Total Recall, Language Processing, and Software Engineering

Zhe Yu
North Carolina State University
USA
zyu9@ncsu.edu

Tim Menzies
North Carolina State University
USA
timm@ieee.org

ABSTRACT

A broad class of software engineering problems can be generalized as the "total recall problem". This short paper claims that identifying and exploring the total recall problems in software engineering is an important task with wide applicability.

To make that case, we show that by applying and adapting the state of the art active learning and natural language processing algorithms for solving the total recall problem, two important software engineering tasks can also be addressed : (a) supporting large literature reviews and (b) identifying software security vulnerabilities. Furthermore, we conjecture that (c) test case prioritization and (d) static warning identification can also be generalized as and benefit from the total recall problem.

The widespread applicability of "total recall" to software engineering suggests that there exists some underlying framework that encompasses not just natural language processing, but a wide range of important software engineering tasks.

CCS CONCEPTS

• **Information systems** → **Learning to rank**; • **Security and privacy** → *Software security engineering*; • **Software and its engineering** → *Software development methods*; *Risk management*;

KEYWORDS

Software engineering, active learning, natural language processing, information retrieval, total recall, literature review, vulnerabilities

ACM Reference Format:
Zhe Yu and Tim Menzies. 2018. Total Recall, Language Processing, and Software Engineering. In *Proceedings of the 4th ACM SIGSOFT International Workshop on NLP for Software Engineering (NL4SE '18), November 4, 2018, Lake Buena Vista, FL, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3283812.3283818

1 INTRODUCTION

Software engineering is a discipline closely involved with human activities. How to help software developers produce better software more efficiently is the core topic of software engineering. Prioritizing tasks can efficiently reduce the human efforts and time required to achieve certain goals of software development. Many of such prioritization problems in software engineering can be generalized as the *total recall problem* (defined in §2).

The total recall problem has been explored in information retrieval for years, and the state of the art solution with active learning and natural language processing aims to resolve the following challenges:

- Among a finite number of tasks, which task to be executed first so that certain goals can be achieved earlier?
- At what point, there is no need to execute the remaining tasks?
- Are all the tasks executed correctly? How to identify wrongly executed tasks and correct them?
- How to scale up the process with multiple humans working in parallel (e.g. crowdsourcing)?

The first challenge has been extensively explored [1, 10, 14] while the rest three have much room to improve [2–4].

This short paper claims that identifying and exploring the total recall problems in software engineering is an important task which benefits both software engineering research and the general solution to total recall problems. To support this claim, in §2 we first introduce the general total recall problem, and its state of the art solutions; then in §3, we provide two software engineering tasks which are solved by applying total recall techniques and better solutions to the challenges of total recall problems were created during the process [16–18]. We also conjecture that two other software engineering tasks can be categorized as the total recall problem. The goal of this short paper is to inspire software engineering researchers to explore total recall problems in software engineering with our preliminary results.

2 THE TOTAL RECALL PROBLEM

The total recall problem in information retrieval aims to optimize the cost for achieving very high recall–as close as practicable to 100%–with a human assessor in the loop [6]. More specifically, the total recall problem can be described as following:

> **The Total Recall Problem:**
> Given a set of candidates E, in which only a small fraction $R \subset E$ are positive, each candidate $x \in E$ can be inspected to reveal its label as positive ($x \in R$) or negative ($x \notin R$) at a cost. Starting with the labeled set $L = \emptyset$, the task is to inspect and label as few candidates as possible ($\min |L|$) while achieving very high recall ($\max |L \cap R|/|R|$).

