BUDGET: a Tool for Supporting Software Architecture Traceability Research

Joanna C. S. Santos, Mehdi Mirakhorli, Ibrahim Mujhid and Waleed Zogaan
Software Engineering Department
Rochester Institute of Technology
Rochester, NY, USA
{jds5109, mxmvse, ijm9654, waz7355}@rit.edu

***Abstract*—Automated traceability techniques based on supervised machine learning algorithms can significantly reduce the cost and effort needed to create and maintain traceability links between requirements, architecture and source code. However, the upfront cost to train these algorithms is the main bottleneck for expanding, and validating these traceability techniques as well as applying them to complex industrial systems. In this tool demo, we present our web-based tool named BUDGET, as a solution to automate creation of training data for the problem of tracing architectural concerns. BUDGET uses Automated Web-Mining, and Big-Data Analysis techniques to generate training data for supervised architecture-traceability techniques. It uses several sampling strategies and mines ultra-large scale code repositories to generate datasets of tactical code snippets. The BUDGET falls in the research tool category and supports researchers in the area of software architecture and requirements engineering.**

I. Introduction

Automated traceability techniques are more increasingly being used by the industry to support developers during software engineering activities, such as compliance checking, verification and validation, acceptance testing and architectural analysis. The automated traceability techniques utilize advanced data mining methods, to automatically trace and connect different software artifacts which are conceptually similar. For example, a developer in healthcare domain can use these techniques to identify which parts of the system satisfy the regulatory codes defined by HIPAA [1] and which parts do not. Another user can use traceability techniques to trace quality attributes such as performance, reliability and availability into architectural tactics/patterns [2] used to implement them and the realization of these tactics/patterns in the source code.

The current state of traceability research relies on supervised or unsupervised techniques for developing tools to trace different types of artifacts. For example, supervised learning algorithms are used for tracing requirements, design decisions or coding concepts that "reoccur" across many systems [3], [6]. Unsupervised learning techniques, such as Vector Space Model (VSM) [8], have been also shown useful for tracing ad-hoc requirements, or in situations where it is difficult to collect sample reoccurring concepts in software artifacts of several systems.

An example of a supervised traceability technique is a classification algorithm [6], [7] we previously developed to detect and trace architectural choices known as *tactics* in the source code of a software system. Tactics are the building blocks of a software architecture and are used to satisfy a specific quality attribute [2]. For example, the *heartbeat* tactic can be used to satisfy an availability concern, while the *audit trail* can be used for satisfying security requirements. Our supervised tactic traceability technique was able to detect several architectural tactics such as *heartbeat*, *scheduling*, *authentication*, *audit trail* and *thread pooling* in source code. This approach was widely used by the traceability community to trace quality requirements through architectural tactics to the source code, perform change impact analysis at the architecture level and prevent architectural issues, such as drift and degradation.

The tactic classifier [5], [6] was trained using manually collected code snippets of architectural tactics from 10 open source projects. Even though the results were promising, we observed that the main challenge in extending this approach to a wider range of architectural tactics is the need to perform an extensive work of *collecting* and *peer-reviewing* training samples. This is a challenge that imposes limitations in applying the technique in an industrial setting. Furthermore, such limitation has prevented traceability researchers from actively extending the current tactic traceability research, or conduct comparative empirical studies in this area.

Problem: Implementation samples of architectural tactics/-patterns can significantly support research in the areas of software architecture. Such datasets will help researchers reason about tactic implementations, discover patterns within these implementations, formulate new design flaws as they relate to tactics/patterns and also develop novel code based architecture analysis techniques. However, creating such datasets requires months of manual effort.

Solution: In this demo paper, we developed BUDGET (Bigdata aUgmented Dataset GEnerator Tool), an on-line tool for automatically generating tactic-related training sets for researchers. BUDGET uses the description of tactics in the text books and queries online technical libraries (e.g. MSDN) or an internet-scale software repository to collect several implementation and documentation of the tactics. BUDGET's underlying search algorithms utilize advanced indexing and performance optimization techniques to query more than 22 million source files of our repository within few seconds. In particular, the current version of the BUDGET is designed

to reduce the upfront costs associated with creating training sets for supervised and semi-unsupervised software traceability approaches. BUDGET is delivered as an online tool[1].

