of the model, i.e., does the model produce aberrant results if the inputs vary slightly or greatly? Sensitivity analysis provides a ranking of the model inputs based on their relative contributions to model output variability and uncertainty [69]. In this study, we seek to assess the impact of the variation of input factors ar and pr on the overall output (which, due to the identity relationships, in turn tells us about er and cr), *process-cost*, of the requirement consistency checking process.

To do this, we first determined the distribution of ar and pr for each sub-process. We generated data for the sub-processes P_1 ar_1 and pr_1 values, P_2 ar_2 and pr_2 values, P_3 ar_3 value, and P_4 ar_4 and pr_4 values in three cases: a) we took the data provided in Tables 1 through 4 in Section 4 and used the lower bound as a worst case, upper bound as a best case, and modeled a uniform distribution; b) we assumed a mean of 0.5 and standard deviation of 0.125 for all values and assumed a normal distribution; and c) we used the values from our empirical study on TeAL [13] and considered them as the means of a normally distributed distribution and used expert opinion to estimate variance (plus or minus 10%). We use uniform distribution because this method gives equal weight to each value within the lower and upper bounds, and is commonly used in sensitivity analysis researches whose main object is to understand model behavior [70, 71, 72]. We use 0.5 as mean and 0.125 as standard deviation because the assessments imply that in such cases the standard deviation is about one fourth of the mean [69]. We estimate the variance as plus or minus 10% because this method has been used in studies [73] to simulate the uncertainty on the mean variance. We randomly generate 500 samples for each case, varying one factor at a time, keeping all other measures set at their mean values, e.g., the mean of the values from literature. This data serves as the input for step three of our research approach.

In step three of our methodology, we generate XY scatter plots for each input factor and the objective function with all the samples. We apply a virtual trend line to the scatter plots and analyze them. We look for possible unexpected plots that may indicate an error in the model. We also look for

†We found the Stanford parser to be 0.937 accurate.
‡Predicates on the first line and arguments in the second line.

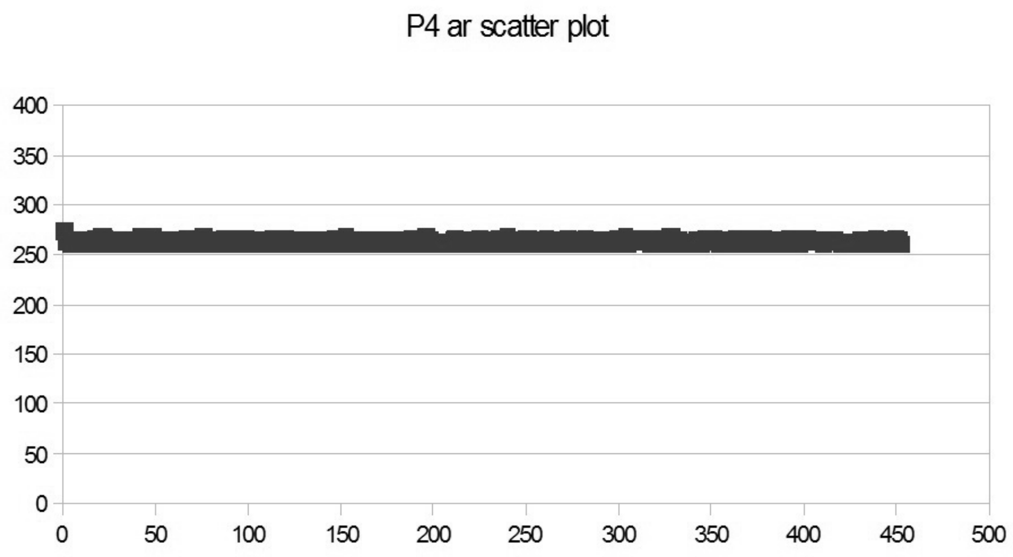

Figure 2. P4 ar scatter plot

interesting trends that may indicate that one factor is highly correlated with the objective function, i.e., the output factors.

4. ANALYSIS RESULTS FOR ERROR LEAKAGE AND WASTED TIME

We examined the three data sources for each of our model factors. We then modeled the process using two different sub-process orderings as well as modeled one sub-process as optional. We examine the output objective function, *process-cost*, as well as the ar and pr of the output result that would be provided to the analyst for evaluation.

4.1. Literature Values

The literature values for the seven factors from Table IV (retrieval and error rates ar_1 and pr_1 for P_1, ar_2 and pr_2 for P_2, ar_3 for P_3, and ar_4 and pr_4 for P_4) were examined and used to generate output objective-function values using scatter plots. The resulting plots were fairly unremarkable so for brevity we present one plot and merely describe the rest here. The data is chaotic for P_2 ar and pr as well as for P_4 pr. P_1 ar has a clear lower bound around 190 and an upper bound of 350. As can be seen in Figure 2, P_4 ar has a columnar shape between 0.450 and 0.625 yielding objective function values bounded by 268 and 280.

4.2. Analysis of Synthetic Data

Figure 3 shows an exemplary scatter plot for the synthetic data for P_2 ar. Note that all such data was generated using a normal distribution with a mean of 0.5. We use this method to generate the data because this is a commonly used method for understanding model behavior in sensitivity analysis. The figure shows that P_2 ar is bounded between objective function values of 125 and 290. P_1 ar ranges between 262 and 274. P_1 pr is tightly clustered at 272. P_2 pr is bounded tightly at 293. In contracts, P_3 ar values range from 276 to 286. P_4 is bounded by 265 (lower bound) and 305 (upper bound). Finally, P_4 pr is tightly clustered at 288. We note that the synthetic data pr for P_4 has fewer variations than for the literature values.

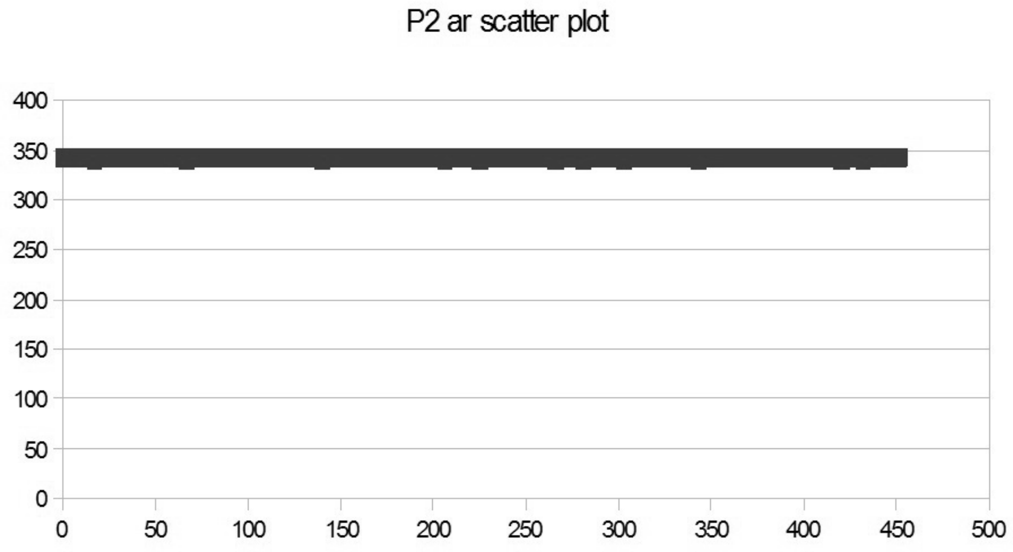

Figure 3. P2 ar scatter plot

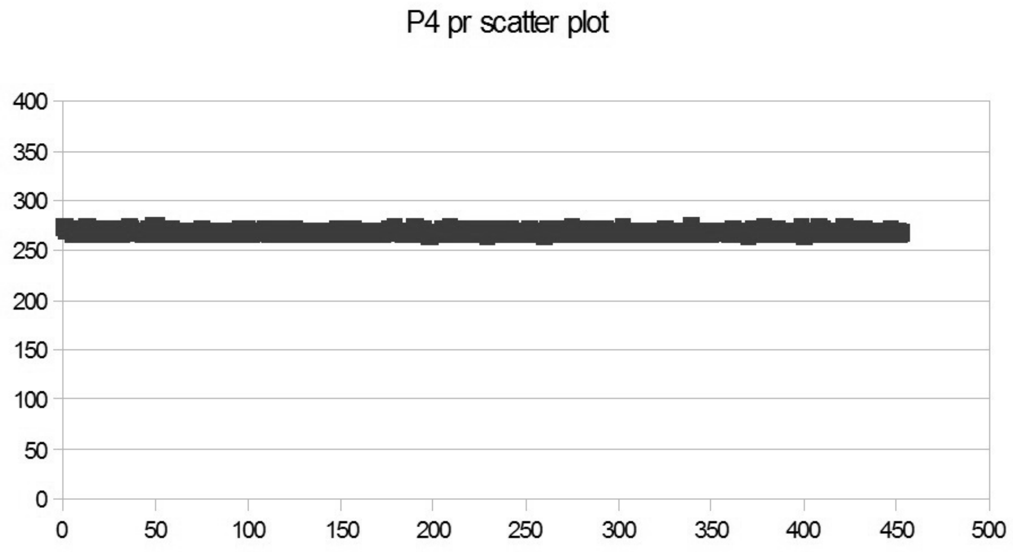

Figure 4. P4 ar scatter plot

4.3. Analysis of TeAL Data

The TeAL study data from Table IV was examined. P_1 ar was found to range very little from 273 to 274.9. P_1 pr is clustered tightly around 263 (and 0.470 for pr). P_2 ar is tightly clustered at 226 and 0.433 for pr. P_3 ar is also tightly clustered, but at 274. P_4 ar is columnar from 260 to 267. As can be seen in Figure 4, P_4 pr appears to be bounded between 265.5 and 271.

