BACHELOR

Smart Journey Mining

Stoev, Stefan S.

Award date:
2022

Department of Mathematics and Computer Science
Process Analytics

Smart Journey Mining

Bachelor End Project Report

Stefan Stoev
1397664

Supervisors:
Felix Mannhardt

28-02-2022

Abstract

Nowadays process mining has different applications in multiple industries such as building process models, identifying bottlenecks in processes and improving on them, such that the optimal output is reached. In this project we work with data about the touch points a user has with a software which evaluates his programming skills. This data could be analyzed to reveal useful insights about user behavior. However real world data is often unstructured and needs to be transformed and labeled in an appropriate manner such that process algorithms work. We attempt to design an approach that deals with duplicate labels in an event log. We focus on analyzing the task execution as a specific subset of the entire process, the one which is related the most to user behavior. By filtering out only necessary events and using the order in which they come in an event log we manage to build a Directly-follows graph that shows the sequence in which tasks come and the average time users spent on them. An arising issue is that the break times of users cannot be identified, thus not revealing the intrinsic time spent.

Contents

Chapter 1

Introduction

1.1 Context and Topic

In today's technological society the interactions between service providers from countless industries and their customers have been digitized to a great extent. The amount of services that can be offered online ranges from meeting people who are on the other side of the planet to ordering of goods and making payments. All of these activities which customers engage in on the web leave a digital "footprint" with some associated timestamp which are collected and stored in some data warehouse. A collection of many such customer footprints is data which if processed and analyzed profoundly may reveal many insights about the customer journey, help analyze user behavior and answer many industry-specific questions to service providers. Process mining is a sub-field of data science which consists of a set of data modelling techniques to discover processes, check compliance, discover bottlenecks and suggest improvements. It uses event log data where each event refers to some activity (also called "tasks"), or well-defined step in some process and is related to a particular case, or process instance. Many events belonging to the same case and ordered chronologically represent one run of the process. The aim of process mining is to create from an event log a consistent and explicit process model which captures the dynamics of the process at hand [13].

Theoretically, building a process model using an event log sounds straightforward, however unfortunately the reality is more complex. In order for the algorithms that built these models to run successfully, they make the assumption that the labels of events are precise and consider that each label is a unique event represented by a single task node in the model. This means that each unique activity that has occurred in one or more user cases is mapped

to its own node in the model and thus, represents some stage of the customer journey. The issue with real word data is that usually it is unstructured and could contain inaccuracies or missing values. By unstructured we mean it is not easy to search in it, because it does not have a recognisable structure. The saying "garbage in - garbage out" in a data science context translates that if you input messy and unorganized data into some algorithm the output will be as messy and unclear and most likely inaccurate, rendering it useless. Therefore, it turns out that one of the most difficult tasks for data scientists is to pre-process their data such that it can produce clear and sound results when some algorithm is applied on it.

There are many companies on the market which can improve their services if they could only understand better their customer behavior and such a company is "GrepS". It is in the technological sector and its purpose is to develop innovative and accurate software for evaluation of programming skills in the "Java" language. In this project we work with data from GrepS, which is a collection of event "footprints" from users which have been given to solve a set of *tasks* and associated *sub-tasks*. Unfortunately, it is not in the typical event log format. Firstly, the length of traces varies greatly, since it is a test and each user may submit it whenever they want. Furthermore, there are activities with duplicated names in many traces. For example the activity "Task Downloaded" occurs multiple times in each case as can be seen on Table 1.1. The multiple occurrences of the same label for the same item in a case do not mark the same activity, but indicate different stages of a task. The label appears when a user starts working on it, begins a sub-task, finishes it, gives feedback to it and receives his results from it. This leads to case heterogeneity which means the event log contains traces of various length with diverse and unstructured behavior [5]. In this project tasks are not viewed as individual exercises, rather we look at them as a sequence and examine the order in which they come in the test. In other words we look at the exercises as *First exercise, Second exercise, ..., Last exercise* and examine which specific items occurred at each possible position.

In the data there are many events which are not done by users, but from the system and happen internally, such as latency measures, loading pages or calculation of results. These events as well constitute a part of the customer journey and could provide useful information on the users, such as extracting user location from latency measures or using the grades which the system calculates to follow user performance during the journey. However, as these events happen internally within the system, reaching a maximum of a couple of seconds, we shift our focus from them. They increase the complexity of the event log and make it even harder for process discovery algorithms to produce useful results. Conversely, the user-initiated activities indicate action from

Case ID	Timestamp	Label	Item
1	01-01-20 15:00:35	Begin test	Logged in
1	01-01-20 15:02:10	Task Downloaded	item 42
1	01-01-20 15:15:18	Task Downloaded	item 42
1	01-01-20 16:01:01	Task Downloaded	item 42
1	01-01-20 16:45:18	Feedback Given	item 42
1	01-01-20 17:00:38	Task Downloaded	item 85
1	01-01-20 17:23:18	Task Downloaded	item 85
2	01-13-20 09:00:35	Begin test	Logged in
2	01-13-20 09:02:20	Task Downloaded	item 20a
2	01-13-20 10:25:13	Task Downloaded	item 20a
2	01-13-20 10:30:32	Feedback Given	item 20a
2	01-14-20 15:32:38	Task Downloaded	item 14
2	01-14-20 16:13:18	Task Downloaded	item 14
2	01-14-20 17:06:02	Task Downloaded	item 42

Table 1.1: A subset of the data which has duplicate labels for the task event. The length of both traces is different because they have different number of given tasks and the label "Task Downloaded" appears for every task.

the developer which could reveal insights about his behavior. One of the aims of this project is to formulate a high-level model of the process which users undergo and attempt to understand their behavior. The way to accomplish this is to deal with data inconsistencies and irregularities. We assume that if appropriate labels are added and activities which are initiated by the system are disregarded, complexity will be reduced and process discovery algorithms may reach more comprehensible and useful results.

1.2 Research Question

The research question which will be of interest for this project is: **How to re-label duplicate labels of heterogeneous customer journey data in order to make it suitable for process mining techniques?**

A sub-question of interest would be:

- How to deal with duplicate labels of the same activity in event log data?
- How to identify when a user is taking a break during the test?

1.3 Approach

For the execution of the project we will use data provided by GrepS. Since it is in an unstructured form we will transform it such that we gather insights about the journey of a developer. If we want to do analysis on timestamped data, it needs to be transformed into an event log. Initially, we look at the data from a high-level perspective and observe the entire process with all of its activities. The dataset contains too many unique events and each node corresponds to an event, we need a simpler model to observe what is happening internally in the process, specifically in the execution of tasks.

We shift the focus on events which are related to some tasks, such as the beginning, submission, execution of sub-tasks, etc. From these events we are able to construct a clearer, less-entangled map of the process, that illustrates as closely as possible the reality of the customer journey.