II. Overview

BUDGET tool provides several features to support our goal of dataset generation. Examples of such key features are:

- An **internet-scale source code repository** of open source systems retrieved from GitHub, SourceForge, Apache and Google Code. This repository is used as the knowledge base to extract sample implementation of the tactics.
- An automated **Web-mining engine** to generate datasets of tactics implementations/specifications through mining technical programming libraries such as MSDN. This engine uses Google Search API (Application Programming Interface) to find web pages discussing frameworks and technical specifications of each tactic.
- An Automated **Big-data analysis engine** which generates datasets of tactical code snippets through mining our ultra large-scale repository of open source systems. BUDGET utilizes massive replication and indexing to improve the response time of this process.
- Different **data-sampling strategies.** BUDGET implements both stratified and random sampling techniques. This enables the researchers to either target specific projects or downsample the repository according to their needs. It also provides features to specify the size and balanced/unbalanced properties of the requested datasets.
- **project filtering features** to obtain the datasets from a set of repositories, or in a specific programming language.

Figure 1 shows the architecture of the BUDGET and its constituent elements. The envisioned stakeholders, the detailed features, and examples of architectural tactics supported by the BUDGET are discussed in the following subsections.

A. Stakeholders

The targeted users of the BUDGET tool are researchers in the area of software system traceability. However, it can be used by researchers in other areas of software engineering such as mining software repositories, software architecture and program analysis. Furthermore, students and educators can use this online tool to retrieve samples of architectural tactics implemented in several open source projects.

B. Web-Mining Approach

Web-based libraries contain a rich knowledge base related to software development, such as API documentations, tutorials, sample code, design concerns and so forth. Web content has previously been used by researchers to generate or augment research datasets [4]. In our paper, we use Web-mining approach to automatically generate a dataset of technical discussions about an architectural tactic. Studies have shown that such datasets are useful in developing and training feature detection techniques [5], [7]. Our approach uses the public knowledge in the technical libraries to generate high quality training sets for a supervised tactic traceability technique.

To do so, the *Web Mining Agent* (Fig. 1) uses the Google Search API and queries technical libraries using a predefined set of tactic-related terms. These terms were collected from the explanations of tactics within textbooks.

For instance, for *Heartbeat* we extracted the following keywords from its description: *"**Heartbeat** is a **fault detection** mechanism that employs a periodic message exchange between a system **monitor** and a process being monitored"* [2]. Then, to perform the search targeting a specific technical library (e.g MSDN), we construct the following search query: `Heartbeat OR fault OR detection OR monitoring site: https://msdn.microsoft.com/en-us/library`. This search query is forwarded to the Google Search API which performs a search over all Web pages within the MSDN library. Previous work [7] shows that textual description of tactics will form good search queries and can result in high quality results.

After retrieving the results, Web Mining Agent ranks the web-pages returned in the search based on relevance to a tactic. The positive samples are the web-pages with the highest similarity to a tactic, and negative samples are the pages with the similarity score of zero. For each retrieved web-page, the agent removes the HTML (HyperText Markup Language) tags and saves the results in a plain text file. This way, only the textual content of the web-pages are saved. In case of negative samples, the search query is modified to return only those web-pages that does not contain any of the tactical-related terms. These negative samples helps a supervised technique to remove dominant words in the web-pages of the library (e.g. the word "Microsoft" in MSDN library) [5].

C. Big-Data Analysis Approach

While the web-mining approach extracts the API specifications of the tactics from technical libraries, the Big-data analysis approach retrieves the actual sample code snippets from open source systems. In order to support generation of tactic-datasets which are diverse, cover several application domains and programming languages, BUDGET utilizes a massively large code base repository. This code repository contains over 116,609 projects from GitHub, Google Code, SourceForge, Apache, and other software repositories. BUDGET has different code crawling applications to extract projects from all these different code repositories. To extract the projects from Github, it uses a torrent system known as GHTorrent that acts as a service to extract data and events.

BUDGET also utilized Sourcerer, an automated crawling, parsing, and fingerprinting application developed by researchers at the University of California, Irvine [9]. It was used to extract projects from publicly available open source repositories such as Apache, Java.net, Google Code and Sourceforge. After projects extraction from these repositories, we performed a data cleaning in which we removed all the empty or very small projects.

