Detection and Analysis of Behavioral T-patterns in Debugging Activities

César Soto-Valero, Johann Bourcier, Benoit Baudry

Detection and Analysis of Behavioral T-patterns in Debugging Activities

César Soto-Valero
Universidad Central de Las Villas
cesarsotovalero@gmail.com

Johann Bourcier
University of Rennes 1-IRISA
johann.bourcier@irisa.fr

Benoit Baudry
KTH Royal Institute of Technology
baudry@kth.se

ABSTRACT

A growing body of research in empirical software engineering applies recurrent patterns analysis in order to make sense of the developers' behavior during their interactions with IDEs. However, the exploration of hidden real-time structures of programming behavior remains a challenging task. In this paper, we investigate the presence of temporal behavioral patterns (T-patterns) in debugging activities using the THEME software. Our preliminary exploratory results show that debugging activities are strongly correlated with code editing, file handling, window interactions and other general types of programming activities. The validation of our T-patterns detection approach demonstrates that debugging activities are performed on the basis of repetitive and well-organized behavioral events. Furthermore, we identify a large set of T-patterns that associate debugging activities with build success, which corroborates the positive impact of debugging practices on software development.

KEYWORDS

Debugging interactions; developers' behavior; T-patterns analysis; empirical software engineering

ACM Reference Format:
César Soto-Valero, Johann Bourcier, and Benoit Baudry. 2018. Detection and Analysis of Behavioral T-patterns in Debugging Activities. In *MSR'18: 15th International Conference on Mining Software Repositories, May 28–29, 2018, Gothenburg, Sweden.* ACM, Gothenburg, Sweden, 4 pages. https://doi.org/10.1145/3196398.3196452

1 INTRODUCTION

Debugging is a widely used practice in the software industry, which facilitates the comprehension and correction of software failures. When debugging, developers need to understand the pieces of the software system in order to successfully correct specific bugs. Modern Integrated Development Environments (IDEs) incorporate useful tools for facilitating the debugging process, allowing developers to focus only in their urgent needs during the fixing work. However, debugging is still a very challenging task that typically involves the interaction of complex activities through an intense reasoning workflow, demanding a considerable cost in time and effort [5].

Due to the complex and dynamic nature of the debugging process, the identification and analysis of repetitive patterns can benefit IDE designers, researchers, and developers. For example, IDE designers can build more effective tools to automate frequent debugging activities, suggesting related tasks, or designing more advanced code tools, thus improving the productivity of developers. Furthermore, researchers can better understand how debugging behavior is related to developers' productivity and code quality. Unfortunately, most of existing studies on debugging activities within IDEs do not consider the complex temporal structure of developers' behavior, thus including only information about a small subset of possible events in the form of data streams [4].

The detection of temporal behavioral patterns (T-patterns) is a relevant multivariate data analysis technique used in the discovery, analysis and description of temporal structures in behavior and interactions [3]. This technique allows to determine whether two or more behavioral events occur sequentially, within statistically significant time intervals.

In this paper, we perform a T-patterns analysis to study debugging behavior. More specifically, we examine the relations of debugging events with other developers' activities. Through the analysis of the MSR 2018 Challenge Dataset, consisting of enriched event streams of developers' interactions on Visual Studio, we guide our work by the following research questions:

- **RQ_1:** What developing events are the most correlated with debugging activities?
- **RQ_2:** Can we detect behavioral T-patterns in debugging activities?
- **RQ_3:** Is the analysis of T-patterns a suitable approach to show the effect of systematic debugging activities on software development?

We aim to answer these question by analyzing a set of 300 debugging sessions filtered from the MSR 2018 Challenge Dataset of event interactions. The objective of our analysis is twofold: (1) to provide the researchers with useful information concerning the application of T-patterns analysis in the study of developers' behavior; and (2) to present empirical evidence about the influence of debugging on software development.

Previous studies analyzed debugging behavior using patterns detection methods. For example, in the development of automated debugging techniques for IDE tools improvement [4]. However, to the best of our knowledge, this is

MSR'18, May 28–29, 2018, Gothenburg, Sweden

ACM ISBN 978-1-4503-5716-6/18/05…$15.00
https://doi.org/10.1145/3196398.3196452

the first attempt of using T-patterns analysis to investigate debugging session data.