In order to get a better understanding of what is happening during the test, we will dive deeper into the events that are related specifically to tasks. This includes all possible aforementioned sub-events of the task and from them we attempt to create a process model only for the execution of the tasks. For the sake of illustrating the process, we decide to look at the tasks of the process as a sequence, instead of individual tasks, so that it is possible to get a more general view of the performance of users. As an example, consider Table 1.1 where item 42 appears as a first task for user with id 1, however in the case of user 2 it is given as a second exercise. Thus, we abstract the notion of specific items and focus on the sequential order of the exercises. The execution of the tasks are the events which are connected the most with user behavior, so we narrow down only the activities related to the tasks. Furthermore, we perform an additional analysis on the feedback which users give to themselves and interpret the average scores they receive. The scores can be used to divide users in different clusters and examine differences between the groups in terms of time, performance and customer journey as a whole. Another idea is to investigate the difficulty for the different exercises and identify which of them proved to be the most difficult for users, judging from their feedback.

1.4 Findings

By re-labelling labels of activities related to tasks, we managed to built a DFG which represents the sequence of their execution. It turns out in 99% in the cases users are given 5 main tasks and the presence of sub-tasks is relatively determined, i.e. it is not mandatory for a user to get a task with

multiple sub-tasks. We perform a rudimentary evaluation of the difficulty of the exercises based on the time people have spent on them. We determine it as "rudimentary" because , due to difficulties we do not account for breaks, which could take a wide range of time. Besides the task which is given at the first position and takes considerably less time, all other ones seem to have been equal in difficulty judging only by the time.

Chapter 2

Background & Preliminary Knowledge

2.1 Understanding Event Logs and Visualizing the Process

Process Mining Preliminaries

To understand the essence of the project a comprehension of what comprises an event log is needed. It is a collection of events and all of them are related a single process. We assume that each single event is associated with a single process instance. Moreover, we assume that each event is related to some activity. In the subsequent chapters when we say "activity" we refer to a member of the set of all possible events which are captured in the event log [12]. All events belonging to the same case and ordered chronologically comprise a single trace of a process, thus the timestamp is a crucial attribute as well [11]. It is possible to have other attributes that give additional information such as cost or resource. Figure 2.1 illustrates an example event log about handling compensation requests. All of the events are grouped by "Case id" and as an additional attribute to distinguish between activities, we have an event id. It serves the purpose to distinguish activities, e.g. activity "Check ticket" with event id 35654425 from "Check ticket" with event id 35654485. A representative trace for the process in Figure 2.1 is <*Register request, Check ticket, Examine thoroughly, Decide, Reject Compensation*>. There are 3 main types of process mining that can be applied on event logs:

- **Process Discovery** where algorithms such as α-miner, inductive miner, heuristic miner take as input event logs and produce a model from it.

- **Conformance** is the process in which we compare an existing process model and some event log for the same process. This way one can check if reality documented in the event log matches the created model and investigate if discrepancies are present.

- **Enhancement** is the part where the model is improved after finding bottlenecks and room for development after performing conformance.

Case id	Event id	Properties				
		Timestamp	Activity	Resource	Cost	...
1	35654423	30-12-2010:11.02	Register request	Pete	50	...
	35654424	31-12-2010:10.06	Examine thoroughly	Sue	400	...
	35654425	05-01-2011:15.12	Check ticket	Mike	100	...
	35654426	06-01-2011:11.18	Decide	Sara	200	...
	35654427	07-01-2011:14.24	Reject request	Pete	200	...
2	35654483	30-12-2010:11.32	Register request	Mike	50	...
	35654485	30-12-2010:12.12	Check ticket	Mike	100	...
	35654487	30-12-2010:14.16	Examine casually	Pete	400	...
	35654488	05-01-2011:11.22	Decide	Sara	200	...
	35654489	08-01-2011:12.05	Pay compensation	Ellen	200	...
3	35654521	30-12-2010:14.32	Register request	Pete	50	...
	35654522	30-12-2010:15.06	Examine casually	Mike	400	...
	35654524	30-12-2010:16.34	Check ticket	Ellen	100	...
	35654525	06-01-2011:09.18	Decide	Sara	200	...
	35654526	06-01-2011:12.18	Reinitiate request	Sara	200	...

Figure 2.1: Extraction of a sample of event log data for handling some compensation requests [12]. Each row represents an event that has an associated activity, is connected to a unique case and occurred at a specific time.

The data collected in our project consists of the steps taken by developers during the execution of a test. We focus on process discovery and try to find a model which describes as accurately as possible our process at hand. In a business context, process mining can reveal multiple insights not only on the mechanisms of the model but on the users which take part in it.

Directly-Follows Graph

In order to visualize how the model looks like and get a deeper understanding of the user behavior, we will use a Directly-follows graph (DFG). They are

a variant of transition systems [13]. The difference is that the shift between states in transition systems is atomic and for DFGs this shift takes time [9].

An example DFG can be seen on Figure 2.2. The nodes of the map substitute the different events of the event log and the directed edges represent the order in which the events come. We make some modifications to the general template of the graph and add additional attributes to the edges such as the participation of the cases in the connection between nodes and as well the average time it takes to get from one node to another. For instance, on Figure 2.2 event A is followed by event B and this order is present in $k\%$ of the cases and it takes on average some amount of time. We use DFGs in a similar fashion as Customer Journey Maps [3], because we try to understand the process at hand and describe what really happened to the users that have undergone the process.

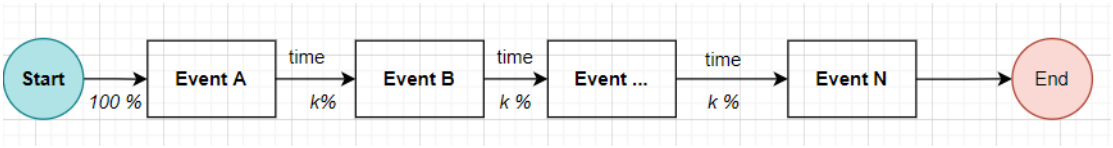


Figure 2.2: Example Directly-follows graph containing N events

Petri Nets

Another renowned modelling method for illustrating process models are Petri nets [13]. They improve on the drawback of Directly-follows graphs and can handle concurrency. A Petri net is a bipartite graph (Figure 2.3) that consists of *places* and *transitions*. A *sound* Petri net is one that there are no two connected adjacent *place* or *transition* nodes. The *places* can contain a *token* and a *marking* is the configuration of the graph based on distribution of tokens over the places. The tokens can be *fired* from one place to another and since the firing is non-deterministic, Petri nets allows for concurrent behavior [1].