4.4. Analysis of Impact of Ordering and Optionality

Next, we examined the potential orderings of the sub-processes. Order 1 describes the "nominal" organization of the sub-processes as described in Section 3: P_1 (retrieve temporal requirements) followed by P_2 (retrieve related requirements) followed by P_3 (parts of speech tagging) followed by P_4 (semantic role labeling). It is also possible to perform parts of speech tagging (P_3) as the first sub-process. There is evidence that retrieval is enhanced when parts of speech tags are present [74]. Analogously, it is posited that semantic role labels could also assist with retrieval, so P_4 (semantic role labeling) could also be performed prior to retrieval activities. Thus Order 2 is defined as P_3 followed by P_4 followed by P_1 followed by P_2.

In addition to possible variations in the ordering of the sub-processes, it is also possible to omit certain sub-processes. A classic example is the ongoing "controversy" of whether or not to use stemming as a pre-process to retrieval. Proponents argue that ar is improved as related terms such as computer, computing, and computed will all be retrieved due to the common stem "comput-". Opponents argue that such stemming can result in increased false positives and thus lowered precision, when terms such as "user" and "usefulness" are deemed similar/related due to the common stem "us-". In some processes, stemming would be performed prior to retrieval. In other processes, stemming would be omitted. In this vein, we note that the retrieval of related requirements could be considered as an optional sub-process. These requirements serve to provide a larger context for the temporal requirements but their retrieval also results in many false positives. Our Optional sub-process is thus P_2, providing a three step sub-process ordering in addition to the four step ordering described above. Three step Order 1 is P_1 followed by P_3 followed by P_4. Three step Order 2 is P_3 followed by P_4 followed by P_1.

4.4.1. Sensitivity to Varied Factor Values To investigate the sensitivity of these Orders to the values of the sub-process factors (P_1 ar and pr, P_2 ar and pr, P_3 ar and P_4 ar and pr), we performed a study and varied the values of each sub-process holding all constant while varying one sub-process at a time. We put all other factors at their upper bound and then set the sub-process of interest to its lower bound (pr and ar or just ar in the case of P_3) or set all sub-process factors to their lower bound and set the sub-process of interest to its upper bound. We used the lower and upper bound from the union of the literature data, synthetic data, and TeAL study data. There are ten scenarios examined:

- Scenario 0 (All max) All factors set to their upper bound
- Scenario 1 (P1 min) All factors set to upper bound except P1 ar and er which are set to the lower bound
- Scenario 2 (P2 min) All factors set to upper bound except P2 ar and er which are set to the lower bound
- Scenario 3 (P3 min) All factors set to upper bound except P3 ar which are set to the lower bound
- Scenario 4 (P4 min) All factors set to upper bound except P4 ar and er which are set to the lower bound
- Scenario 5 (P1 max) All factors set to lower bound except P1 ar and er which are set to the upper bound
- Scenario 6 (P2 max) All factors set to lower bound except P2 ar and er which are set to the upper bound
- Scenario 7 (P3 max) All factors set to lower bound except P3 ar which is set to the upper bound
- Scenario 8 (P4 max) All factors set to lower bound except P4 ar and er which are set to the upper bound
- Scenario 9 (All min) All factors set to lower bound

We also varied the importance of a lost link to take into consideration the possible relative difficulty of finding a lost link (errors of omission) versus vetting a false positive (errors of commission). We weighted a lost link as having a weight of 1, 2 (meaning it is twice as hard to

Table V. Order 1, four steps, weight 1,2,5

Secnario	Lost link weight 1	Lost link weight 2	Lost link weight 5
0 (All max)	161	199	315
1 (P1 min)	182	226	365
2 (P2 min)	306	390	643
3 (P3 min)	167	216	365
4 (P4 min)	185	243	417
5 (P1 max)	312	403	677
6 (P2 max)	206	279	498
7 (P3 max)	331	428	719
8 (P4 max)	327	421	705
9 (All max)	332	430	727

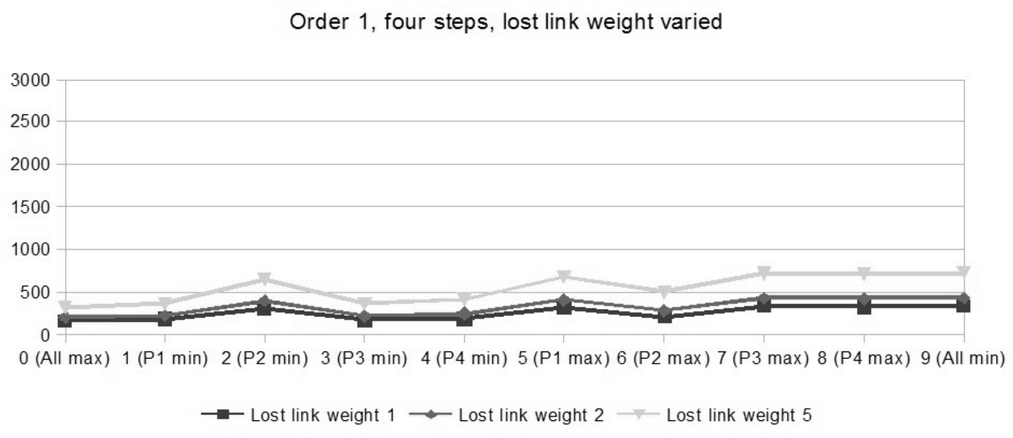

Figure 5. Order 1, four steps, varied weight of lost links

retrieve a lost link as it is to vet a false positive), and 5 (five times as hard). There is precedence for simulating errors this way in prior trace recovery research [75]. We captured the objective function value for each such scenario.

Table XV depicts the results for Order 1, four step process with weights of 1, 2, and 5 for lost links. Scenarios are in the columns and weights of lost links are in the rows. As can be seen in row 1, when all values for all factors were set to their maximum with a lost link weight of 1, the objective function was 1601. This was the lowest value of all the Order 1 scenarios, though P_3 variations caused a close result. In the row labeled 3 (p3 min), we see that when all factors are held to the maximum except for P_3 ar and pr, we achieve an objective function of 167, very close to the overall best.

Figure 5 depicts these scenarios graphically. As can be seen, the lowest process-cost occurs for the lost link weight of 1 with all factors set to their maximum values (upper bound) 161. This is not surprising. When all other factors are at their upper bound and P_3 ar is at its lower bound, the process-cost is very close to that best value, at 167. The lost link weight of 2 behaves much the same way as the lost link weight of 1, but with higher process-cost. Lost link weight of 5 has much higher process-cost values.

Of interest is the ar, pr that results for the notable process-cost values for link weight 1. Table XVI provides the resulting ar/pr (at the end of all sub-processes) for the scenarios. ar is 0.103 and pr is 0.045 for the All min scenario. For the All max scenario it is much higher at 0.648 ar and 0.325 pr. For the P_3 min scenario it is 0.549 and 0.307. As can be seen and as highlighted in red, the All min scenario is not in the "sweet spot" for process output that most benefits analysts, discussed in

Table VI. Resulting ar/pr for scenarios - Order 1, four steps, link weight 1

Secnario	Ar	Pr
0 (All max)	**0.648**	**0.325**
1 (P1 min)	**0.580**	**0.276**
2 (P2 min)	0.232	0.104
3 (P3 min)	**0.549**	**0.307**
4 (P4 min)	0.471	0.232
5 (P1 max)	0.170	0.078
6 (P2 max)	0.336	0.173
7 (P3 max)	0.120	0.051
8 (P4 max)	0.140	0.061
9 (All max)	0.103	0.045

Table VII. Order 2, four steps, weight 1,2,5

Secnario	Lost link weight 1	Lost link weight 2	Lost link weight 5
0 (All max)	428	622	1202
1 (P1 min)	448	649	1251
2 (P2 min)	623	891	1675
3 (P3 min)	457	705	1450
4 (P4 min)	548	818	2042
5 (P1 max)	751	1152	2362
6 (P2 max)	576	912	1918
7 (P3 max)	763	1143	2255
8 (P4 max)	672	1002	1992
9 (All max)	771	1178	2411