[1] http://design.se.rit.edu/budget/

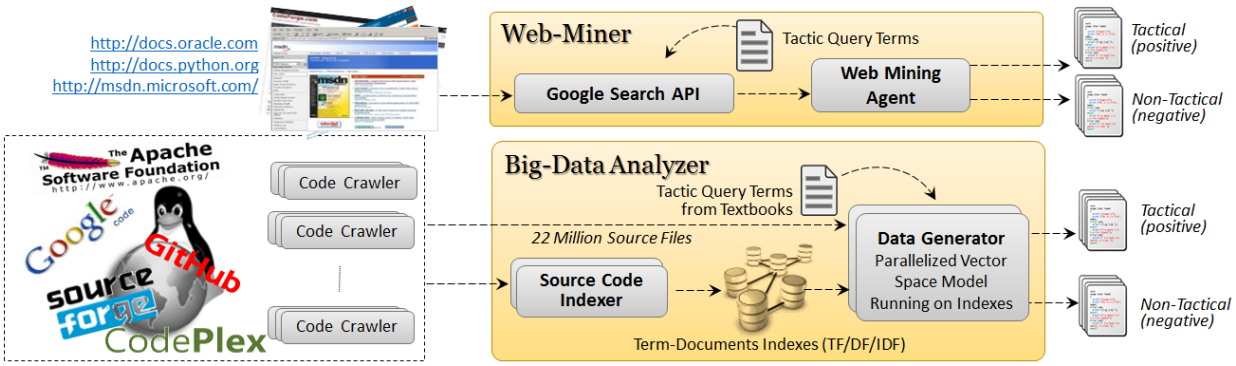


Fig. 1. Overview of BUDGET's data generation process.

Given the high amount of data being stored, we replicated all the indexing and searching processes. Several document indexing techniques were used to guarantee faster response time when searching for tactic-related code snippets. To perform the indexing, we preprocessed each source code by removing stop words and stemming the terms into their root form. Then we created an index that for each source file it stores: *term frequency (TF)*, *document frequency (DF)*, *TF/IDF*[2] value and the *location of source file*. This is an inverted index in which we can obtain a list of documents that contain a specific term.

The *Data Generator* component implements a paralleled version of Vector Space Model (VSM) [8] capable of running over 22 million source files in a few seconds. The VSM is a standard approach which computes the cosine similarity between a query and a document, each of which is represented as a vector of weighted terms. A more complete explanation is provided in most introductory information retrieval textbooks [8]. This component is used to generate a tactical dataset based on the search query extracted from text books. It calculates the cosine similarity score between provided query and all source files in our code repository.

For each tactic, the most relevant source files exhibiting highest similarity to the trace query are selected. For generating negative samples, this component also retrieves n samples of non-tactical files for each tactic from the same project (n is defined by the user). Previously it has been proven that unrelated sample data has significant impact on the accuracy of a classifier trained by both positive and negative samples [5].

D. User Interfaces (UIs) and Features

BUDGET user interfaces are shown in Figure 2. The initial UI on the web portal of BUDGET (Figure 2(a)) enables a user to specify the properties of the dataset and select an approach to generate the data. These properties are specified through configuration parameters of the tool. As shown in sub-figure 2(a), the parameters that shall be specified by the user before generating datasets are: the *architectural tactic*, the *dataset size* and the *approach* (Web Mining or Big-data analysis). Besides these parameters, BUDGET has additional ones specific to the data generation approach selected by the user. These further parameters provide flexibility to the user to adjust the generated data according to their research needs.

When using Web Mining technique, the user can also specify which Web sites to search for tactic-related content (Figure 2(c)). The default value of this parameter is a list of links that encompasses the MSDN, Oracle and the Python documentation.

In case of using Big-Data Analysis, BUDGET provides parameters for indicating the programming languages of the generated samples and a sampling strategy (Figure 2(b)) which defines the way the tool inspects the tactic-related code snippets from our ultra-large scale local repository. The three possible sampling strategies are *Best Cases*, *Random Sampling* and *Stratified Sampling*. Further description of these techniques can be found on the demo page.

After selecting the sampling strategy, the sampled tactical files are sorted based on the similarity score to the tactic query. Subsequently, the tool generates the N positive and M negative samples defined by the user. For that, the tool selects the N most similar tactical files and the M least related files for generating the positive and negative samples, respectively. Currently, BUDGET supports the generation of a dataset with up to 1000 positive/negative samples. Besides using the default text book terms in the Big-Data Analysis and Web Mining approaches, the tool has the flexibility of using user-defined terms to generate datasets. For that, the user selects *"Other"* as a tactic of interest and provide at least three terms as a list of keywords separated by commas.

The datasets generated by BUDGET are available as compressed file in ZIP format which can be downloaded. This ZIP file will organize the samples in different directories based on the sample type (positive or negative) and the source of the sample (see Figure 2(d)).

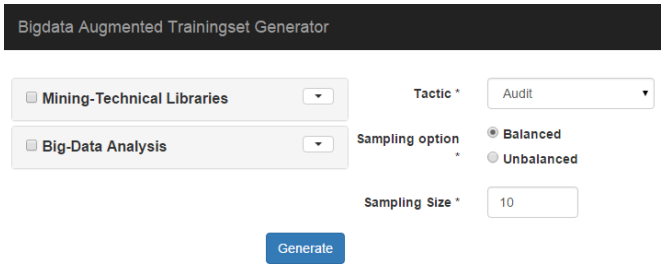
E. Tactics Supported by BUDGET

Current version of BUDGET has hard-coded queries for ten tactics such as *heartbeat*, *ping-echo scheduling*, *resource pooling*, *checkpointing*, *kerberos*. The user can add a search query for any additional tactic.