2 DATA MANAGEMENT

The dataset for the 2018 MSR Challenge, released on March 2017 by the KaVE Project[1], contains over 11M enriched events that correspond to 15K hours of working time, originating from a diverse group of 81 developers [6]. The data was collected using FeedBaG, an interaction tracker for Visual Studio, which was designed with the purpose of capturing a large set of different in-IDE interactions during software developing in the shape of enriched event streams [1].

The THEME software[2] supports the detection, visualization and analysis of T-patterns. It has been successfully applied in many different areas, from behavioral interaction between human subjects and animals to neural interactions within living brains [2]. Due to the data transferred by contributors is anonymous, we base our T-patterns analysis on the session Id that identifies developers' work during each calendar day. Our filtering routine removes duplicate events and generates individual session files with a structure appropriate for THEME. Date-time information of triggered events is converted to epoch-second values, which is an integer representing the number of elapsed seconds from 1970-01-01T00:00:00Z. Only sessions with debugging interactions where retained for further analysis. Our resulting dataset contains 300 sessions and more than 662K events. Figure 1 shows an example of the data inputs: the variable vs. value correspondence table with the debugging-related event types filtered ("vvt.vvt") and a data file of debugging interactions ("DebuggingSession.txt").

"vvt.vvt"

BuildEvent
BuildEvent_Successful
BuildEvent_Unsuccessful
CommandEvent
Debug_Start
Debug_StepInto
Debug_StepOver
File_SaveSelectedItems
. . .
EditEvent
EditEvent_Large
EditEvent_Short
IDEStateEvent
IDEStateEvent_Startup
IDEStateEvent_Shutdown
. . .

"DebuggingSession.txt"

Time	Event
1471997780	: {means tracking begins}
1471997781	*IDEStateEvent_Startup*
1471997802	*EditEvent_Large*
1471998141	*Debug_Start*
1471998167	*Debug_StepOver*
. . .	. . .
1472084417	*File_SaveSelectedItems*
1472085440	*BuildEvent_Successful*
1472085557	*IDEStateEvent_Shutdown*
1472615935	& {means tracking ends}

Figure 1: Data input structure for THEME software.

We are mostly interested in debugging events triggered using commands, such as *"Debug.Start"* or *"Debug.StepInto"*, which represent the user's invocation of a direct debugging action in the IDE. We decide to keep other related event types that can bring additional information about the programmer's debugging behavior (e.g., *"EditEvent"*, *"TestEvent"* or *"BuildEvent"*). To do so, we append onto each event type string its respective descriptor. For instance, we retain information about the amount of editing according to the size of changes made in the file (e.g., *"Large"* or *"Short"*), the result of tests (e.g., *"Successful"* or *"Failed"*), or the build result (e.g., *"Successful"* or *"Unsuccessful"*).

Our analysis goes beyond the discovery of events' associations. We are more interested in explaining those connections in terms of developers' behaviour by means of T-patterns analysis. In the following, we perform the events analysis using THEME software. First, we show how interesting T-patterns can be detected and visualized through the fine-grained inspection of interactions in individual debugging sessions. Next, we aim to find general behavioral patterns that occur within statistical significance time thresholds for all the debugging sessions studied.

3 T-PATTERNS ANALYSIS

In this section, we summarize the main concepts regarding the detection and analysis of T-patterns [3]. Through the use of an active debugging session as case study, we illustrate the benefits of using THEME software as a tool for exploring hidden real-time structures of programming behaviour in IDEs. Our general approach consists of 3 phases: (1) visualization of debugging interactions in the form of T-data; (2) detection of T-patterns in debugging sessions; and (3) validation and analysis of the detected T-patterns.

T-data. A T-data consists in a collection of one or more T-series, where each T-series represents the occurrence points $p_1, ..., p_i, ..., p_n$ of a specific type of event during some observation interval $[1, T]$. Figure 2a shows an example of T-data coded from a debugging session with 166 squared data points (events occurrences), 25 T-series (event types), and a duration of 823 units. Each T-series in the Y-axis represents an event activity triggered in the IDE during the session, while the X-axis is the time in which each specific event was invoked. For the search parameters used, the blue squares represent detected T-patters, while the red ones did not.