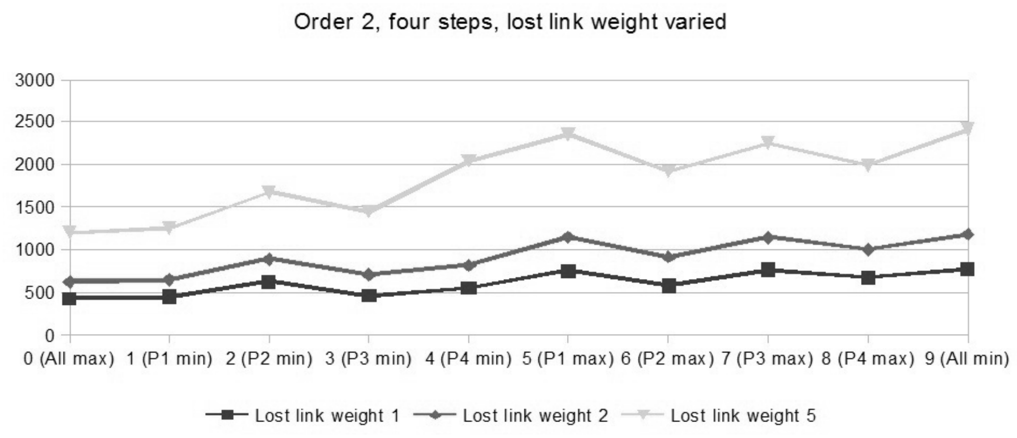

Figure 6. Order 2, four steps, varied weight of lost links

Section 1 (of ar between 0.25 and 0.62 and precision between 0.25 and 0.5). The All max scenario does result in ar/pr in the sweet spot as do the P_3 min and P_1 min scenarios.

Next we examine the Order 2, four step scenarios. Table XVII presents the process-cost values for these scenarios. Figure 6 depicts the values graphically. As can be seen, the All max scenario has the lowest process-cost at 428. The next best value is for P_1 ar and er (scenario 1) set to their lower bound with a process-cost of 448. The All min scenario for lost link weight of 1 has a process-cost of 771. Figure 6 shows that lost link weight 1 outperforms the other two lost link weights. In

Table VIII. Resulting ar/pr for scenarios - Order 2, four steps, link weight 1

Secnario	Ar	Pr
0 (All max)	0.613	0.915
1 (P1 min)	0.598	0.844
2 (P2 min)	**0.464**	**0.510**
3 (P3 min)	0.504	0.814
4 (P4 min)	**0.418**	**0.584**
5 (P1 max)	0.195	0.217
6 (P2 max)	**0.330**	**0.472**
7 (P3 max)	**0.254**	**0.255**
8 (P4 max)	**0.340**	**0.378**
9 (All max)	0.180	0.191

Table IX. Order 1, three steps, weight 1,2,5

Secnario	Lost link weight 1	Lost link weight 2	Lost link weight 5
0 (All max)	167	210	338
1 (P1 min)	327	420	698
3 (P3 min)	174	228	392
4 (P4 min)	194	258	450
5 (P1 max)	196	268	484
7 (P3 max)	334	432	727
8 (P4 max)	329	424	712
9 (All max)	334	434	736

general, changes to P_1 ar and pr and to P_3 ar to their lower bound resulted in lower process-costs than adjustments to other factors.

The ar and pr of several scenarios for lost link weight 1 are shown in Table XVIII. The All max scenario yields a resulting ar, pr of 0.613, 0.915. In contrast the All min scenario has a fairly dismal ar, pr of 0.18, 0.191. Note that the ar, pr for P_1 ar and er set to their lower bound results in ar, pr of 0.598, 0.844 very close to the All max scenario. This is not intuitive, but interesting to note. Looking at our ar, pr sweet spot we see that P_2 min, P_4 min and P_2, P_3, and P_4 max scenarios all fall in the desired range. The precision is too high for the All max scenario and P_1 min and P_2 min scenarios and both ar, pr are too low for the P_1 max and All min scenario.

Next, we examine the two Orders with three sub-processes, sub-process P_2 has been omitted. Table XIX and Figure 7 depict the resulting process-cost values for Order 1, three step, with lost link weight varied. The All max scenario for lost link weight 1 has the lowest process-cost at 167. The next best is P_3 ar set to its lower bound with a resulting process-cost of 174. The figure shows that the sub-process "max" scenarios are much less desirable than the sub-process "min" scenarios in terms of the resulting process-cost.

Table X presents the ar and pr for the scenarios. The best ar and pr resulted for the All max scenario with ar of 0.613 and pr of 0.291. The P_3 min scenario was a close second with ar of 0.504 and pr of 0.268. Both of these scenarios result in ar, pr pairs in the analyst sweet spot. The All min scenario is much worse with ar and pr of 0.087 and 0.038, far from the sweet spot. None of the other scenarios fall in the sweet spot.

Table XI and Figure 8 show the Order 2, three step, varied lost link weight results. The lowest process-cost results from the All max scenario. The P_3 min scenario has the next lowest value followed by P_4 min. It is clear from the figure and table that the All min scenario is much worse than the All max and P_3 min scenarios.

Table XII depicts the ar, pr values for each of the Order 2, three step scenarios for lost link weight of 1. It is interesting that all of the scenarios **except** for All max, All min, and P_3 min result in ar,

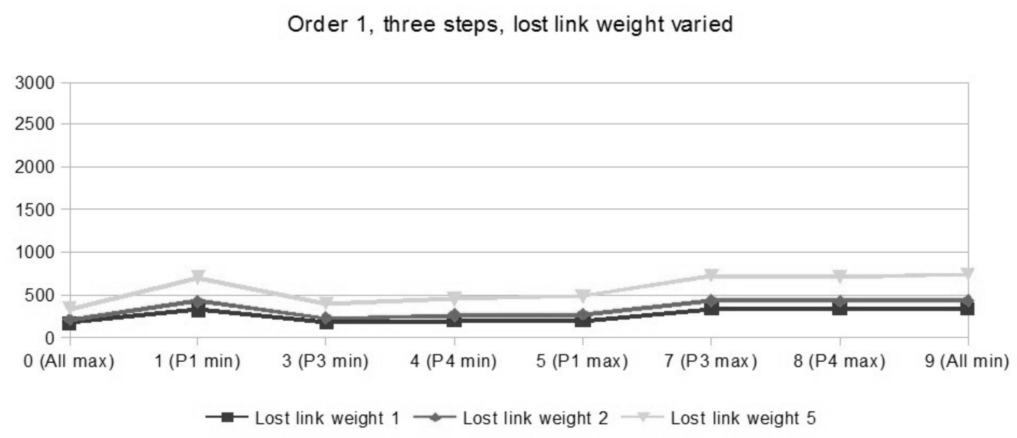

Figure 7. Order 1, three steps, varied weight of lost links

Table X. Resulting ar/pr for scenarios - Order 1, three steps, link weight 1

Secnario	Ar	Pr
0 (All max)	**0.613**	**0.291**
1 (P1 min)	0.156	0.066
3 (P3 min)	**0.504**	**0.268**
4 (P4 min)	0.418	0.194
5 (P1 max)	0.344	0.179
7 (P3 max)	0.106	0.045
8 (P4 max)	0.128	0.055
9 (All max)	0.087	0.038

Table XI. Order 2, three steps, weight 1,2,5

Secnario	Lost link weight 1	Lost link weight 2	Lost link weight 5
0 (All max)	428	621	1202
1 (P1 min)	642	918	1745
3 (P3 min)	457	706	1449
4 (P4 min)	548	840	1712
5 (P1 max)	557	885	1869
7 (P3 max)	763	1136	2254
8 (P4 max)	763	1002	1992
9 (All max)	771	1181	2411

pr in the sweet spot. In general, the All max scenarios have resulted in sweet spot results, but for this Order and step combination the precision is too high for the sweet spot.

4.4.2. Sensitivity to Order and Optionality Here we compare the results of Order 1 and Order 2 as well as three steps versus four step processes. Table XIII presents the rank orders of the varied orders and number of sub-processes when the ar and pr values for all sub-processes are set to their upper bound. From this table it is clear that Order 1, four steps has the lowest process-cost followed by Order 1 with three steps. The Order 2 processes follow Order 1 with much higher process-costs of 428 for each. For Order 2, we can see that an optional sub-process can be removed with no additional effort for analysts.