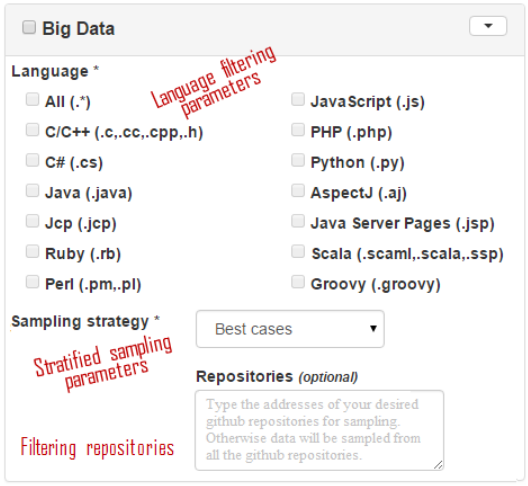
III. Evaluation

The accuracy of the datasets generated by BUDGET was evaluated by two members of our team. We generated a

[2]IDF:inverse document frequency

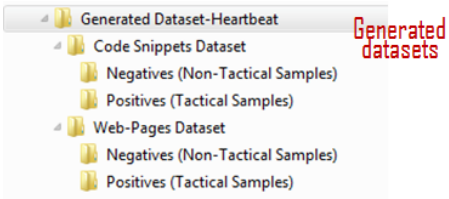


(a) BUDGET's initial page with the generic dataset generation parameters.



(b) Generating data using big-data analysis approach.

(c) Mining technical programming libraries.



(d) Generated dataset by BUDGET.

Fig. 2. Snapshot of BUDGET UIs

balanced dataset (size equals to 10) using both Web-mining and Big-data analysis approaches. In case of Big-data analysis approach, we used the best-case sampling strategy and source files in any programming languages were included in the sampling

The percentage of correctly generated data points for each tactical dataset is reported in the Table I. The Big-data analysis approach relatively results more accurate datasets. The automated support reduces the manual effort for obtaining such dataset. Even in case of the manual vetting of the automatically generated dataset, the total effort will be significantly less that purely manual approach. For instance, it took us about 3 months to collect and peer review tactical data from 10 different projects for 5 architectural tactics.

TABLE I
QUALITY OF AUTOMATICALLY GENERATED TRAINING-SET

Tactic	Web-Mining	Big-data
Audit	60%	100 %
Scheduling	100%	100 %
Authentication	91%	100 %
Heartbeat	60%	90%
Pooling	80%	90%

IV. CONCLUSION

BUDGET enables the traceability researchers to mine software repositories of 22 million source codes to create training sets. Since BUDGET is accessible for the public, it would enable the researchers in the community to conduct similar experiments, reproduce the results and expand their work. Based on our assessment, the tool achieved an average accuracy of 90% and above for the best-case sampling strategy. Certainly this level of accuracy can provide the researchers with an initial dataset with a good quality. Further manual vetting can enhance the dataset. The cost of automated dataset generation followed by a manual vetting is significantly less than a purely manual approach which typically takes weeks to months to create datasets. In future work we aim to extend the BUDGET tool and generate different sets of traceability artifacts.

V. ACKNOWLEDGEMENT

The work in this paper was partially funded by the US National Science Foundation grant #CCF-1543176.

REFERENCES

[1] The Health Insurance Portability and Accountability Act of 1996 (HIPAA), 1996.
[2] F. Bachmann, L. Bass, and M. Klein. Deriving architectural tactics: A step toward methodical architectural design. Technical report, DTIC Document, 2003.
[3] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*, pages 155–164. ACM, 2010.
[4] J. Cleland-Huang, A. Czauderna, M. Gibiec, and J. Emenecker. A machine learning approach for tracing regulatory codes to product specific requirements. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1, ICSE 2010, Cape Town, South Africa, 1-8 May 2010*, pages 155–164, 2010.
[5] J. C.-H. Mehdi Mirakhorli. Detecting, tracing, and monitoring architectural tactics in code. *IEEE Trans. Software Eng.*, 2015.
[6] M. Mirakhorli, A. Fakhry, A. Grechko, M. Wieloch, and J. Cleland-Huang. Archie: a tool for detecting, monitoring, and preserving architecturally significant code. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 739–742. ACM, 2014.
[7] M. Mirakhorli, Y. Shin, J. Cleland-Huang, and M. Cinar. A tactic centric approach for automating traceability of quality concerns. In *International Conference on Software Engineering, ICSE (1)*, 2012.
[8] G. Salton. *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.
[9] I. University of California. The sourcerer project. sourcerer.ics.uci.edu.