Different strategies have been applied to solve the total recall problem, including supervised learning and semi-supervised learning. However, the state of the art solutions to the total recall problem apply active learning [6] to learn from natural language processing features (e.g. bag-of-words, tf-idf) extracted from the current

NL4SE '18, November 4, 2018, Lake Buena Vista, FL, USA

ACM ISBN 978-1-4503-6055-5/18/11…$15.00
https://doi.org/10.1145/3283812.3283818

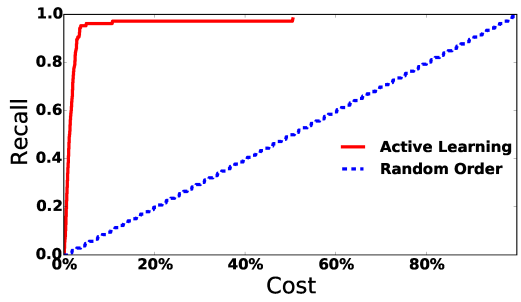

Figure 1: Power of active learning on total recall problems.

labeled set L and re-rank the rest of the candidates $E \setminus L$ so that candidates that are more likely to be positive get inspected next. This active learning solution has been proven effective in different use cases. A demonstration of the benefit of this active learning strategy can be found in Figure 1, where with active learning (solid red line), high recall (close to 100%) can be achieved by inspecting only a small portion of the candidates.

In evidence-based medicine, researchers screen titles and abstracts to determine whether one paper should be included in a certain systematic review. Wallace et al. [14] designed patient active learning to help researchers prioritize their effort on papers to be included. With patient active learning, half of the screening effort can be saved while still including most relevant papers [14].

In electronic discovery, attorneys are hired to review massive amount of documents looking for relevant ones for a certain legal case and provide those as evidences. Cormack and Grossman [1] proposed continuous active learning to save attorneys' effort from reviewing non-relevant documents, which further can save a large amount of the cost of legal cases.

The active learning strategy for total recall problem can be described as following:

Step 0 Given a candidate set E and initialize the labeled set $L = \emptyset$.
Step 1 Using strategies like ad hoc search or random sampling to select next example x to label ($L \leftarrow L \cup x$).
Step 2 Repeat **Step 1** until "enough" positive examples have been labeled ($|L \cap R| \geq K$).
Step 3 Train/update a supervised learning model with current labeled set L.
Step 4 Use the trained model to predict on the unlabeled set $E \setminus L$ and select next example x to label ($L \leftarrow L \cup x$).
Step 5 Repeat **Step 3** and **Step 4** until the stopping rule is met.

Where detail settings vary from case studies to case studies:

- In **Step 2**, Cormack and Grossman [1] believe learning should start as soon as possible ($K = 1$) while Wallace et al. [14] suggest to start learning until more training examples are available.
- In **Step 3**, most studies exact text features from examples and train a support vector machine with linear kernel. However, there are different opinions on how to balance the training data, e.g. Wallace et al. [14] proposed a technique called aggressive undersampling to drop off the negative examples closes to the positive ones while Miwa et al. [10] adjusted the weight of each training example to punish more for misclassifying positive examples.
- In **Step 4**, Cormack and Grossman [1] select examples that are most likely to be positive while Wallace et al. [14] select most uncertain examples.
- In **Step 5**, Cormack and Grossman [1] stop the process when a sufficient number of positive examples have been found while Wallace et al. [14] stop the learning when the model becomes stable and then apply the model to classify the unlabeled examples.

While this active learning and natural language processing strategy has been extensively explored to resolve the main challenge of total recall problems, there still exists large room for improvement in the three other challenges.

2.1 When to Stop

In practice, there is no way to know the number of positive examples before inspecting and labeling every candidate example. It is thus impossible to know exactly what recall has been reached during the process. Then, how do we know when to stop if the target is reaching, say 95% recall? This is a practical problem that directly affects the usability of the active learning strategy. Stopping too early will result in missing many valuable positive examples; while stopping too late will cause unnecessary cost when there are no more positive examples to retrieve. So far, researchers have developed various stopping rules to solve the "when to stop" problem.

- Ros et al. [12] developed the the most straightforward rule, which decides that the process should be stopped after 50 negative examples are found in succession.
- Cormack and Grossman proposed the knee method [2], which detects the inflection point i of current recall curve, and compare the slopes before and after i. If $slope_{<i}/slope_{>i}$ is greater than a specific threshold ρ, the review should be stopped.
- Wallace et al. [13] applied an estimator to estimate the number of positive examples $|R|$ and let the users decide when to stop by showing them how close they are to the estimated number.