The ar, pr pair for Order 1, four steps All max scenario is 0.648, 0.325 and for Order 1, three step it is 0.613, 0.291. These are both in the analyst sweet spot. In contrast, the Order 2, 4 step is at

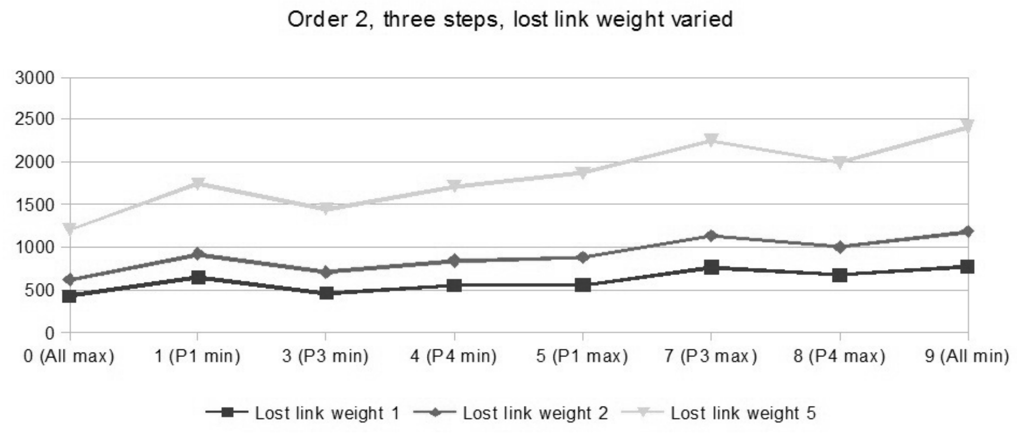

Figure 8. Order 2, three steps, varied weight of lost links

Table XII. Resulting ar/pr for scenarios - Order 2, three steps, link weight 1

Secnario	Ar	Pr
0 (All max)	0.613	0.889
1 (P1 min)	**0.449**	**0.470**
3 (P3 min)	0.504	0.788
4 (P4 min)	**0.418**	**0.568**
5 (P1 max)	**0.344**	**0.508**
7 (P3 max)	**0.254**	**0.255**
8 (P4 max)	**0.340**	**0.376**
9 (All max)	0.180	0.191

Table XIII. Varying order and number of processes - with ar, pr at max (upper bound)

order/num of sub-processes	**process-cost**	**rank**
order_1_4_step	160.6	1
order_1_3_step	167.3	2
order_2_4_step	428	3
order_2_3_step	428	3

0.613, 0.915 and is not in the sweet spot. Order 2, three step is at 0.613, 0.889, also not in the sweet spot.

We see that processes with three sub-processes can land in the sweet spot. We also see that Order 2 can achieve similar process-costs as four step sub-processes. In addition, earlier sections showed that occasionally a lower quality sub-process (put at its min value or lower bound) still resulted in low process-costs and ar, pr in the sweet spot. This is useful as we may have reasons for wanting to select a lower quality technique for certain sub-processes, perhaps for ease of integration into our process (e.g., our tools are all written in Java and the higher quality technique is in a language that could be difficult to integrate). We can see from such analyses that a lower quality technique is acceptable at certain sub-processes without much impact to our overall quality.

5. IMPACT OF ERROR LEAKAGE AND WASTED TIME ON ANALYSIS

The previous sections shows the large difference in performance between different configurations of the sub-processes P_1 to P_4, including which techniques to use, the order of the techniques, and the optional techniques. They showed that the choice of the techniques influence the quality of the overall process. They did not consider the analysts' efforts and their opportunities to change the outcome of the processes through verification/validation and, hence, to impact the quality of the overall process.

In this section, we simulate the actual processes used by analysts for checking the consistency of temporal requirements to measure the actual variation in number of errors and wasted time due to different accuracy of the four sub-processes P_1 to P_4 as well as the sensitivity of the overall process to technique accuracy and/or human errors. Based on the result in section 4, we model the process of Order 1 with 4 steps.

First, we discuss the assumptions that we make for the simulation, followed by our genetic algorithm implementation, the experiments that we performed, and their results.

5.1. *Assumptions*

We assume a working day of eight hours, divided into a set of working periods, wps. Each wp lasts two hours. We further assume that an analyst cannot work two consecutive wps to simulate fatigue and interruptions. These assumptions are legitimate because they fit a "regular" schedule of 40 hours of work per week and two straight hours of focused work. It is also realistic because our industry experience indicates that reviewing lists of retrieved elements is tedious and time consuming work; lack of rest or shift in focus can lead analysts to introduce errors.

We assume that all analysts are equally competent and capable to verify the output of each sub-process. We make this assumption to simplify the simulation but also because, to the best of our knowledge, there is no previous work on the impact of inter-rater agreement in requirements engineering, in general, and consistency checking, in particular. Only El Emam [76] studied inter-rater agreement in software process assessments. Future work should be devoted to perform experiments to measure Cohen's kappa during consistency checking of requirements by different analysts.

Following the previous assumption and for the same reasons, we also assume that the verification efforts of each sub-process are known: efp is the effort to verify false positives, efn the effort to verify a false negative, etp the effort to verify true positives, and etn the effort to verify a true negative. We assume that these verification efforts are the same for all analysts and do not depend on the sub-process, the order of the tasks, or the fact that the same analysts previously verified the same element: no learning effect.

5.2. *Problem Formulation*

We model the problem as an array of integers, namely req, where each cell represents a requirement to be verified (i.e., the set $R = \{r_1, \ldots, r_n\}$) and a matrix, wpd, of dimensions $WP \times D$ representing the verification work plan. Recall that we denote by T the subset of R consisting of all temporal requirements in R and assume that $T = \{t_1, \ldots, t_m\}$. In our formulation, $req[l]$ represents the l-th element requirement of R and thus if $req[l]$ is equal to zero if it is not relevant (i.e., it is not in T) whereas its value is one if $req[l]$ is a temporal requirement (i.e., it belongs to T).

We assume that there are D available analysts and the overall process is assigned a maximum completion time of WP time slots, each time slot lasting T_{wp}. Each row in the wpd matrix represents a working period, denoted wp. Matrix columns represent analysts: a cell (i, j) contains the sub-process to which analyst j is assigned at a particular work period i; thus $wpd[i, j] = k$ means that analyst j in the working period i is assigned to the sub-process k. Recall that an analyst must rest after each work period; analyst j, after having performed the task in time slot i, must rest for $r \geq 1$ time slots before being assigned to a new task. Table XIV shows a sample matrix with three analysts and six working periods. In this matrix, the sub-process $P1$ is finished in the first two working periods. In the third period one analyst is resting (the cell is -1) and the other two are assigned $P2$.

Table XIV. Sample wpd matrix

Work Period	Analyst 1	Analyst 2	Analyst 3
1	0	1	1
2	1	-1	-1
3	-1	2	2
4	3	-1	-1
5	-1	3	0
6	4	-1	4

In the fourth period only one analyst is available, thus $P3$ can only be completed in the fifth period. Finally the whole process takes six working periods.

Overall, the verification process work plan uses as many analysts as there are non-empty columns in the matrix wpd; it lasts as long as the last row of wpd where a task is found. Analysts are assigned as many tasks as there are non-zero cells in the wpd matrix for a fixed column. Furthermore, each sub-process P_1 to P_4 is represented as the retrieval and error rates ar and er. The higher the error rates, the more time each sub-process will require and, thus, the longer the entire process will last. Our goal is to assign tasks to analysts, that is to fill the wpd matrix, in such a way as to minimize the process length for a given set of error rates, ar and er, and known verification/validation costs. We decompose the cost into three terms. The first term T_{work} considers the times that individual analysts are working. If a wp is two hours, then T_{wp} = 120 minutes. T_{wp} may be greater than 120 minutes if the analysts need more time to check the consistency of the requirements:

$$T_{work} = T_{wp} \sum_{1}^{D} \sum_{1}^{WP} \delta(wpd(wp, d))$$

where the function $\delta()$ returns one if the cell is not empty, zero otherwise.

The second term, overall process time, accounts for the total process length; it uses the last row in which an analyst was assigned to compute the total time of the consistency checking process. For example, if row 20 has the last entry for any analyst and T_{wp} is 120 minutes, then the total time is 120 $\times$ 20 = 2,400 minutes. The term is written as:

$$T_p = T_{wp} argmax_i[wpd(i, j)]$$

The third term quantifies the time wasted because of the quality of the results of a sub-process. If an analyst needs more than T_{wp} to finish her task, say 150 minutes, the difference of 30 minutes violates the constraint of two hours work/two hours rest. This term represents the penalty for not being able to finish checking some assigned requirements within a given work period:

$$T_{penalty} = \sum_{1}^{D} \sum_{1}^{WP} \alpha(T_{wp} - wpd(wp, d)) \times \frac{\beta(wpd(wp, d))}{T_{wp}}$$

where the function $\beta()$ returns the actual time needed to complete the task assigned to analyst d, in work period wp for the sub-process $wpd(wp, d)$, and $\alpha()$ is a penalization function increasing penalty for negative values of $T_{wp} - wpd(wp, d)$.