2.2 Challenges with event log data

Bose et al. [5] make a high-level systematic identification of process characteristic issues that can manifest in an event log. The ones we are interested in are: **case heterogeneity** and **event granularity**. Case heterogeneity refers to flexibility of event logs. This means the event log consists of traces of various length and scenarios which lead to unstructured behavior. This

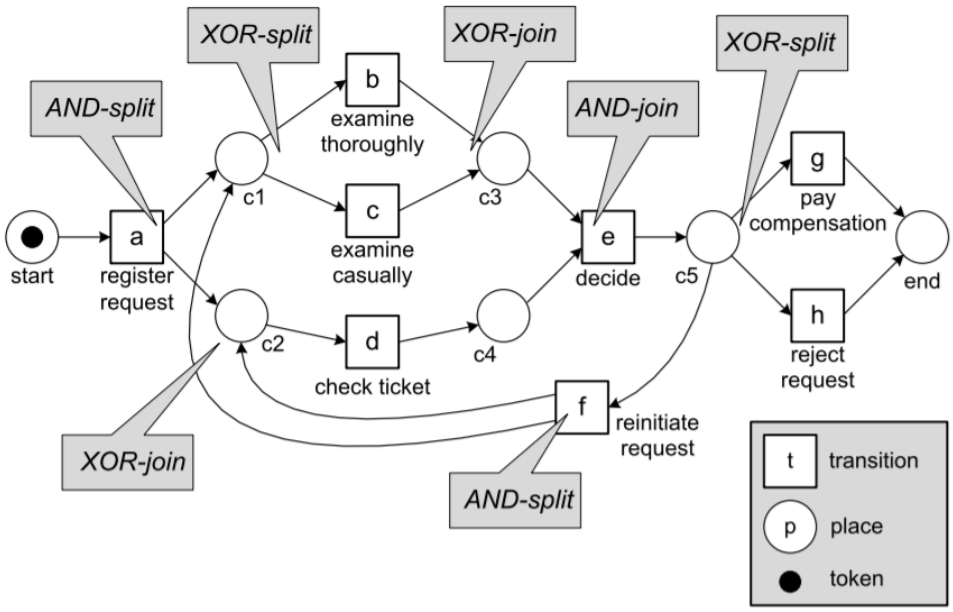


Figure 2.3: An example Petri net [13] representing the event log on Figure 2.1

is possible to occur in cases where the number of steps in a process is not strict and subject to change over time due to new legislation, seasonal effects and other force-major effects [5]. Take for example scenarios where the user can finish the process whenever he decides to or if the data is gathered by systems that support a wide range of variety like medical support ones [5]. A proposed approach to deal with case heterogeneity is trace clustering. Its objective is to partition traces into homogeneous sets of cases. Those traces which share similar characteristics (length, structure, etc.) are put into the same cluster and those who are dissimilar are in different clusters [4].

The main challenge is event granularity which occurs when there is a substantial number of unique activities in the event log and sub-activities, leading to the so-called *spaghetti* process models [5]. In our data this problem refers to those activities which are nested and have duplicate labels. In Table 2.1 you can see a sample of the dataset which serves as an example for nested activities and duplicated labels. Such events are the item/task ones and the latency measures, as mentioned earlier, and assigning users to a virtual cloud. Each of the labels need to be re-labelled such that the initialization of a task, its finish, break time and the feedback to it can be distinguished. In order to deal with this issue, you have to search for common execution patterns across

cases. For instance for each specific item there is a fixed number of activities per case. Usually, these sequences are not random and share some domain relationship [4] as we will see later. One way to deal with them would be to simplify them and look at such sequences from a high-level viewpoint which simplifies the process model. To be more exact, the sub-items we refer to are the different steps of a task - beginning, doing sub-task, submitting, logging out, etc.

Case ID	Timestamp	Label	Item
70	11-01-2020 15:25:35	Task Downloaded	item 20a
70	11-01-2020 15:48:40	Task Downloaded	item 20a
70	11-01-2020 16:10:46	Task Downloaded	item 20a
70	11-01-2020 16:13:58	Feedback Given	item 20a
70	11-01-2020 16:30:04	Task Downloaded	item 14
70	12-01-2020 09:00:35	Task Downloaded	item 14
70	12-01-2020 09:34:41	Feedback Given	item 14
83	03-02-2020 22:00:12	Task Downloaded	item 85
83	03-02-2020 22:40:53	Task Downloaded	item 85
83	03-03-2020 08:12:31	Task Downloaded	item 85
83	03-03-2020 08:44:30	Feedback Given	item 85

Table 2.1: This sample of the dataset is an example of nested activities and duplicated ones. There are several activities which refer to a specific item/task for developers with id 70 and 83, such as beginning the task, completing a sub-point, ending it and giving feedback to it.

To know how to deal with duplicate tasks we first need a clear definition of them. In the paper of Duan and Wei [8] which is a systematic review of the methods to handle duplicated tasks, 3 types of duplicate tasks are identified:

- Tasks which have the same label
- Tasks which appear more than once in a trace
- Tasks which are enabled in different contexts

The third category is of utmost interest to us. Tasks which have identical labels but are enabled in different contexts are the cause for a lot of confusion in process mining algorithms. Consider the following case in Figure 2.4 where activity "A" is followed again by activity "A" [5]. It is unclear whether a new occurrence of activity "A" marks the beginning of a new

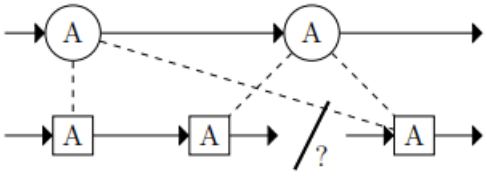


Figure 2.4: An occurrence of duplicate tasks. Both events "A" could belong to the same instance of activity "A" or it is possible they belong to different instances of the activity "A" [5]. The circular nodes represent events in an event log and the square ones are sub-parts of process "A". The question is whether the second occurrence of activity "A" denotes the beginning of the same activity or does it indicate the end of the previous one?

event or it denotes the end of the previous occurrence of "A". This uncertainty stops process mining algorithms which discover processes to reach sound results. The separate events could mean the same thing however their meanings change depending on the context they are enabled in.

Duan and Wei [8]do not refer to loops, i.e. they exclude cases in which the same type of the process is executed repeatedly. In our case however, there are occurrences which can be identified as loops, such as when a subject has refreshed the web page which leads to some activity being executed more than once. In the paper of San Pedro and Cortadella [7] the precision of the process model manages to increase from 50% to 90% after dealing with duplicated labels. While many authors agree there are ways to detect duplicated labels, they reach the conclusion that there does not exist a single generic way to deal with them [8] [7]. Despite this, Lu et al. [10] describe an approach which does not make use of some specific algorithm that refines the labels in their internal mechanism [14] but work directly on the event log.

2.3 Background on GrepS

GrepS is a company based in Norway which develops a system for authentic skill-analysis of software engineers. The test they have created not only helps companies when making a decision if they should recruit a new developer by letting him do the test, but is also useful for people who just want to have an adequate evaluation of their skills. The test is meant for highly-skilled individuals and the evaluation method is what makes it special. They define "skill" as a specific type of ability, where each individual has his own unique distinguishing features. In their measurement model, the developers in GrepS

assume that variables for skill, task performance, time and quality are needed to calculate the final outcome "true skill". Particularly, they rely on the Rasch measurement model which is an item response theory model (IRT) that combines an additive conjoint structure with probabilities [2]. Generally, IRT models present items/tasks to an individual and formulate an estimate of his abilities based on the sum-scores of the item responses. Additionally, they rely heavily on estimates for the difficulty of the task given and the responses across people. A method based on maximum likelihood function is used in order to estimate the difficulty of the task and the ability of the person performing it. Unlike the original dichotomous Rasch model where only two score categories are available (correct, incorrect), the test of GrepS uses a polytomous Rasch model which permits multiple score categories $0 \dots M_i$ where M_i is the maximum score for item i.