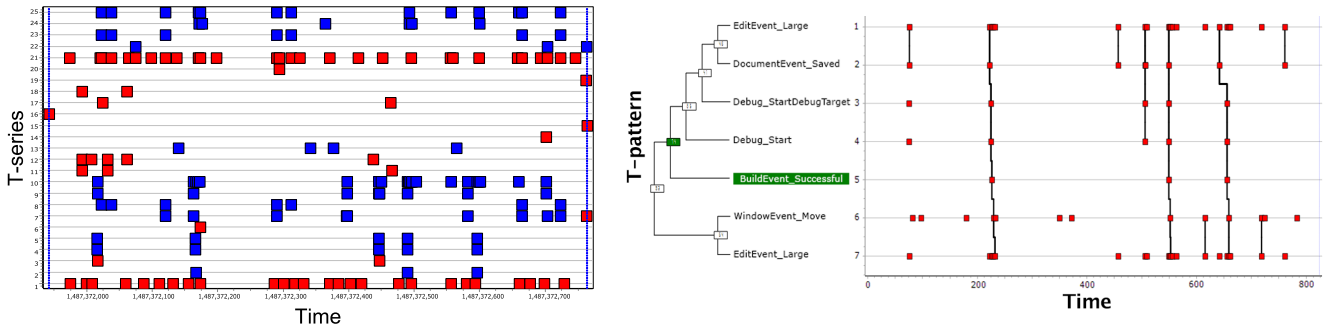
T-pattern. A T-pattern is composed of m ordered components $X_1 \dots X_i \dots X_m$, any of which may be occurrence points or T-patterns, on a single dimension (time in this case), such that, over the occurrences of the pattern the distances $X_i \rightarrow X_{i+1}$, with $i \dots m-1$, varies within a significant small interval $[d_1, d_2]_i$, called a critical interval (CI). Hence, a T-pattern Q can be expressed as:

$$Q = X_1[d_1, d_2]_1 \dots X_i[d_1, d_2]_i X_{i+1} \dots X_{m-1}[d_1, d_2]_{m-1} X_m$$

where m is the length of Q and $X_i[d_1, d_2]X_{i+1}$ means that within all occurrences of the pattern in T-data, after an occurrence of X_i at the instant t, there is a time window $[t + d_1, t + d_2]_i$ within which X_{i+1} will occur. Any T-pattern Q can be divided into at least one pair of shorter ones related by a corresponding CI: $Q_{left}[d_1, d_2]Q_{right}$. Recursively, Q_{left} and Q_{right} can thus each be split until the pattern $X_1 \dots X_m$ is expressed as the 1 to m terminals (occurrence points or event types) of a binary-tree.

T-patterns detection. The T-patterns detection algorithm consists in a set of routines for CI detection, pattern construction and pattern completeness competition. The algorithm works bottom-up, level-by-level and uses competition

[1] Available at http://www.kave.cc/datasets
[2] For more information see http://patternvision.com



(a) T-data representation.

(b) T-pattern visualization.

Figure 2: T-patterns analysis of a debugging session, both figures were created with THEME.

and evolution to deal with redundant detections, where partial and equivalent patterns are removed. THEME provides statistical validation features, global and per pattern, using randomization or Monte Carlo repeated simulation [2].

T-patterns visualization. A T-pattern can be viewed as a hierarchical and self-similar pseudo fractal pattern, characterized by significant translation symmetry between their occurrences. Figure 2b shows the binary detection tree of a complex T-pattern of length 7 found in the debugging session of Figure 2a. The large vertical lines connecting event points indicate the occurrence time of the T-pattern. The node marked in green indicates an event that can be predicted from the earlier parts of the pattern (also called T-retrodictor).

4 GENERAL FINDINGS

We perform an exploratory data analysis to examine the association among events. We use the phi coefficient of correlation, a common measure for binary correlation, and the tidytext R package in order to visualize how often events appear together relative to how often they appear separately [7]. Figure 3 shows the 10 developers' activities that we find more correlated with debugging ($\phi > 0.5$).