Overall, the objective function can be written as:

$$\begin{aligned} T_{work} + T_p + T_{penalty} = T_{wp} \textstyle\sum_1^D \sum_1^{WP} \delta(wpd(wp, d)) + T_{wp} argmax_i[wpd(i, j)] \\ + \textstyle\sum_1^D \sum_1^{WP} \alpha(T_{wp} - wpd(wp, d)) \times \frac{\beta(wpd(wp,d))}{T_{wp}} \end{aligned}$$

which we seek to minimize, i.e., to minimize the time taken by the analysts individually (T_{work}), the overall time taken by the analysts (T_p), and the length of the penalty because of extra work ($T_{penalty}$).

Finally, two situations can be simulated: the first assumes ideal analysts, who make no error and, thus, no error slips through the sub-processes; the second assumes that some errors slip through.

Table XV. Configuration of Genetic Algorithm

Initial Population Size	200
Crossover Rate	40%
Mutation Rate	5%
Elitism	5
Number of Generations	1000

In these two situations, the formulation remains unchanged as only the input of each sub-process (output of the previous sub-process) is changed by introducing errors in the arrays output by the sub-processes.

5.3. Implementation as Genetic Algorithm

Given the assumptions and problem formulation, our goal is to allocate the analysts to sub-processes to minimize the result of the objective function, which represents the analysts' times, the total time to complete the verification process, and the penalties, i.e., which represent the analysts' effort. We call fitness the result of the objective function. We resolve the problem with a genetic algorithm written in Java. Our implementation uses elitism and specialized mutation and crossover operators.

The initialization of each sub-process is performed as follows. First, an input array of 500 integers ($N = 500$), $req[l]$, representing R is created. Elements of $req[l]$ are randomly assigned zero and one, where the number of ones is based on T (the size of T is 100). These ones represent the set of temporal requirements in R. Then, each verification sub-process input is generated by introducing errors in the input array (mimicking the technique's behavior) based on its assigned pr and ar. ar and pr values for each sub-process are stored in an auxiliary array to ease computation. If a sub-process, say P_1, introduces an error, it adds 2 to the value of the current array position, given that 0 means a true negative, 1 means a true positive, 2 means a false positive, and 3 means a false negative. We model 1, 2 verification times as two minutes each, 3's verification time as four minutes each, and 0 as requiring no effort [77, 78].

The **initial population** is a set of work plans using D analysts and taking WP time periods. An initial work plan, i.e., a problem solution, is created as follows. The number of analysts D and of working periods WP is randomly selected between a minimum and maximum. The empty matrix wpd is then created and each analyst has an assigned number of requirements to verify. We estimate the number NP of time periods needed for processing each sub-process based on the elements in its array. We then randomly select analysts in a time period and randomly assign N/NP requirements to each of them. If there are requirements left, then we move to the next time period.

Mutation Operator: Given an individual column of matrix wpd, we can swap two random cells of which one must be non-zero. We thus swap the sub-process to which an analyst j is assigned during a work period wp. Thus, indirectly, we simulate the impact of the values of ar and er of the sub-processes on the consistency checking because the wps depend on ar and er, on which depend the times worked by analysts (T_{work}) and the overall time (T_p).

Crossover Operator: The crossover selection mechanism is based on the roulette wheel. We randomly select the number of columns to be swapped. We then take that number of columns from the right part of the columns of one possible solution and swap them with the left part of the columns of another solution. For example, if one wpd has 5 analysts and another wpd has 8 and we randomly select 3 as the number of columns to be swapped, we take the first three columns of the 8-length wpd and swap them with the first 3 of the 5-length wpd. Thus, we are swapping the assignments of work periods to analysts. Overall our genetic algorithm was configured as shown in Table XV.

To model the situation where an analyst must work more than one work period without rest, we add a penalization mechanism. If, in performing her task, an analyst must go beyond the allocated wp, then the resting time is increased by the extra time. So, if Analyst 2 is working on requirements in P_1 during work period 3 and this work takes 180 minutes, then the two hour work period is exceeded by 60 minutes and the rest penalty is 60 minutes.

5.4. Experiments

We performed experiments using the following four scenarios:

Scenario 1: We define an input configuration of 500 values (0, 1, 2, and 3 to represent true positives, false positives, etc. as defined above) and we compute the output of P_1 to P_4 based on the seven values from the literature survey (probability of false positives, etc.). We keep the inputs constant for 1,000 generations and we model the verification by perfect analysts, i.e., zero injected errors (that is, zero false positives selected as true links, zero true links rejected as false, etc.).

Scenario 2: Same as Scenario 1 above, but the analysts make some errors (e.g., inject incorrect links).

Scenario 3: Same as Scenario 1 above, but, at the beginning of each generation, we randomly select values (probability of false positives, etc.) that are within the range of the literature survey values for P_1 to P_4 to simulate analysts who use different techniques and thresholds. We use these values to compute the 500 values of 0, 1, 2, and 3.

Scenario 4: Same as Scenarios 2 and 3 combined: we add missed errors and inject wrong links.

For a fixed set P_1 to P_4 of seven real numbers (these are the values of P_1 ar, P_1 pr, P_2 ar, P_2 pr, P_3 ar, P_4 ar, P_4 pr), which we keep the same throughout all our experiments for comparison purposes, we run these four scenarios 50 times, then compare the input variations with the output variations. For example, consider the data that we collect across the different runs:

Run\type	Input	P_1	P_2	P_3	P_4	Fit-fct	Work Plan
1	array	array	array	array	array	array	two-d array
2	array	array	array	array	array	array	two-d array
3	array	array	array	array	array	array	two-d array
4	array	array	array	array	array	array	two-d array
				...			
50	array	array	array	array	array	array	two-d array

The input array is the array containing only 0s (true negative) and 1s (true positive), while the P_1 array is the output after P_1 has been reviewed (either perfectly, i.e., "as is" or with analysts' error, which would change the values by adding two to some of them). The two-d array for work plan shows what sub-process different analysts work on in what work period. It should be noted that ideally, we need to perform a direct comparison between the ar/pr data that is applied to the same dataset. However, much of the ar/pr data does not exist. So we apply the convenience sampling method and uses the ar/pr data collected from different datasets. We then compare the variation in each of the runs for each of the scenarios.

5.5. Results

We analyze the relationships among input arrays, $req[l]$, the output arrays of the four sub-processes, and the fitness values based on the results of Scenarios 3 and 4. We do not examine Scenarios 1 and 2 further because all runs use the same set of seven values for the sub-process accuracy, regardless of the number of analysts, thus the time costs of sub-processes in these runs are the same. Later we use Scenarios 1 and 2 to study the relationship among ar/pr values and the fitness values.

In Scenarios 3 and 4, each run uses a new input array and generates a new set of sub-process arrays. It should be noted that the input array is randomly generated based on the lower/upper bounds in Tables I-IV. In this way we simulate that different techniques are used in each sub-process. We calculate the time costs of these arrays and the Pearson correlation between fitness values and these time costs. Tables XVI and XVII show the results for Scenario 3, with 10 or 20 analysts. For example, the correlation coefficient between the input array and P_1 array is 0.798. Table XVI shows strong positive relationships between input and P_1 or P_2. The relationships between the fitness value and input or P_2 are also very strong. There is also a strong positive relationship between the fitness value and P_1. Table XVII shows some different results. While the relationship between input and P_1 is still very strong and positive, Table XVII implies that there is only a strong positive relationship, instead of a very strong positive one, between input and P_2. Similarly, there are strong positive relationships between the fitness value and input or P_2.

Table XVI. Pearson Correlation for Scenario 3 with 10 Analysts

	Input	P_1	P_2	P_3	P_4	Fitness
Input	1					
P_1	0.798	1				
P_2	0.756	0.223	1			
P_3	-0.094	-0.271	-0.016	1		
P_4	-0.051	-0.051	-0.072	0.075	1	
Fitness	0.865	0.612	0.730	0.015	-0.079	1

Table XVII. Pearson Correlation for Scenario 3 with 20 Analysts

	Input	P_1	P_2	P_3	P_4	Fitness
Input	1					
P_1	0.889	1				
P_2	0.574	0.149	1			
P_3	-0.029	-0.174	0.0519	1		
P_4	0.213	0.141	0.208	-0.154	1	
Fitness	0.681	0.535	0.487	0.143	0.181	1

Table XVIII. Pearson Correlation for Scenario 4 with 10 Analysts

	Input	P_1	P_2	P_3	P_4	Fitness
Input	1					
P_1	0.673	1				
P_2	0.514	-0.088	1			
P_3	-0.123	-0.223	-0.197	1		
P_4	0.054	-0.036	0.156	-0.243	1	
Fitness	0.888	0.622	0.579	-0.071	0.099	1

Table XIX. Pearson Correlation for Scenario 4 with 20 Analysts

	Input	P_1	P_2	P_3	P_4	Fitness
Input	1					
P_1	0.712	1				
P_2	0.518	-0.061	1			
P_3	0.492	0.317	0.135	1		
P_4	-0.211	-0.167	-0.103	0.054	1	
Fitness	0.818	0.731	0.444	0.488	-0.081	1

Tables XVIII and XIX show the results for Scenario 4 with 10 or 20 analysts. Table XVIII shows that, with error leakage, the only very strong positive relationship exists between the fitness value and the input. The relationships between input and P_1 or P_2 and between the fitness value and P_1 or P_2 are only strongly positive. However, Table XIX shows three very strong positive relationships. They exist between the input and P_1, the fitness value and input or P_1.