The need for this type of evaluation comes from the fact that software quality is not unitary concept [2]. There are different levels to it and some solutions which arrive to the same output may have different quality when if you compare their inner-mechanisms. Furthermore, another factor which is considered when evaluating skill is the trade-off between quality of the solution and the time taken to reach it. If you have to compare the solutions of two developers and their solutions are of equal quality, but one of them has reached it within less time, then this will indicate higher performance. However, comparing a high-quality solution which took a long time versus a low-quality solution which took less time is not an easy task. In the paper of GrepS two main strategies are used to deal with this issue [2]. They combine both strategies and use item-specific scoring rules to calculate the score for the developer.

Chapter 3

Problem Exposition

3.1 Context/Business Understanding

The data which GrepS has gathered about the journey of their customers contains all the touch points a developer has had with the system. Unfortunately, it is unstructured and there are many activities which have duplicated labels and are nested. The heterogeneity of the data comes from the fact that it contains different types of values, such as strings, integers, floating numbers, etc. Furthermore, we classify the events in two main categories: user-initiated and system-initiated ones. Since we are focusing on observing the behavior of the people performing the test, we include only events which are initiated by them. Therefore, to make a proper analysis and re-labelling, some domain-specific specifications and assumptions need to be made. To construct a story out of a trace from all the separate 26 activities we need to stand in the shoes of the participant. What is more, the participant experiences only a user interface which displays only the tasks and saves all the internal system processes such as latency measures, score updates, etc. This is why for the main analysis we focus only on user-initiated events and disregard system-initiated ones.

When a developer begins the test he logs into the system and is taken to the virtual test environment where he is given on average a set of 5 tasks (also called "items") to solve, out of 8 possible. The items appear in different positions in the test. Therefore, it is possible for all of the items to appear in any of the positions from 1 to 6, since the maximum amount of tasks a developer has in our data is 6. Considering this, we decide to focus on the tasks not as single items, but rather as parts of a sequence that is formed by consecutive tasks. In other words, we look at some task not as "item X" but as being the first task, second task, etc.

A phenomenon which constitutes a part of every test are breaks. Breaks are the time a developer has been logged out of the system and not worked on the task he has began. Users have the right to be offline while have started the test for more than 24 hours. Thus, breaks could greatly influence the time spent per task. For this reason, they need to have a separate activity in the event log which marks the beginning of a break. Unfortunately, the data does not contain a specific event which marks the logging out of a user. The multiple task events for a specific item which occur between the events which mark the beginning and the end of the item in a case could be because of multiple reasons: refreshing the web page, beginning a sub-point of the exercise or logging back is possible, as well. This makes the identification of a break period difficult to realize. One way to identify break periods would be to investigate tasks on which the user has spent more than some amount of hours. However, we cannot be certain what fraction of this time the user has actually been idle. For example if a user has spent 20 hours on some task, we cannot say with certainty how much time has he actually worked and how much has been a break. Another approach would be to investigate the logging in events which appear between task ones. Presumably if a user logs in the middle of the test, he either took a break or the system has failed and requested from the user to log in. On Figure 3.1 you can see an example trace where a user most likely took a break. The developer began the first sub-task in 10:02 AM and his next touch point with the system is at 17:00 when he logs in. In the 7 hours between the events the developer finishes the task, however it cannot be derived how many of these hours did he rest.

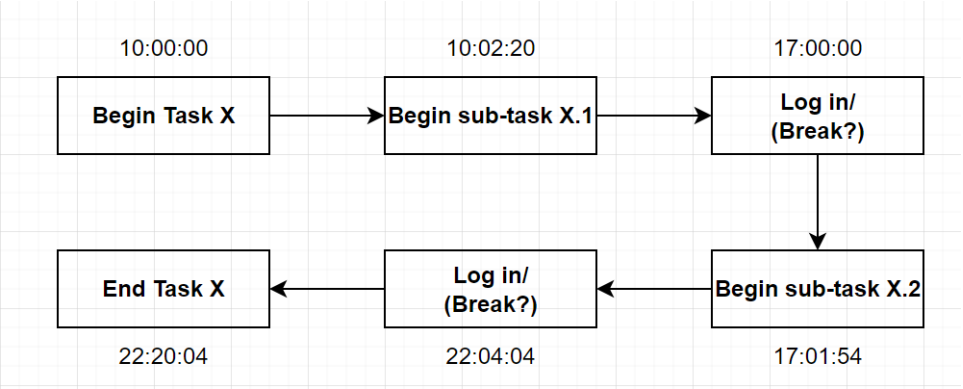


Figure 3.1: An example trace where most likely a break occurred. The time above the nodes indicates the time of start of the event. The "Log in" event appears 7 hours after beginning the first sub-task and minutes right before beginning the next sub-task. The real time during which the user has worked on the first sub-task cannot be defined.

For the scope of the project, we assume all users are in the same time zone and we will use the UTC time zone as reference for the results to avoid confusion (see Section 3.2 for details).

Next, prior beginning the test participants are given 5-10 minutes to familiarize themselves with the tasks [2]. This guarantees that there is no effect of a person's reading speed on his results. The provided data is from the moment people downloaded the code to the moment the last solution is submitted. These assumptions facilitate in understanding as much as possible the reached results.

3.2 Data Understanding

The initially provided data is in a format which cannot be analyzed, thus needed to be transformed such that it has a structure similar to an event log. In the beginning, we created the following attributes:

- Company ID which connects each developer to a specific company.
- Category which identifies whether the case is B2B or B2C. In the B2B case it means that some customer of GrepS is using the test, most likely for recruiting purposes. In the B2C some developer is making use of the test for himself.
- Developer ID which is a unique identifier for developers and can be used as a case identifier.
- Timestamp
- Label which is the activity label.
- Type which identifies the type of the event occurring.
- Message which contains additional information on the different events.

The data contains the minimum required columns an event log should have-case id, timestamp, activity label. However, the labels which are related to the tasks have duplicated names thus, need to be reformulated such that they indicate at which part of the task the developer is at currently. This plays a similar role as the 'life cycle status' column which an event log generally has.

There are a total of 200 developers in the dataset, however out of these 200 only around 60 of them actually begin the test. The other 140 only register themselves without actually giving it a try to solve anything. You

can see on Table 3.1 a trace where the user did not engage into the test. He only registered and began the test without actually doing any exercise. Furthermore, more than 90 % of the cases fall into the B2B category, thus we can assume the results for the developers have been used for recruiting reasons. The total number of unique activities is 27 which includes both system- and user-initiated ones. Despite having so many different activities, the average number of exercises a developer is given is 5.

Case ID	Timestamp	Type	Label
10	2021-03-21 12:09:07	state	Registered
10	2021-03-21 12:09:46	latency	Latency Measured
10	2021-03-21 12:10:06	latency	Latency Measured
10	2021-03-21 12:10:40	latency	Latency Measured
10	2021-03-21 12:11:45	subject	Logged in
10	2021-03-21 12:12:10	state	ready to start
10	2021-03-21 12:13:07	state	started

Table 3.1: In this sample of the data the developer did not perform any actions on the exercises and only registered to the system.