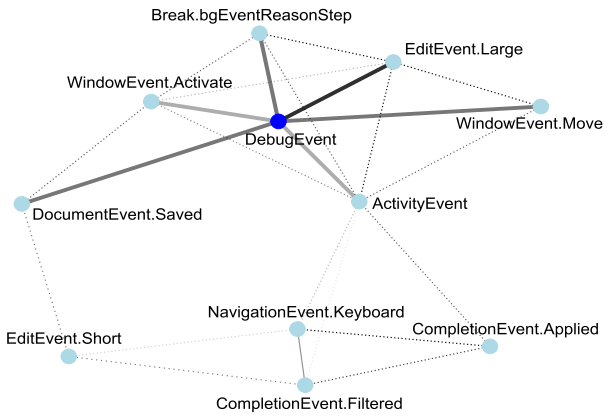


Figure 3: Pairwise correlation between events related to debugging activities.

From the figure, we observe that debugging activities are strongly correlated with code editing, window interactions, document saving, and activity events. In addition, we found that code completion, keyboard navigation and short code editing events are not directly correlated with debugging activities. Based on the observation of Figure 3, we derive the answer to the $\mathbf{RQ}_1$ as follows:

> **Answer to $\mathbf{RQ}_1$:** Debugging activities are more correlated with editing, file handling, window interactions and activity events than with other general commands or event types.

We are mostly interested in analyzing general patterns of events that occur within the debugging workflow. Such patterns allow for insights into the dynamic nature of developer's behavior while debugging software. Accordingly, all debugging sessions were ordered and concatenated in time to conform a single dataset for global analysis with THEME. Thus, the 300 debugging sessions were merged, resulting in a dataset with 263 different event types and more than 460K events' occurrences.

The following search parameters were fit in THEME via grid search: (a) detection algorithm = FREE; (b) minimum number of occurrences of pattern = 10; (c) significance level = 0.0005 (0.05% probability of any CI relationship to occur by chance); (d) maximum number of hierarchical search levels = 10; (e) exclusion of frequent event types occurring above the mean number of occurrences of ± 1 standard deviations.

For the above parameters, more than of 12K of T-patterns were detected. We run the algorithm on 10 randomized versions of the data, using the same search parameters, to check if the set of detected T-patterns differentiate significantly from those obtained randomly. Figure 4 shows the comparison between the distributions of the detected patterns on the original data and the average number of patterns detected after the randomization procedure. The incidence of T-patterns in real data was significantly greater than in its randomized versions. Accordingly, it is clear that the T-patterns detected in the original dataset were not obtained by chance. This result demonstrates that debugging activities are organized on the basis of behavioral events, which occur sequentially

Table 1: Summary of T-patterns detected which reflect the relation of debugging activities with build results.

Build Result	Occurrence	Length	Duration	T-pattern Example
Successful	735	4.87±0.72	580.09±232.51	(*Debug.Start*((*Debug.StepOver Debug.StopDebugging*)*BuildEvent.Successful*))
Unsuccessful	67	2.25±0.25	120.71±35.91	(*Debug.Start*(*Edit.Delete*(*DocumentEvent.Saved BuildEvent.Unsuccessful*)))

and within significant constraints on the time intervals that separates them. Based on this result, we derive the answer to the $\mathbf{RQ}_2$ as follows:

Answer to $\mathbf{RQ}_2$: The validation of the T-patterns detected using THEME provides meaningful evidence about the presence of behavioral patterns in debugging activities.

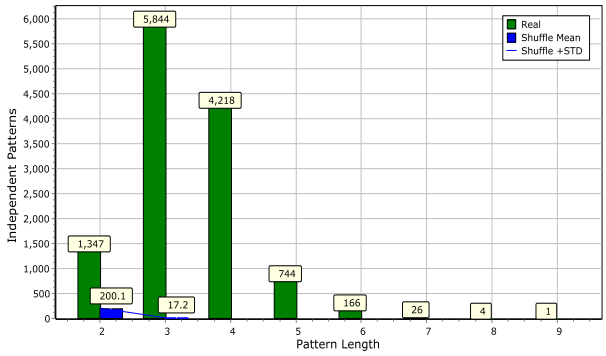


Figure 4: Distribution of T-patterns lengths detected in real and randomized data.