We also investigated the relationship between the seven ar/pr values and the fitness values. We used seven sets of fixed values as the input of Scenarios 1 and 2. The values in each set are the same. We selected 30%, 40%, 45%, 50%, 55%, 60%, and 70%. For example, with seven 50% as input (these are the values of P_1 ar, P_1 pr, P_2 ar, P_2 pr, P_3 ar, P_4 ar, P_4 pr), the ar and pr for all sub-processes are 50%. We use fixed ar/pr values so that we can focus on the relationship between the ar/pr values and the fitness values. Because the ar/pr values are fixed instead of randomly generated based on the literature survey, Scenarios 3 and 4 are exactly the same as Scenarios 1 and 2. Thus, we only run this part of the experiments for Scenarios 1 and 2.

Figure 9 shows the box plot for fitness values of Scenarios 1 and 2 with 10 analysts. It appears that the fitness values of Scenario 2 ($s2\_30$, ..., $s2\_70$) are lower than that of Scenario 1. It is notable that for five out of seven fixed values, fifty runs generate the same fitness values. This tendency is more obvious in Figure 10, which shows the fitness values of the two scenarios for 20 analysts. For each combination of scenario and fixed value, the fitness values of fifty runs are the same.

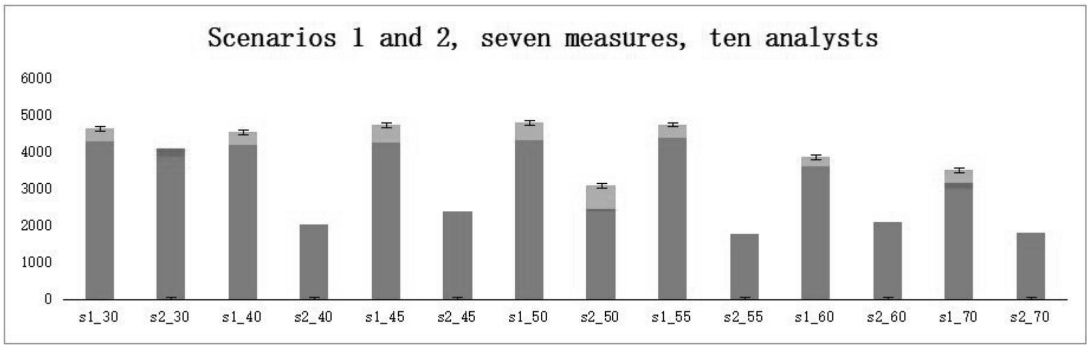

Figure 9. Box plot for fitness value, scenarios 1 and 2 with ten analysts, seven fixed measures

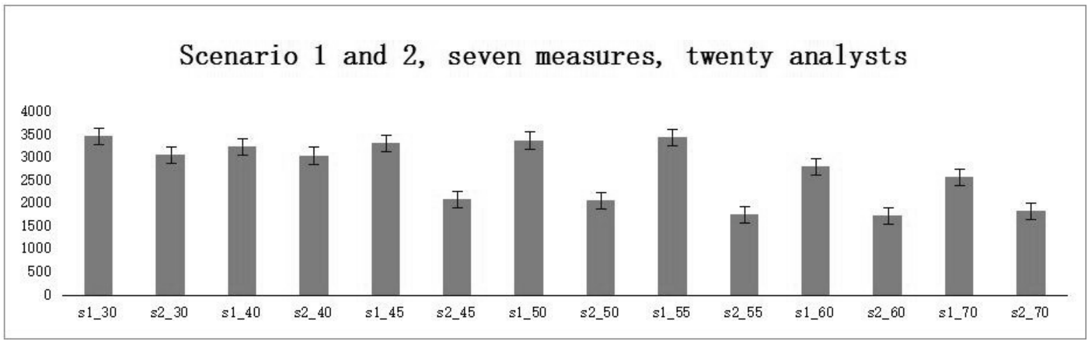

Figure 10. Box plot for fitness value, scenarios 1 and 2 with twenty analysts, seven fixed measures

5.6. *Discussion*

The results give some insights to the relationships among the pr/ar values, the input array, the sub-process arrays, and the fitness values. Tables XVI-XIX have something in common: the relationships among the input, P_1, P_2, and the fitness values are obvious, they are either very strongly positive or strongly positive. It may appear that the arrays of P_1 and P_2 are longer than those of P_3 and P_4 because of the number of elements in each process: we chose P_1 array with 500 elements, and P_2 with 400 elements, while P_3 and P_4 arrays each have 200 elements. However, when we tried to vary the number of elements in P_1 and P_2, we saw little change in the relationship between these two process. Thus we believe that P_1 and P_2 are related by the technology we used. Also, the effects of P_1 and P_2 on the fitness values are greater than that of P_3 and P_4 as shown in Tables XVI-XIX in Scenario 3.

The Pearson correlations between the fitness values and input/P_1/P_2 of Scenario 3 are higher than that of Scenario 4. In Scenario 4, many false negative elements are mistakenly treated as true negatives and the time that should be spent on these false negative elements (four minutes for each element) is saved. This decreases the time cost but introduces errors. The impact of imperfect analysts also shows in Figures 8 and 9. The fitness values of Scenario 2 are smaller than those of Scenario 1.

Another interesting observation illustrated by Figure 9 is that, with 20 analysts and a set of fixed values, the fitness values of each run are the same. With 20 analysts, a sub-process can be finished in one time period. In this case, the fitness value is already optimal. It is easy to generate such a "best wpd" at the beginning of the GA.

Figure 8 also shows five cases in which all runs have the same fitness values. However, only the runs of Scenario 2 have this tendency because the imperfect analysts reduce the total time costs. The fitness values of other runs, with higher time costs, cannot be optimized so easily with only 10 analysts.

We thus showed that error leakage negatively impacts the analysts' efforts when there are not enough analysts to perform all verification of the outputs of the sub-processes within the allocated work periods. It appears that in such cases, the quality of the process outputs is out of the "sweet spot," which prevents analysts from making the most improvement to the results. Therefore, analysts should strive to prevent error leakage by using appropriate sub-processes and by performing high quality verifications.

6. THREATS TO VALIDITY

Construct validity is threatened by our choice of considering only retrieval rates and precision values in our model. It is possible that a determinant factor in the results of a process is the dataset or some other unknown factor. We limited this threat by performing three analysis using three sets of values: from the literature, centered around 0.5 mean, and from the TeAL study. Yet, future work should try to consider other factors.

Content validity is threatened by our use of lower and upper bounds from the literature. Getting values for many of the techniques from the literature (see Section 4) and using those literature values to "limit" our search threaten the content of our study because these bounds may artificially constraint the search. We limited this threat by considering many different works, from different authors and on different datasets. Yet our chosen references form a survey of convenience because we used as our search strings a set of keywords related to the validated accuracy of each technique category and did not perform a systematic literature review.

Internal validity of the obtained results is not threatened because our model accounts only for the ar and pr values of the different techniques used in the requirement consistency checking process.

Conclusion validity seems high because there is intuitively a relation between the results obtained from an overall process and the ar and pr values of the techniques used in this process.

External validity is threatened by our choice of the requirement consistency checking process as the running example. The results and conclusions obtained here may not apply to different requirements processes. Future work is necessary to replicate our study on other processes, made of other techniques, with other ar and pr values.

7. CONCLUSION AND FUTURE WORK

Various techniques exist to build processes to solve problems in requirement engineering, such as information retrieval, natural language processing, classification, clustering. Complex requirements problems are typically addressed using a collection of such techniques piped together. Existing techniques are powerful, but not perfect, and errors leak from one technique to the next much as defects leak into subsequent software development phases. Errors due to the imperfect and imbalanced precision and recall of piped techniques will impact the accuracy of the final result and will result in wasted time due to false positives introduced into the process.

Errors in the techniques have been the subject of much work but researchers focused on the quality of individual techniques, such as improving the recall and precision of retrieval techniques. In this paper, we presented a first study of the error leakage between techniques and its impact on the accuracy and wasted effort resulting from this error leakage. We showed, through an exploratory sensitivity analysis of a consistency checking process, that error leakage indeed impacts the accuracy and effort of an analyst. It was also clear from the model of the analyst verification of the sub-process results that error leakage negatively impacts analysts' effort when there are not enough analysts to perform all verification of the outputs of the sub-processes within the allocated work periods. Analysts should wisely select sub-processes and strive to perform high quality verification. In addition, it is possible that longer rest periods and/or shorter work periods would assist with reducing error leakage and thus wasted time.