2.2 Human Error Correction

When solving the total recall problems, the next example to be inspected relies on model trained on previously labeled examples. What if those labels can be wrong? The trained model could be misled by wrongly labeled examples and thus make worse decision on which example to inspect next. By now, researchers have applied various strategies to correct such human errors:

- One simple way to correct human errors is by majority vote [9], where every example will be inspected by two different humans, and when the two humans disagree with each other, a third human is asked to make the final decision.
- Cormack and Grossman [4] built an error correction method upon the knee stopping rule [2]. After the review stops, examples labeled as positive but are inspected after the inflection point ($x > i$) and examples labeled as negative but inspected before the inflection point ($x < i$) are sent to humans for a recheck.

2.3 Scalability

How to scale up the active learning framework with multiple humans working simultaneously on the same project remains an open challenge to the total recall problem. In electronic discovery, Cormack and Grossman [3] proposed S-CAL where model trained on a finite number of examples is used to predict and estimate the precision, recall, and prevalence of the target large set of examples. However, this approach sacrifices the adaptability of the active learner. In evidence-based medicine, Nguyen et al. [11] explored the task allocation problem where two types of human operators are available: (1) crowdsourcing workers who are cheaper but less accurate, and (2) experts who are much more expensive but also more accurate. This approach allows the system to operate more economically but still cannot scale up with growing number of training examples.

3 TOTAL RECALL PROBLEMS IN SOFTWARE ENGINEERING

As discussed in §1, there are tasks in software engineering that resemble the problem of total recall. In this section, we will discuss four such problems. The first two tasks have been explored by the authors previously and the promising results convince us that applying active learning and natural language processing strategy is the key to solving these software engineering problems.

3.1 Primary Study Selection

For the same reason of medicine researchers, software engineering researchers also need to read literature to stay current on their research topics. Researchers conduct systematic literature reviews [8] to analyze the existing literature and to facilitate other researchers. Among different steps of systematic literature reviews, primary study selection is in exactly the same format as citation screening in medical systematic reviews. The problem is also for reading titles and abstracts to find relevant papers to include, except that those papers are about software engineering. As a result, the primary study selection problem can be described as a total recall problem:

- E: set of papers returned from a search query.
- R: set of relevant papers to the systematic literature review.
- L: set of papers that have been reviewed and labeled as relevant or non-relevant by human researchers.

While exploring this total recall problem [16, 17], the authors:

(1) Designed a new active learning framework by combining advantages of the state of the art approaches [1, 10, 14] in §2. When reaching the same recall, the new framework costs 20-50% less than these prior state of the art approaches.
(2) Created an estimator for $|R|$ based on semi-supervised learning for the *when to stop* challenge. This estimator showed better accuracy than the one from Wallace et al. [13] and provided better stopping rules than the prior methods in §2.1. With the help of this estimator, the users can know what recall has been reached and make better decision on whether to settle with current recall or spend more effort to achieve a higher recall.
(3) Proposed an error identification method, in which only examples with labels that the current active learner disagrees most on are rechecked by a different human expert. This method has been proven to be more cost effective in addressing the *human error correction* challenge than prior methods in §2.2.

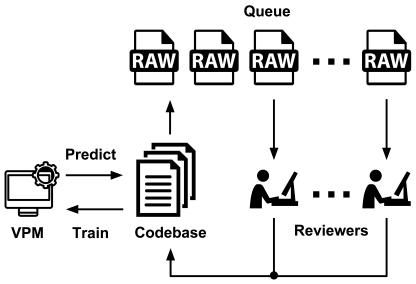

Figure 2: Poactive Security Review and Test Framework

Datasets and tools for reproduction and making further improvements on the primary study selection problem can be found at Seacraft, Zenodo[1] and Github[2].

3.2 Software Security Vulnerability Prediction

Society needs more secure software. It is crucial to identify software security vulnerabilities and fix them as early as possible. However, it is time-consuming to have human experts inspecting the entire codebase looking for the few source code files that contain vulnerabilities. The solution to this problem, prioritizing the human inspection effort towards codes that are more likely to have vulnerabilities is also a total recall problem:

- E: the entire codebase of a software project.
- R: set of source code files that contain vulnerabilities.
- L: set of source code files already inspected by humans.