Once T-patterns have been detected, the next challenge is to select relevant T-patterns for subsequent analysis. We are interested in study T-patterns that associate debugging activities with build results. To this end, we used the filters available in THEME, which allow to search for the presence of desired event types in patterns. We found a total of 735 T-patterns that directly associate debugging activities with successful builds, whereas only 67 T-patterns were found for unsuccessful builds. This result shows that, after a methodical sequence of debugging activities, generally the developers have much more chances to achieve successful builds.

Table 1 present a global comparison between the T-patterns found in debugging sessions that are directly related with successful and unsuccessful build results. From the table, we can see that T-patterns related to successful builds occurs more frequently and have a more complex structure, with higher values of patterns' length and duration. On the other hand, T-patterns associated with unsuccessful builds present a more simple structure, with a mean length value of nearly 2 events only and a duration that is almost five times smaller than T-patterns associated with successful builds. This result show that more complex debugging sessions (e.g., those in which developers utilize more specialized debugging tools or invert more time to complete) are more likely to pass the builds and correct software failures.

By analyzing the T-patterns of sessions with unsuccessful builds, we find that their contain mostly events that introduce minor changes in code (e.g., *"Edit.Delete*, *"Edit.Paste"*). We hypothesize that this type of debugging sessions were used to quick trace the effect of these changes. Table 1 also shows representative examples of T-patterns occurrences for both types of build results. Based on the T-patterns analysis performed, we derive the answer to the $\mathbf{RQ}_3$ as follows:

Answer to $\mathbf{RQ}_3$: The quantitative analysis of detected T-patterns in debugging sessions shows that, in general, complex debugging activities achieve successful builds.

5 CONCLUSION

In this paper, we introduced T-patterns analysis as a useful approach to better understand developer's behavior during in-IDE activities. Through the analysis of 300 sessions with debugging interactions, the results obtained using the THEME software bring evidences about the presence of common T-patterns during debugging. In particular, our analysis show a strong connection between debugging activities and successful builds. We believe that the study of the developers' activities using T-patterns analysis can advance the understanding about the complex behavioral mechanism that meddle during the process of software developing, which can benefit to both practitioners and IDE designers. In order to aid in future replication of our results, we make our THEME project, filtered dataset and R scripts publicly available online[3].

REFERENCES

[1] S. Amann, S. Proksch, and S. Nadi. 2016. FeedBaG: An interaction tracker for Visual Studio. In *2016 IEEE 24th International Conference on Program Comprehension (ICPC)*. 1–3.

[2] Magnus Magnusson, Judee Burgoon, and Maurizio Casarrubea. 2016. *Discovering hidden temporal patterns in behavior and interaction: T-pattern detection and analysis with THEME™*. Springer-Verlag New York.

[3] Magnus S. Magnusson. 2000. Discovering hidden time patterns in behavior: T-patterns and their detection. *Behavior Research Methods, Instruments, & Computers* 32, 1 (2000), 93–110.

[4] Chris Parnin and Alessandro Orso. 2011. Are Automated Debugging Techniques Actually Helping Programmers?. In *Proceedings of the 2011 International Symposium on Software Testing and Analysis (ISSTA '11)*. ACM, New York, NY, USA, 199–209.

[5] Michael Perscheid, Benjamin Siegmund, Marcel Taeumel, and Robert Hirschfeld. 2017. Studying the advancement in debugging practice of professional software developers. *Software Quality Journal* 25, 1 (2017), 83–110.

[6] Sebastian Proksch, Sven Amann, and Sarah Nadi. 2018. Enriched Event Streams: A General Dataset For Empirical Studies On In-IDE Activities Of Software Developers. In *Proceedings of the International Conference on Mining Software Repositories*.

[7] Julia Silge and David Robinson. 2016. tidytext: Text Mining and Analysis Using Tidy Data Principles in R. *The Journal of Open Source Software* 1, 3 (2016).

[3] https://github.com/cesarsotovalero/msr-challenge2018