Following the study presented in this paper, we will move in several directions: one direction is to learn the features that are inherent in software artifacts and lead to errors in the techniques. For example, we can attempt to learn what inherent characteristics of a text lead it to have incorrect parts of speech tags assigned. By having many training examples, we can move toward developing a decision support assistant that will assist with configuration from the learned examples. We will take into consideration simulated or synthetic data. The next direction is to consider the case of requirements or elements where some of the POS tags or semantic roles are correct and some are not, rather than using simplifying assumptions. We plan to model the impact of the analyst when some requirement engineering process has an analyst-in-the-loop. As mentioned in the Introduction, studies have been conducted that indicate the relative quality of analyst work as they vet the results of an automated trace recovery tool [7, 8, 10]. Such values can be used for future modeling. In addition, trace recovery researchers are continuing to study the accuracy of analysts in retrieving results missed by automated tools. Those emerging results will also be used in our future work of modeling the impact of the analyst in the loop. Besides, we assume that all analysts are equally competent in Section 5. We will perform experiments to study the cases in which different analysts are used in consistency checking. Another direction is to explore more factors other than the quality of the sub processes that may affect analysts' effort. In this paper, we have studied the order and optionality of the techniques. Other factors remain as future work. Finally, we plan to perform a systematic literature survey on techniques used in software engineering processes in general and requirement engineering processes in particular. This future work will help us in answering the more general question: How are we to know what techniques to select for each sub process?

Future work should be devoted to perform experiments to measure Cohens kappa during consistency checking of requirements by different analysts.

ACKNOWLEDGEMENT

We thank Drs. Stromberg and Bathke, and Joshua Lambert of the Statistics Dept. for reviewing earlier versions of the paper. We thank the NSF, under grant CCF-1511117, for partially funding this research.

REFERENCES

1. OMG. Software process engineering metamodel spem 2.0 omg specification. *Technical Report*, ptc/backslash 07-11-01,OMG 2008.
2. Biggerstaff TJ, Mitbander BG, Webster DE. Program understanding and the concept assignment problem. *Commun. ACM* May 1994; **37**(5):72–82, doi:10.1145/175290.175300. URL http://doi.acm.org/10.1145/175290.175300.
3. Dekhtyar A, Dekhtyar O, Holden J, Hayes JH, Cuddeback D, Kong WK. On human analyst performance in assisted requirements tracing: Statistical analysis. *Requirements Engineering Conference (RE), 2011 19th IEEE International*, IEEE, 2011; 111–120.
4. Hayes JH, Dekhtyar A. A framework for comparing requirements tracing experiments. *International Journal of Software Engineering and Knowledge Engineering* 2005; **15**(05):751–781.
5. Hayes JH, Dekhtyar A, Osborne J. . improving requirements tracing via information retrieval. *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, IEEE, 2003; 138–147.
6. Hayes JH, Dekhtyar A, Sundaram SK. Advancing candidate link generation for requirements tracing: The study of methods. *Software Engineering, IEEE Transactions on* 2006; **32**(1):4–19.
7. Cuddeback D, Dekhtyar A, Hayes J. Automated requirements traceability: The study of human analysts. *2014 IEEE 22nd International Requirements Engineering Conference (RE)* 2010; **0**:231–240, doi:http://doi.ieeecomputersociety.org/10.1109/RE.2010.35.
8. Dekhtyar A, Dekhtyar O, Holden J, Hayes JH, Cuddeback D, Kong W. On human analyst performance in assisted requirements tracing: Statistical analysis. *RE 2011, 19th IEEE International Requirements Engineering Conference, Trento, Italy, August 29 2011 - September 2, 2011*, 2011; 111–120, doi:10.1109/RE.2011.6051649. URL http://dx.doi.org/10.1109/RE.2011.6051649.
9. Kong W, Hayes JH, Dekhtyar A, Holden J. How do we trace requirements: an initial study of analyst behavior in trace validation tasks. *Proceedings of the 4th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2011, Waikiki, Honolulu , HI, USA, May 21, 2011*, 2011; 32–39, doi:10.1145/1984642.1984648. URL http://doi.acm.org/10.1145/1984642.1984648.
10. Kong W, Hayes JH, Dekhtyar A, Dekhtyar O. Process improvement for traceability: A study of human fallibility. *2012 20th IEEE International Requirements Engineering Conference (RE), Chicago, IL, USA, September 24-28, 2012*, 2012; 31–40, doi:10.1109/RE.2012.6345824. URL http://dx.doi.org/10.1109/RE.2012.6345824.
11. Basili V. Lecture of professor victor r. basili for the confirmation of the laurea honoris causa in ingeneria informatica at the university of sannio ; URL http://www.compaid.com/caiinternet/ezine/basilitalk.pdf.
12. Shaw M. What makes good research in software engineering? *STTT* 2002; **4**(1):1–7, doi:10.1007/s10009-002-0083-4. URL http://dx.doi.org/10.1007/s10009-002-0083-4.

13. Li W, Hayes JH, Truszczyński M. Temporal action language (tal): A controlled language for consistency checking of natural language temporal requirements. *NASA Formal Methods*. Springer, 2012; 162–167.
14. Hayes JH, Guéhéneuc YG, Li W, Truszczynski M, Antoniol G. Error leakage and wasted time: Sensitivity analysis of a requirements consistency checking process. *North American Search Based Software Engineering Symposium (NasBASE)*, 2015.
15. Le TDB, Lo D. Will fault localization work for these failures? an automated approach to predict effectiveness of fault localization tools. *Software Maintenance (ICSM), 2013 29th IEEE International Conference on*, IEEE, 2013; 310–319.
16. Ramasubbu N, Balan RK. Overcoming the challenges in cost estimation for distributed software projects. *Proceedings of the 34th International Conference on Software Engineering*, IEEE Press, 2012; 91–101.
17. Surian D, Liu N, Lo D, Tong H, Lim EP, Faloutsos C. Recommending people in developers' collaboration network. *Reverse Engineering (WCRE), 2011 18th Working Conference on*, IEEE, 2011; 379–388.
18. Antoniol G, Ayari K, Di Penta M, Khomh F, Guéhéneuc YG. Is it a bug or an enhancement?: A text-based approach to classify change requests. *Proceedings of the 2008 conference of the center for advanced studies on collaborative research: meeting of minds*, ACM, 2008; 23.
19. Herzig K, Just S, Zeller A. It's not a bug, it's a feature: How misclassification impacts bug prediction. *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, 2013; 392–401.
20. Bird C, Bachmann A, Aune E, Duffy J, Bernstein A, Filkov V, Devanbu P. Fair and balanced?: Bias in bug-fix datasets. *Proceedings of the the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, ACM, 2009; 121–130.
21. Bachmann A, Bird C, Rahman F, Devanbu P, Bernstein A. The missing links: Bugs and bug-fix commits. *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering*, ACM, 2010; 97–106.
22. Herzig K, Zeller A. The impact of tangled code changes. *Mining Software Repositories (MSR), 2013 10th IEEE Working Conference on*, IEEE, 2013; 121–130.
23. Bird C, Rigby PC, Barr ET, Hamilton DJ, German DM, Devanbu P. The promises and perils of mining git. *Mining Software Repositories, 2009. MSR'09. 6th IEEE International Working Conference on*, IEEE, 2009; 1–10.
24. Kalliamvakou E, Gousios G, Blincoe K, Singer L, German DM, Damian D. The promises and perils of mining github. *Proceedings of the 11th Working Conference on Mining Software Repositories*, ACM, 2014; 92–101.
25. Rahman F, Posnett D, Herraiz I, Devanbu P. Sample size vs. bias in defect prediction. *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ACM, 2013; 147–157.
26. Dit B, Guerrouj L, Poshyvanyk D, Antoniol G. Can better identifier splitting techniques help feature location? *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*, IEEE, 2011; 11–20.
27. Binkley D, Lawrie D. The impact of vocabulary normalization. *Journal of Software: Evolution and Process* 2015; **27**(4):255–273.
28. Binkley D, Lawrie D, Uehlinger C, Heinz D. Enabling improved ir-based feature location. *Journal of Systems and Software* 2015; **101**(C):30–42.
29. Haiduc S, Bavota G, Marcus A, Oliveto R, De Lucia A, Menzies T. Automatic query reformulations for text retrieval in software engineering. *Proceedings of the 2013 International Conference on Software Engineering*, IEEE Press, 2013; 842–851.
30. Harman M, Mansouri SA, Zhang Y. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)* 2012; **45**(1):11.
31. Baker P, Harman M, Steinhöfel K, Skaliotis A. Search based approaches to component selection and prioritization for the next release problem. *Software Maintenance, 2006. ICSM'06. 22nd IEEE International Conference on*, IEEE, 2006; 176–185.
32. McMinn P. Search-based software test data generation: A survey. *Software testing, Verification and reliability* 2004; **14**(2):105–156.
33. Di Penta M, Harman M, Antoniol G. The use of search-based optimization techniques to schedule and staff software projects: An approach and an empirical study. *Software: Practice and Experience* 2011; **41**(5):495–519.
34. O'Keeffe M, Cinnéide MÓ. Search-based refactoring: An empirical study. *Journal of Software Maintenance and Evolution: Research and Practice* 2008; **20**(5):345–364.
35. Canfora G, Di Penta M, Esposito R, Villani ML. Qos-aware replanning of composite web services. *Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on*, IEEE, 2005; 121–129.
36. Panichella A, Dit B, Oliveto R, Di Penta M, Poshyvanyk D, De Lucia A. Parameterizing and assembling ir-based solutions for se tasks using genetic algorithms. *Proceedings of the 23rd IEEE International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 2016.
37. Hurtado Alegría JA, Bastarrica MC, Bergel A. Analyzing software process models with avispa. *Proceedings of the 2011 International Conference on Software and Systems Process*, ACM, 2011; 23–32.
38. Iwata K, Nakashima T, Anan Y, Ishii N. Errors estimation models for embedded software development projects. *Software Engineering Research, Management & Applications, 2007. SERA 2007. 5th ACIS International Conference on*, IEEE, 2007; 815–822.
39. Pandey D, Suman U, Ramani A. An effective requirement engineering process model for software development and requirements management. *Advances in Recent Technologies in Communication and Computing (ARTCom), 2010 International Conference on*, IEEE, 2010; 287–291.
40. Krusche S, Alperowitz L, Bruegge B, Wagner MO. Rugby: an agile process model based on continuous delivery. *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, ACM, 2014; 42–50.
41. Sadiq M, Hassan T. An extended adaptive software development process model. *Issues and Challenges in Intelligent Computing Techniques (ICICT), 2014 International Conference on*, IEEE, 2014; 552–558.
42. Huth M, Ryan M. . *Logic in Computer Science: Modelling and Reasoning about Systems*. Cambridge University Press, 2004.