We mentioned that we will account only the ones which the user has initiated, since the main focus is on his behavior and the number is decreased to 8 distinct events. The events which we classified as user-initiated ones are: *registered*, *logged in*, *ready to start*, *task downloaded*, *feedback given*, *item completed* and *test completed*. The events which were considered as system-initiated ones are latency measures, assigning the user to a database server and updating/sharing the user results. The problem with tasks which have duplicated names that we showed on Figure 2.4 occurs here, since we cannot distinguish between the different steps of the activity "Task Downloaded". When a new occurrence of the activity happens, we do not know if it the user has began a new task or he is continuing the previous one.

Initially, the timestamps in the data are not specific dates or hours, but rather in each timestamp cell there is some amount of seconds that at first glance looks arbitrary. However, it turns out all of the events have a common origin in time and simply if we convert the amount of seconds to days and add these days to an origin of our choice, we will get specific dates. You can see a sample of the data which shows the original timestamp values in seconds and the converted time. Due to not having any geographical location data on the users in advance, there is no certainty about from how many different parts of the world users access the test. Thus, we convert the timestamps to

days and choose as an origin date 1^{st} January 2021. When we discuss dates and time we refer to the UTC timezone ,as mentioned in Section 3.2.

Case ID	Timestamp(**before** conversion)	Timestamp(**after** conversion)	Label
2	4028947	2021-02-16 16:09:07	Registered
2	4486716	2021-02-21 23:18:36	Begin test
2	4486718	2021-02-21 23:18:38	Task Downloaded
2	4487560	2021-02-21 23:32:40	Feedback given
2	4487585	2021-02-21 23:33:05	Task Downloaded

Table 3.2: A sample of the data which contains the timestamps in their original form and their converted values using as origin 1st January 2021.

The "Message" column contains additional data on the activities like specific values for latency measures, information on updates of the score of some exercise or you can also find the answers to the feedback questionnaire which developers have given after every completed item. Moreover, as mentioned earlier, the rasch score is a measurement which is an additive conjoint structure, i.e. the scores are updated after every exercise and the scores on 6 different dimensions are present in the "Message" column. The "Type" column contains a word which describes on a high-level the activity which occurred. You can see a sample of the data on Table 3.3 where we include both columns to illustrate the contained in them. This additional information can be used for instance to track the performance change of developers throughout the test and find parts of the test where users faced more difficulty than with other.

Case ID	**Type**	Label	**Message**
3	State	Registered	registered
3	Latency	Latency Measured	region=azure-eu-west-1, average=31.0, limit=80
3	Subject	Logged in	Logged in: Web page
3	state	Started	Started
3	Task	Task Downloaded	Task event: taskDownloaded, item 20a, timeUsed
3	Feedback	Give Feedback	item 20a, response: [0=1,1=3,2=2], comments length: 67

Table 3.3: A sample of the data that shows the "Type" and "Message" columns.

What is more, after every submitted question the system presents a questionnaire for the user to fill in. The questions asks developers not only to evaluate the quality of the task they have just done, but evaluate their own performance. By analyzing this information we may get some idea of the average score people give to themselves.

The software system also updates the score of the user after every question he has done and evaluates him on 6 different dimensions. These scores

are saved in the dataset as a separate activity and allow for tracking the performance change. However, this analysis goes a bit out of the scope of the paper, so it will not be looked into.

Since our data is gathered from users doing a test, the length of the test for different users varies. Most people solve 4-5 exercises, whereas others quit after some of the first ones or immediately after logging in. There is not a general process pattern which applies to all cases.

3.3 Naive Method

In the beginning we use all activities present in the event log, including system-initiated ones, in order for better illustrating the entire process a developer goes through with the test. The DFG in our project captures all the touch points a developer has with the software. The map can be seen on Figure 3.2 and on Table 3.5 you can see a trace which corresponds to the DFG. Nodes represent activities and the edges show connections between activities. Generally, the greater the number of activities, the more difficult it gets for the map to be understood and this applies for our case as well. We have a total of 27 different activities, which results in an entangled, *spaghetti*-like DFG [6]. Apart from the numerous activities, the connections between some of the events adds complexity in understanding what is happening. It appears all activities have a connection with the "Logging in" node. This implies that users either refreshed the page and it sent them back to the page of the website to log in or they had taken a break and this is why they have to log in again.

We tried to make the graph clearer, so we aggregated some activities which are similar and occur consequently. For example all the task events which indicate that a user is a some specific task were aggregated in one node - "Task", due to the reason that the number of tasks across different users vary. On Table 3.4 you can see a sample of the data which is aggregated into the "Task" node. Most of the cases have the following execution of activities: In the beginning everyone registers/signs in the system during the test. All the tasks done by users are aggregated into the node "Task". Afterwards, they wait for their evaluation after completing the test. There are some cases which have ended straightly after finishing the tasks and scores events are omitted. Others have ended just after logging in which means they did not attempt the test. Initially, we thought ties pointing to "Logging In", excluding "register" mean that the user has refreshed the page. It turns out however, that besides refreshing the label for logging in could also indicate the continuation of the test after taking a break. To be able to confirm this

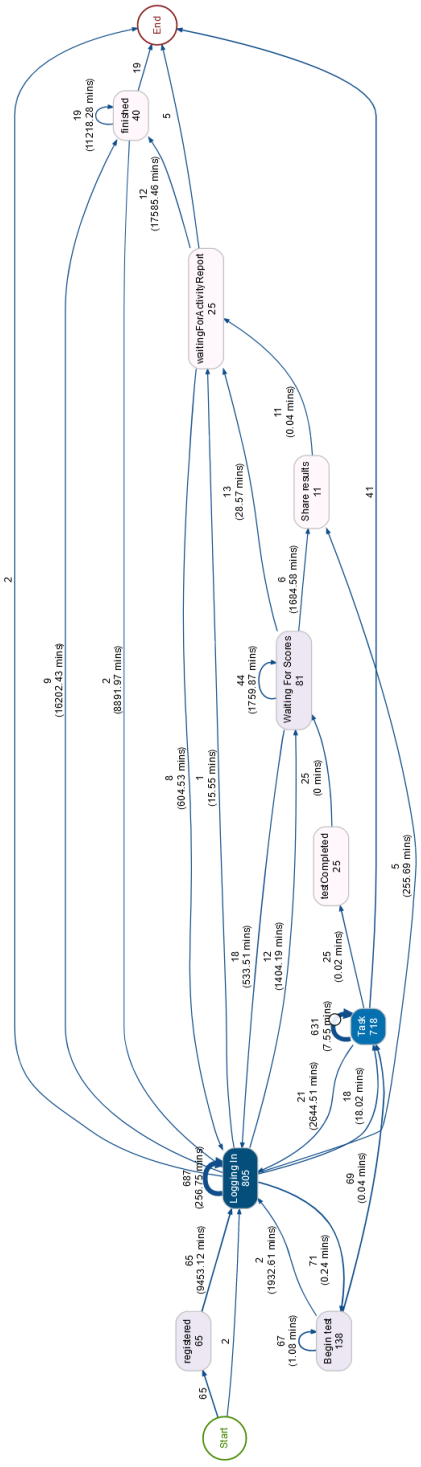
we need to look closer into the time difference between some exercise and a consecutive "Log in" event. If the time difference is greater than a couple of minutes, we can assume that in the majority of cases people have taken a break.