With simulations on the known vulnerabilities from Mozilla Firefox project [18], the authors:

(1) Extracted bag-of-words from source codes and applied same active learning algorithm as the primary study selection problem [16, 17] to select which source code file to inspect next and to stop the inspection when target recall is reached. Results showed that 90, 95, 99% recall can be achieved by having human experts inspect 20, 26, 46% of the source code files, respectively [18]. Results also showed that text features perform better than traditional software metrics features.
(2) Adapted the error correction method from primary study selection [17] to identify missing vulnerabilities[3]. Results showed that it can reach higher recall with lower cost comparing to other error correction methods mentioned in §2.2.
(3) Designed a centralized system where human operators can work in parallel, as shown in Figure 2.

The real benefit of treating vulnerability prediction as a total recall problem is that no labeled data (known vulnerabilities) are required to start the process, thus these vulnerabilities can be identified and fixed before the software is deployed. On contrast, conventional methods such as supervised learning or unsupervised learning

[1] https://doi.org/10.5281/zenodo.1162952
[2] https://github.com/fastread/src
[3] When inspecting codes, human may miss some vulnerabilities (false negatives) but may never have false positives, and the false negative rate can be as high as 47% [7].

require post-deployment information such as known vulnerabilities from bug reports or crash dump stack traces.

3.3 Static Warning Identification

Static Analysis tools (e.g., FindBugs) are widely used to find defects in software. These tools scan source code or binary code of a software project and infer bugs, security vulnerabilities, and bad programming practices with heuristic pattern matching techniques [15]. Problem is, the high false positive rate of the reported warnings causes most of the warnings not acted on by developers [15]. Reducing such false positives is a total recall problem:

- E: all warnings reported by the static analysis tools.
- R: set of warnings that reveal true defects.
- L: set of warnings that have been inspected by human experts.

This static warning identification problem has been explored with supervised learning methods and various features [15]. Analyzing the warnings with natural language processing and applying active learning to select which warning to inspect might help reduce false positives and make static analysis tools more practical to use.

3.4 Test Case Prioritization

Regression testing is an expensive testing process to detect whether new faults have been introduced into previously tested code. To reduce the cost of regression testing, software testers may prioritize their test cases so that those which are more likely to fail are run earlier in the regression testing process [5]. The goal of test case prioritization is to increase the rate of fault detection, and by doing so, it can provide earlier feedback on the system under test, enable earlier debugging, and increase the likelihood that, if testing is prematurely halted, those test cases that offer the greatest fault detection ability in the available testing time will have been executed [5]. This test case prioritization problem can also be treated as a total recall problem with the following notations:

- E: all candidate regression test suites.
- R: set of test suites that will detect faults.
- L: set of test suites that have been executed.

Existing techniques for prioritizing test cases are "unsupervised", i.e. these techniques decide an order of the test cases to be run and stick with it. Applying active learning to adapt the ordering with knowledge learned from test cases executed can potentially further increase the rate of fault detection and reduce more cost. However, this problem has not been explored as a total recall problem and the following challenges need to be resolved:

- What types of feature can be extracted from the test cases so that the learned model can accurately predict the likelihood of fault detection from a test case before its execution.
- How to balance the priority of test cases increasing coverage of statements/functions most and the more-likely-to-fail test cases.
- How to handle test dependence. If test cases are not independent, changing their order may fail some tests but pass others [19].

4 CONCLUSIONS AND FUTURE WORK

Many of the software engineering problems can be generalized as the total recall problem. This paper identified four such problems. With two case studies of primary study selection and vulnerability prediction, we showed that exploring these total recall problems in software engineering would benefit both software engineering research and the general solution to total recall problems. We hope this paper will attract more researchers studying and improving the total recall problems in software engineering. Future work in this area includes but is not limited to the follows:

- Improve core algorithm for the total recall problem—reaching same recall with less cost.
- Build more accurate estimator for current recall achieved.
- Resolve human errors more efficiently.
- Scale up the total recall solutions (probably by ensemble learning) and utilize low-cost workers through crowdsourcing.
- Apply total recall techniques to reduce false alarms in static code analysis and to prioritize test cases.
- Identify more total recall problems in software engineering.