43. Marek VW, Truszczyński M. Stable models and an alternative logic programming paradigm. *The Logic Programming Paradigm*. Springer, 1999; 375–398.
44. Niemelä I. Logic programs with stable model semantics as a constraint programming paradigm. *Annals of Mathematics and Artificial Intelligence* 1999; **25**(3-4):241–273.
45. Doherty P, Gustafsson J, Karlsson L, Kvarnström J. Tal: Temporal action logics language specification and tutorial. *Computer and Information Science* 1998; **3**(015).
46. Hayes JH, Dekhtyar A, Sundaram SK, Holbrook EA, Vadlamudi S, April A. Requirements tracing on target (retro): Improving software maintenance through traceability recovery. *Innovations in Systems and Software Engineering* 2007; **3**(3):193–202.
47. Klein D, Manning CD. Accurate unlexicalized parsing. *Proceedings of the 41st Annual Meeting on Association for Computational Linguistics-Volume 1*, Association for Computational Linguistics, 2003; 423–430.
48. De Marneffe MC, MacCartney B, Manning CD, *et al.*. Generating typed dependency parses from phrase structure parses. *Proceedings of LREC*, vol. 6, 2006; 449–454.
49. Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P. Natural language processing (almost) from scratch. *The Journal of Machine Learning Research* 2011; **12**:2493–2537.
50. Kong WK, Hayes JH, Dekhtyar A, Dekhtyar O. Process improvement for traceability: A study of human fallibility. *Requirements Engineering Conference (RE), 2012 20th IEEE International*, IEEE, 2012; 31–40.
51. Wagner S. Global sensitivity analysis of predictor models in software engineering. *Predictor Models in Software Engineering, 2007. PROMISE'07: ICSE Workshops 2007. International Workshop on*, IEEE, 2007; 3–3.
52. Marcus A, Maletic J, *et al.*. Recovering documentation-to-source-code traceability links using latent semantic indexing. *Software Engineering, 2003. Proceedings. 25th International Conference on*, IEEE, 2003; 125–135.
53. Antoniol G, Canfora G, Casazza G, De Lucia A, Merlo E. Recovering traceability links between code and documentation. *Software Engineering, IEEE Transactions on* 2002; **28**(10):970–983.
54. Oliveto R, Gethers M, Poshyvanyk D, De Lucia A. On the equivalence of information retrieval methods for automated traceability link recovery. *Program Comprehension (ICPC), 2010 IEEE 18th International Conference on*, IEEE, 2010; 68–71.
55. Baldridge J. The opennlp project 2005. URL http://opennlp.apache.org/index.html.
56. Bajwa IS, Lee M, Bordbar B. Resolving syntactic ambiguities in natural language specification of constraints. *Computational Linguistics and Intelligent Text Processing*. Springer, 2012; 178–187.
57. Chen KJ, Huang CR, Chang LP, Hsu HL. Sinica corpus: Design methodology for balanced corpora. *Language* 1996; **167**:176.
58. Liu SH, Chen KJ, Chang LP, Chin YH. Automatic part-of-speech tagging for chinese corpora. *Computer Processing of Chinese & Oriental Languages* 1995; **9**(1):31–47.
59. Lua K. Part of speech tagging of chinese sentences using genetic algorithm. *Proceedings of ICCC96, National University of Singapore* 1996; :45–49.
60. Tsai YF, Chen KJ. Reliable and cost-effective pos-tagging. *ROCLING*, 2003.
61. Thou´esny S. Increasing the reliability of a part-of-speech tagging tool for use with learner language. *Presentation given at the Automatic Analysis of Learner Language (AALL209) workshop on automatic analysis of learner language: from a better understanding of annotation needs to the development and standardization of annotation schemes*, 2009.
62. Rizzolo N, Roth D. Learning based java for rapid development of nlp systems. *LREC*, 2010.
63. UOW. URL http://www.uow.edu.au/~dlee/software.htm.
64. Regional real time transit information system system requirements version 3.0 2005. URL http://www.mtc.ca.gov/planning/tcip/Real-Time_TransitSystemRequirements_v3.0.
65. Li W, Brown D, Hayes JH, Truszczynski M. Answer-set programming in requirements engineering. *Requirements Engineering: Foundation for Software Quality*. Springer, 2014; 168–183.
66. Barnickel T, Weston J, Collobert R, Mewes HW, Stümpflen V. Large scale application of neural network based semantic role labeling for automated relation extraction from biomedical texts. *PLoS One* 2009; **4**(7):e6393.
67. Parse selection for self-training. URL http://depts.washington.edu/uwcl/twiki/pub/Main/JimWhite/Parse_Selection_for_Self-Training.pdf.
68. Trained models. URL http://nilc.icmc.usp.br/nlpnet/models.html.
69. USEPA. Risk assessment guidance for superfund (rags): Volume iii part a, process for conducting probabilistic risk assessment 2001. URL http://www.epa.gov/oswer/riskassessment/rags3adt/pdf/rags3adt_complete.pdf.
70. Steenbergen RDJM, van Gelder P, Miraglia S, Vrouwenvelder A. *Safety, reliability and risk analysis: Beyond the horizon*. CRC Press, 2013.
71. Saltelli A, Tarantola S, Campolongo F, Ratto M. *Sensitivity analysis in practice: a guide to assessing scientific models*. John Wiley & Sons, 2004.
72. Makowski D. Uncertainty and sensitivity analysis in quantitative pest risk assessments; practical rules for risk assessors. *NeoBiota* 2013; **18**:157.
73. Hei XJ, Cheung L. *Access Networks: 4th International Conference, AccessNets 2009, Hong Kong, China, November 1-3, 2009, Revised Selected Papers*, vol. 37. Springer, 2010.
74. Spanoudakis G, Zisman A, Prez-miana E, Krause P, Systems BPD. Rule-based generation of requirements traceability relations. *Journal of Systems and Software* 2004; **72**:105–127.
75. Dekhtyar A, Hayes JH, Larsen J. Make the most of your time: How should the analyst work with automated traceability tools? *Proceedings of the Third International Workshop on Predictor Models in Software Engineering*, PROMISE '07, IEEE Computer Society: Washington, DC, USA, 2007; 4–, doi:10.1109/PROMISE.2007.8. URL http://dx.doi.org/10.1109/PROMISE.2007.8.
76. El Emam K. Benchmarking kappa: Interrater agreement in software process assessments. *Empirical Software Engineering* 1999; **4**(2):113–133.

77. Dekhtyar A, Hayes JH, Larsen J. Make the most of your time: How should the analyst work with automated traceability tools? 2007; .
78. Dekhtyar A, Hayes JH, Smith M. Towards a model of analyst effort for traceability research. *Proceedings of the 6th International Workshop on Traceability in Emerging Forms of Software Engineering*, ACM, 2011; 58–62.