Case ID	Timestamp	**Label**	Item
70	11-01-2020 15:25:35	Task Downloaded	item 20a
70	11-01-2020 15:48:40	Task Downloaded	item 20a
70	11-01-2020 16:10:46	Task Downloaded	item 20a
70	11-01-2020 16:13:58	Feedback Given	item 20a
70	11-01-2020 16:30:04	Task Downloaded	item 14
70	12-01-2020 09:00:35	Task Downloaded	item 14
70	12-01-2020 09:34:41	Feedback Given	item 14
70	03-02-2020 22:00:12	Task Downloaded	item 85
70	03-02-2020 22:40:53	Task Downloaded	item 85
70	03-03-2020 08:12:31	Task Downloaded	item 85
70	03-03-2020 08:44:30	Feedback Given	item 85

Table 3.4: Sample data which is aggregated into a single node - "Task"

In order to go more in depth we can dissect nodes which are aggregated and look at them as a separate process. Since the aim of the entire test is to evaluate the programming skills of users, the node of interest is the task one. It is possible that it can give information task-specific experiences, such as which task seems to be the hardest or if the system has crashed and the user has refreshed multiple times.

Figure 3.2: Directly-follows graph of the journey a developer takes on the GrepS test. The multiple connections between some of the activities and the presence of many nodes makes the map difficult to read.



Case ID	Timestamp	**Label**	Item
11	11-01-2020 13:00:00	Registered	Registration
11	11-01-2020 13:02:11	Latency measure	Measure latency
11	11-01-2020 13:04:11	Log in	Logged in
11	11-01-2020 15:25:35	Task Downloaded	item 20a
11	11-01-2020 15:48:40	Task Downloaded	item 20a
11	11-01-2020 16:10:46	Task Downloaded	item 20a
11	11-01-2020 16:13:58	Feedback Given	item 20a
11	11-01-2020 16:30:04	Task Downloaded	item 14
11	12-01-2020 09:00:35	Task Downloaded	item 14
11	12-01-2020 09:34:41	Feedback Given	item 14
11	03-02-2020 22:00:12	Task Downloaded	item 85
11	03-02-2020 22:40:53	Task Downloaded	item 85
11	03-03-2020 08:09:30	Log in	Logged in
11	03-03-2020 08:12:31	Task Downloaded	item 85
11	03-03-2020 08:44:30	Feedback Given	item 85
11	03-03-2020 09:00:00	Test completed	Finished test
11	03-03-2020 09:00:22	Waiting for scores	waiting for scores
11	03-03-2020 09:01:01	Share results	sharing results
11	03-03-2020 09:02:00	Waiting for activity report	waiting for report
11	03-03-2020 09:02:45	Finished	finished

Table 3.5: Sample data that shows an example trace of the DFG on Figure 3.2.

Chapter 4

Approach

4.1 Time Analysis

The aim of the project is to re-label the labels of activities which have duplicated names and build a model which describes the journey of a developer in the GrepS software. Like previously explained, a trace in our event log is about users undergoing a test and the exercises which are given vary across cases. Furthermore, we know there is a set of 8 unique exercises, each with its own corresponding sub-parts. The fact that a specific item (e.g. "item 42") can be given as the last task in one trace and as second exercise in another trace, makes the building of a sound process model even more complicated. The same holds true for all items, thus we decide to change our perspective of them. Instead of viewing tasks as unique items, we look at them depending on the order in which they come in the test, being a first task, second one, etc. To illustrate what we mean you can check Table 4.1. This is how we would like the data in our event log to look like. Every event can be connected to a specific case and every activity done by the user is connected to an item, where items are specific exercises from the test. An example trace of the event log of Table 4.1 is: *<Register, Log in, Begin test, Begin 1st task, Begin 1st sub-task, Give Feedback, End 1st task, ..., Begin last task, Finish last task, end test>*.

We take a couple of steps in order to transform and re-label the labels such that we make the event log suitable for process discovery algorithms. The first step is to disregard the system events and leave only the ones that are related to the execution of a task. Now we are left with only the user-initiated events (check Section 3.2 to see which are the exact events which are user-initiated ones). We need to recognize for each user which tasks are the *first*, *second* until the last task. We can use the information in the "Item" column

Case ID	Timestamp	Label	Item
70	11-01-2020 15:25:35	Register	Register
70	11-01-2020 15:48:40	Log in	Log in
70	11-01-2020 15:51:46	Begin test	Begin test
70	11-01-2020 15:53:58	Begin 1st Task	item 20a
70	11-01-2020 16:30:04	Begin 1st sub-task	item 20a
70	12-01-2020 09:00:35	Begin 2nd sub-task	item 20a
70	12-01-2020 09:34:41	Feedback Given	item 20a
70	12-01-2020 09:40:00	Finish 1st task	item 20a
70	14-01-2020 22:00:12	Log in	Resume test
70	14-01-2020 22:03:53	Begin 2nd task	item 85
70	15-01-2020 00:12:31	Feedback given	item 85
70	15-01-2020 02:44:30	Finish 2nd task	item 85
70	15-01-2020 03:57:11	Begin 3rd task	item 42
70	15-01-2020 04:30:10	Begin 1st sub-task	item 42
70	15-01-2020 07:00:00	Log in	Resume test
70	15-01-2020 07:02:10	Begin 2nd sub-task	item 42
70	15-01-2020 08:08:10	Finish 3rd task	item 42
70	15-01-2020 08:10:01	Finish test	End test
80	19-01-2020 10:00:10	Register	Register
80	19-01-2020 10:03:10	Log in	Log in
80	19-01-2020 10:08:11	Begin test	Begin test
80	19-01-2020 10:09:55	Begin 1st task	item 20a
80	19-01-2020 10:32:10	Begin 1st sub-task	item 20a
80	19-01-2020 11:01:20	Begin 2nd sub-task	item 20a
80	19-01-2020 11:17:08	Give Feedback	item 20a
80	19-01-2020 11:20:10	Finish 1st task	item 20a
80	19-01-2020 11:23:19	Begin 2nd task	item 17
80	19-01-2020 12:56:23	Give feedback	item 17
80	19-01-2020 13:00:09	Finish test	End test

Table 4.1: This is the format we want our data to take. The "Label" column should indicate the progress of a developer in the test. Every task which the user engages in is documented and each sub-task can be distinguished within cases using the item number.

to be able to make a connection between the candidate for re-labelling the corresponding position it appears in. We filter the first and last occurrence of each item, for each case which leaves us with the moment a developer begins an exercise and the one where he submits it, disregarding the moments they

give feedback. On Table 4.2 we have highlighted the events which we are left with after filtering. We are left with 2 events for each item per case and the maximum number of exercises given to a developer is 6. This implies the maximum possible events in a case is 6 multiplied by 2 events for a task which is 12 events in total. If the events are sorted by timestamp within cases, then we can re-label them based on the order in which they appear in a trace. In other words, the first event of a case will always be "Begin 1st Task", the second - "End 1st Task", the third - "Begin 2nd Task" and so on until the last possible task.