REFERENCES

[1] Gordon V Cormack and Maura R Grossman. 2014. Evaluation of machine-learning protocols for technology-assisted review in electronic discovery. In *Proceedings of the 37th international ACM SIGIR conference on Research & development in information retrieval*. ACM, 153–162.

[2] Gordon V Cormack and Maura R Grossman. 2016. Engineering Quality and Reliability in Technology-Assisted Review. (2016), 75–84.

[3] Gordon V Cormack and Maura R Grossman. 2016. Scalability of Continuous Active Learning for Reliable High-Recall Text Classification. In *Proceedings of CIKM 2016*. ACM, 1039–1048.

[4] Gordon V Cormack and Maura R Grossman. 2017. Navigating Imprecision in Relevance Assessments on the Road to Total Recall: Roger and Me. In *The International ACM SIGIR Conference*. 5–14.

[5] Sebastian Elbaum, Alexey G Malishevsky, and Gregg Rothermel. 2002. Test case prioritization: A family of empirical studies. *IEEE transactions on software engineering* 28, 2 (2002), 159–182.

[6] Maura R Grossman, Gordon V Cormack, and Adam Roegiest. 2016. TREC 2016 Total Recall Track Overview. In *TREC*.

[7] Les Hatton. 2008. Testing the value of checklists in code inspections. *IEEE software* 25, 4 (2008).

[8] Barbara A Kitchenham, Tore Dyba, and Magne Jorgensen. 2004. Evidence-based software engineering. In *Proceedings of the 26th international conference on software engineering*. IEEE Computer Society, 273–281.

[9] Marco Kuhrmann, Daniel Méndez Fernández, and Maya Daneva. 2017. On the pragmatic design of literature studies in software engineering: an experience-based guideline. *Empirical Software Engineering* 22, 6 (2017), 2852–2891.

[10] Makoto Miwa, James Thomas, Alison O'Mara-Eves, and Sophia Ananiadou. 2014. Reducing systematic review workload through certainty-based screening. *Journal of biomedical informatics* 51 (2014), 242–253.

[11] An Thanh Nguyen, Byron C Wallace, and Matthew Lease. 2015. Combining crowd and expert labels using decision theoretic active learning. In *Third AAAI Conference on Human Computation and Crowdsourcing*.

[12] Rasmus Ros, Elizabeth Bjarnason, and Per Runeson. 2017. A Machine Learning Approach for Semi-Automated Search and Selection in Literature Studies. In *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering*. ACM, 118–127.

[13] Byron C Wallace, Issa J Dahabreh, Kelly H Moran, Carla E Brodley, and Thomas A Trikalinos. 2013. Active literature discovery for scoping evidence reviews: How many needles are there. In *Proceedings of KDD-DMH*.

[14] Byron C Wallace, Thomas A Trikalinos, Joseph Lau, Carla Brodley, and Christopher H Schmid. 2010. Semi-automated screening of biomedical citations for systematic reviews. *BMC bioinformatics* 11, 1 (2010), 1.

[15] Junjie Wang, Song Wang, and Qing Wang. 2018. Is There A "Golden" Feature Set for Static Warning Identification?. In *Proceedings of the ESEM 2018*.

[16] Zhe Yu, Nicholas A. Kraft, and Tim Menzies. 2018. Finding better active learners for faster literature reviews. *Empirical Software Engineering* (07 Mar 2018).

[17] Zhe Yu and Tim Menzies. 2017. FAST2: Better Automated Support for Finding Relevant SE Research Papers. (2017). http://arxiv.org/abs/1705.05420

[18] Zhe Yu, Christopher Theisen, Hyunwoo Sohn, Laurie Williams, and Tim Menzies. 2018. Cost-aware Vulnerability Prediction: the HARMLESS Approach. *CoRR* abs/1803.06545 (2018). arXiv:1803.06545 http://arxiv.org/abs/1803.06545

[19] Sai Zhang, Darioush Jalali, Jochen Wuttke, Kıvanç Muşlu, Wing Lam, Michael D Ernst, and David Notkin. 2014. Empirically revisiting the test independence assumption. In *Proceedings of the 2014 International Symposium on Software Testing and Analysis*. ACM, 385–396.