SIG Proceedings Paper in LaTeX Format*

Extended Abstract†

Ben Trovato‡
Institute for Clarity in Documentation
Dublin, Ohio
trovato@corporation.com

G.K.M. Tobin§
Institute for Clarity in Documentation
Dublin, Ohio
webmaster@marysville-ohio.com

Lars Thørväld¶
The Thørväld Group
Hekla, Iceland
larst@affiliation.org

Valerie Béranger
Inria Paris-Rocquencourt
Rocquencourt, France

Aparna Patel
Rajiv Gandhi University
Doimukh, Arunachal Pradesh
India

Huifen Chan
Tsinghua University
Haidian Qu, Beijing Shi, China

Charles Palmer
Palmer Research Laboratories
San Antonio, Texas
cpalmer@prl.com

John Smith
The Thørväld Group
jsmith@affiliation.org

Julius P. Kumquat
The Kumquat Consortium
jpkumquat@consortium.net

ABSTRACT

This paper provides a sample of a LaTeX document which conforms, somewhat loosely, to the formatting guidelines for ACM SIG Proceedings.[1]

CCS CONCEPTS

• **Computer systems organization** → **Embedded systems**; *Redundancy*; Robotics; • **Networks** → Network reliability;

KEYWORDS

ACM proceedings, LaTeX, text tagging

ACM Reference Format:
Ben Trovato, G.K.M. Tobin, Lars Thørväld, Valerie Béranger, Aparna Patel, Huifen Chan, Charles Palmer, John Smith, and Julius P. Kumquat. 1997. SIG Proceedings Paper in LaTeX Format: Extended Abstract. In *Proceedings of ACM Woodstock conference (WOODSTOCK'97),* Jennifer B. Sartor, Theo D'Hondt, and Wolfgang De Meuter (Eds.). ACM, New York, NY, USA, Article 4, 2 pages. https://doi.org/10.475/123_4

*Produces the permission block, and copyright information
†The full version of the author's guide is available as `acmart.pdf` document
‡Dr. Trovato insisted his name be first.
§The secretary disavows any knowledge of this author's actions.
¶This author is the one who did all the really hard work.
[1]This is an abstract footnote

WOODSTOCK'97, July 1997, El Paso, Texas USA

ACM ISBN 123-4567-24-567/08/06…$15.00
https://doi.org/10.475/123_4

1 INTRODUCTION

1.1 Context

Our research aims to study how developers develop source code by analyzing their in-IDE activities, focusing particularly on testing. The data is captured from Visual Studio IDE using FeedBag, a general purpose interaction tracker. FeedBag does not only capture the events themselves, but also the additional context information such as timing, location, session... This saves us the trouble of recreating the context of the interaction by analyzing the event stream. Some of these context information (e.g session) are used for every event, while some others are captured for specific types of events only.

We use the latest version of KaVE dataset (released on Jan 18, 2018). It contains 11M events from 81 developers over 1527 days, which are divided into 16 different types. Each directory in the dataset is the first date some users submitted their events, and the full directory name can be used as user identification. Among 81 users, only 25 have test events, with user âĂIJ2016-05-09/4.zipâĂİ performing the majority of the tests (2200/3609).

2 FORMULATE PROBLEM

We would like to propose the predictive model to predict whether a sequence of developer's actions causes an error in testing.

We need to distinct our approach from bug localization and defect prediction.

Are there any errors that are caused by wrong sequence of actions.

Taking events into account

3 DATA EXTRACTION

Each test event contains several test methods which have different test results. We aim to build a sequence of events preceding each test method and apply the LSTM model on it to discover what pattern of actions/events that cause test failure. There are 11M events in the dataset and only 3K of them are test events, so the number of non-test events is enormous. Some events, for example Completion Events and Edit Events, contain information about the surrounding codes as well (TypeShape and SST), making the file size a lot bigger than a normal event. Copying the whole JSON file of each event to the sequence seems to be a bad solution, costing huge amount of memory and running time. Therefore, we have come up with another way to preprocess the data before storing it in the sequences.

We noticed the events are divided into 16 different types, and although they share only a few common attributes like timing, session, and trigger, the total number of attributes presented in the dataset is relatively small. Furthermore, some attributes already have a known set of values according to KaVE project repository e.g, TriggeredBy attribute only has 5 possible values:

- *Unknown* (0)
- *Click* (1)
- *Shortcut* (2)
- *Typing* (3)
- *Automatic* (4)

This gave us the idea of constructing a dictionary for every attribute presented in the dataset that maps every possible value of that attribute with a number. By doing this, we can transform every event to a vector of numbers, every number represents the value of the corresponding attribute as stored in the corresponding dictionary. Each vector will contains values for all attributes, so for attributes that are not present in the original file of the sequence, we mark it with a value of -1.

Below is an example of a vector representation of a Command Event:

[5, -1,...,12615, 15919, -1,..., 11, -1,...,3707, 11, -1,..., 0, -1,...]

The "..." represents attributes that do not appear in the original event file and is marked with -1, while other numbers in order of appereance are:

- **$type**: KaVE.Commons.Model.Events.CommandEvent
- **ActiveWindow**: 0Win:vsWindowTypeDocument main.c
- **ActiveDocument**: 0Doc:C/C++/Project1/main.c
- **CommandID**: VsAction:1:Edit.CharLeft
- **IDESessionUUID**: 8f80a55a-0eb6-4ff7-ac07-303dc1214f63
- **KaVEVersion**: 0.1015-Default
- **TriggeredBy**: 0 (Unknown)

We call the attributes with a known set of values e.g, TriggeredBy, known attributes, for which we can build a dictionary ourselves. However, for unknown attributes, a script is needed to extract all of its distinct values from the dataset. This is an extremely long process due to the datasetâĂŹs significantly large size. As we went through every file, we discovered that there is one file with a strange symbol that cannot be encoded using the default ASCII code (we are using PyCharm IDE for this task) which crashes the whole program. As a result, we have to switch to use UTF8 encoding scheme instead and restart the whole process.

After generating the dictionaries, we iterated through every file in the dataset and encode the values. At this stage, we could write the whole vector to the sequence of events as mentioned before, however, in order to make it even less memory consuming and compatible with the LSTM model, we stored every unique event vector in a list and write only the index of that event to the sequence. Hence, at the end of the extracting process, we end up with an array of event vectors, an array of sequences, each sequence contains the indices of the events in the first array, and an array of all the test methodsâĂŹ results, corresponding to the sequences.

REFERENCES