Case ID	Timestamp	Label	Item
70	11-01-2021 12:00:00	Log in	Logged in
70	11-01-2021 15:25:35	Task Downloaded	item 20a
70	11-01-2021 15:48:40	Task Downloaded	item 20a
70	11-01-2021 16:10:46	Task Downloaded	item 20a
70	11-01-2021 16:13:58	Feedback Given	item 20a
70	11-01-2021 16:30:04	Task Downloaded	item 14
70	12-01-2021 09:00:35	Task Downloaded	item 14
70	12-01-2021 09:34:41	Feedback Given	item 14
83	03-02-2021 17:11:01	Log in	Logged in
83	03-02-2021 22:00:12	Task Downloaded	item 85
83	03-02-2021 22:40:53	Task Downloaded	item 85
83	03-03-2021 08:12:31	Task Downloaded	item 85
83	03-03-2021 08:44:30	Feedback Given	item 85

Table 4.2: The events which are highlighted mark the beginning and the submission of the items they correspond to.

On Table 4.3 you can see the format our data has after transformation and changing the labels. All this information can be used to give feedback to GrepS to improve the user experience. Many companies use the software as an additional opinion for recruiting purposes. Furthermore, the test could also serve in the B2C case for developers who want to know their level of programming capabilities.

4.2 Identifying Break Times

To get an understanding what is the actual time people spent on the exercises, the break periods have to be identified. The difficulty of doing this comes from the fact that there is not a specific event which marks that the user is

Case ID	Timestamp	Label	Item
70	11-01-2021 15:25:35	**Begin 1st Task**	item 20a
70	11-01-2021 16:10:46	**End 1st Task**	item 20a
70	11-01-2021 16:30:04	**Begin 2nd Task**	item 14
70	12-01-2021 09:00:35	**End 2nd Task**	item 14
83	03-02-2021 22:00:12	**Begin 1st Task**	item 85
83	03-03-2021 08:12:31	**End 1st Task**	item 85

Table 4.3: This is how the data looks like after transformation.

taking a break. Furthermore, an event which is connected with the logging out of the user from the system is not present either. The only label which is connected in a way with the user being idle are the "Log in" events which appear in stages of the test different from the beginning. We assume that a "Log in" event that occurs some hours after an exercise is started signifies that the user was taking a break. However, it is also possible the case that when the user refreshes the web page, this activity is also saved in the event log with label "Log in". We hypothesize that the time differences between the different exercises will estimate the periods when users are being idle.

Chapter 5

Evaluation

5.1 Objective

In this project we focused on the problem of reformulating the labels of activities in an event log which have duplicated names. This re-labelling is essential if we want to apply a process discovery algorithm on the data which builds a model of all the touch points which a customer has with the product of GrepS. The aim of creating a model is to get a better understanding of the customer journey which a developer undergoes in the programming evaluation software.

5.2 Setup

The provided data from GrepS was in an unstructured form and needed to be transformed such that it is suitable for applying a process discovery algorithm on it. We formulated our strategy around the fact that the items in the test come in different order for different users. This made us decide to focus on building a sequential-order task model. This means we abstract ourselves from the notion of specific exercises and view them from the order in which they come in the test (first task, second task, etc.). We view the process from the perspective of the user, hence exclude system activities. We compare the initial DFG which contains all 27 activities, including system ones, and a DFG which includes only the start and end of the tasks.

The two constructed DFGs differ in many ways from one another. Firstly, the amount of nodes the task-specific map has is twice as less than the initial one. Secondly, the every event in the task-specific map has its own node, there are no aggregated events such as the ones in the other map. Next, the elimination of system events and leaving only those who are connected to

the user gives a clearer picture about the user behavior. Furthermore, the smaller amount of events which are left after disregarding users who have not began the test after logging in helps to give a more realistic idea what happens during the execution of the tasks which is of interest for the project.

5.3 Execution

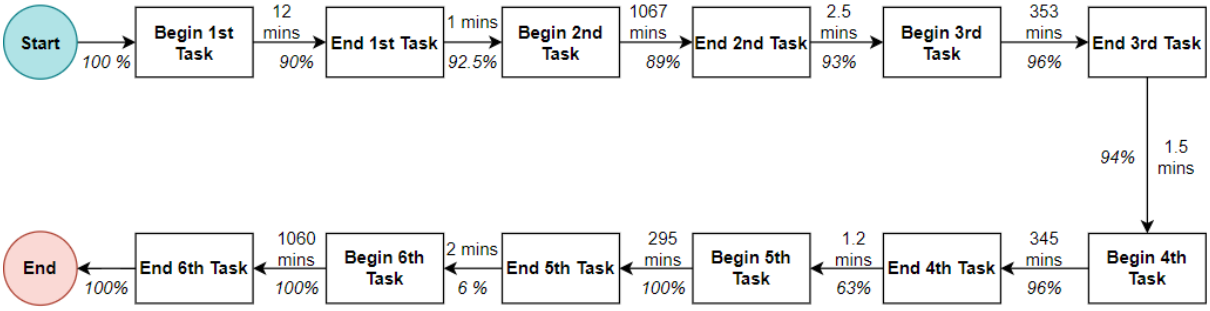
In the beginning, we construct the modified DFG on Figure 3.2 in ""Rstudio" using all of the data we have an aggregating some of the duplicate label events. There are some major steps which need to be executed that transform the data into the appropriate event log format. Firstly, the data needed to be converted such that it is in a dataframe format, i.e. with columns and rows. Secondly, we identified the minimum set of columns to construct the event log, namely: $caseID$, $timestamp$, $activity\ label$. From this event log we build an initial DFG which has many unclear connections between events and also the presence of numerous activities included in it make it hard to read and understand whatsoever. We filter out only the task-related events that are initialized by the user. After that, we re-label the activities to correspond to the order in which they come in the test. This means the first event for a developer is the beginning of the first task, the second event is the end of the first task, third event is beginning of second task and so on. The maximum tasks which a user could get in a test is six, thus the maximum number of events for a user is $2 * 6 = 12$ events. From here we iterate over the data and build a rule which re-labels the activity on every row based on the number in which it appears in a case.

5.4 Results

Task-specific DFG

The DFG on Figure 3.2 has too many events included which makes it difficult to gather information from it. Thus, we focused on the Task node from the DFG. We simplified it by removing system-initiated events and mainly focusing on the execution of the tasks and reached a new DFG which is shown on Figure 5.1. We abstract ourselves from the specific item number and look at the tasks as a sequence of items. This means we represent the events in the following way: *<initialize first task, end first task, initialize second task, end second task, etc.>*. Most of the developers did only 5 tasks. The maximum number which only a few of them have had to done is 6 tasks. It seems the most time spent is on the second task- around 17 hours and the task where

Figure 5.1: A graph filtered from the DFG produced from the transformed data. It is clearer to see the time spent as a whole on tasks and notably the time between tasks is relatively small - around 2 minutes. The users which do not finish a the test are sent directly to the "End" node and the process just finishes for them. Check Appendix A to see the originally produced DFG



people spent the least time is the first one- 12 minutes. The average time for the other tasks is around 5-6 hours. However, the process map does not account for breaks, they are a separate item for investigation.

Something to note is that the time between the shift from one task to another is on average 1-2 minutes. One would expect that after doing a certain task, a developer should take a break before continuing, but it seems this is not the case. This time would barely be enough to even fill the questionnaire, thus we assume that there is some kind of auto start after the submission of a task. The automatic transition leads us to the thought that if developers took breaks, then they would not be counted towards the time between the tasks, but would be added towards the time of the next task which is given.

Analysis of Item Positions

Initially, we assumed each different item can come in every possible position in the test, however when we looked more carefully into the data we realized our assumption might be false. If you look at Table 5.1 which has information on the frequency of each item at each possible position, you can see that for every item there is some position which dominates all the other ones. For the 1^{st} exercise "item 20a" is given 100% of the time. The same holds for "item 38" and "item 14" which respectively fill the first and the second position, with a few occurrences which are exceptions. For the 3^{rd} position "item 36"

is given the most (with "item 85" appearing a couple of times). We decided to look more carefully into the time distributions for the different exercises positions and they can be observed on the boxplot on Figure 5.2. A boxplot is a suitable visualization, since it can illustrate symmetry and skewness of the data.

Item number	**Most frequent position**	**count**	**Total times the item is given**
item 20a	1st pos.	19	19
item 38	1st pos.	24	25
item 14	2nd pos.	32	34
item 36	3rd pos.	22	31
item 85	4th pos.	17	28
item 42	5th pos.	9	15
item 24	5th pos.	2	2
item 34	5th pos.	2	2

Table 5.1: The frequency of every item at each possible position in the test with the total number of times the item was given.

In the end, by making simplifications and assumptions about the customer journey, we managed to re-label duplicated task labels and build a CJM which gives a better illustrations of the process which developers undergo. From observing the CJM, we deducted that on average developers spend the most time on the second task.

We decided to also look at the time spent on specific item numbers and check if there is some relation between it and the time spent on tasks as a sequence. On Table 5.3 you can see the average time people have spent on every possible task from 1 to 5. Task position 6 is not included because only 1 person had to do more than 5 tasks and we cannot make sound conclusions based on such a small sample. Additionally, you can see the item which has appeared the most for every separate position. We see that for the 2^{nd} task on which the most time was spent, item 14 had appeared the most times. The average time for the 2^{nd} one is 1060 minutes or about 17.6 hours and on Table 5.2 you can see that for item 14 the average time spent is 240 minute. The difference is more than 4 times, which would imply that there could be some outliers which skew so much the time on this second task. The task that took less time from the people is number 3 and the average time spent on it is approximately the same as the average time of item 36 which is the most frequently appearing item at that corresponding position. The 4^{th} task is interesting because it took around 348 minutes (5.8 hours) to complete, however item 85 which appears most often at the position, took around 13020 minutes (217 hours). We checked the relative frequency of item 85 on the 4^{th} position and it occurred more than 60% of the cases, therefore the skew

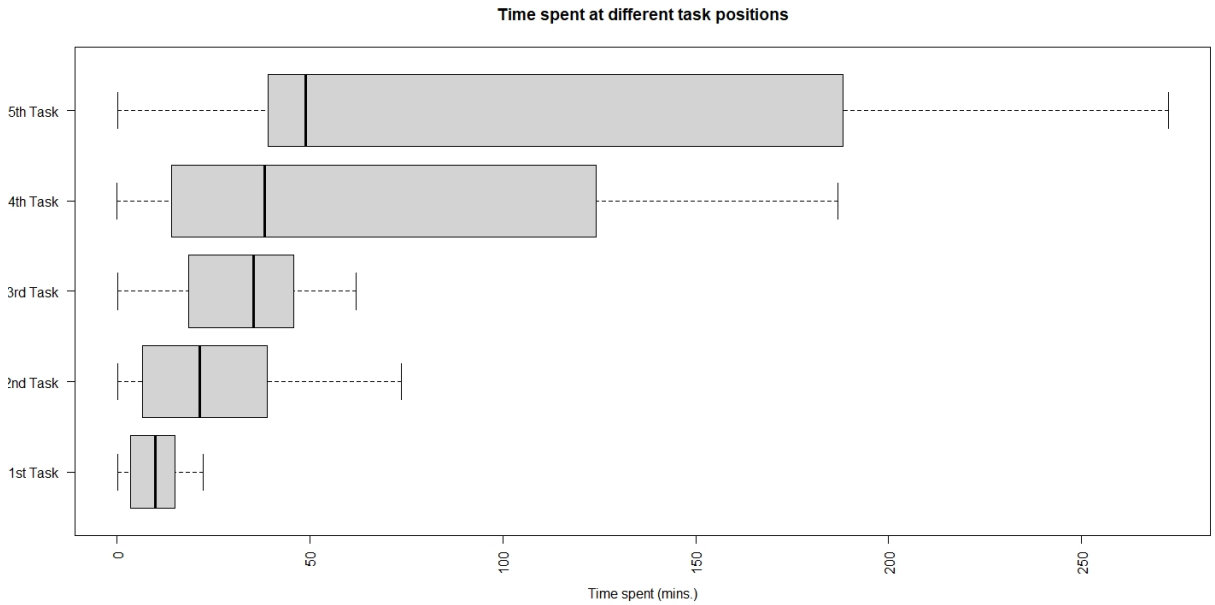


Figure 5.2: The time distribution for every task position in minutes.

in the average time for task number 4 should have been affected by item 85. However, when we observed more carefully the cases where item 85 was in this position, we saw that the biggest difference which is present is about 620 minutes (10.3 hours). Thus, we can say for item 85 that the average time which is indicated in Table 5.2 is affected by some outlier and most certainly the it includes breaks. If we check the boxplot on Figure 5.2, we can confirm that the time for position 4 is skewed to the right.

Item number	Average time spent (mins.)
item 85	13020
item 24	774
item 34	714
item 36	408
item 14	240
item 42	144
item 38	60
item 20a	20

Table 5.2: The average time spent per specific **item** in the test in minutes.

As for position 5, it takes less than 5 hours (294 mins.) and the item

corresponding to it is 42 and its average is about twice as less - 144 minutes. The boxplot shows that it is skewed to the right, hence outliers are affecting the distribution. Finally, the least time is spent on the 1^{st} task, around 12 minutes, and item 38 along with item 20a are given equally many times. Item 38 takes around 60 minutes to solve and item 20a about 20 only. Judging by the time, it seems people did not have much trouble with items 38 and 20a. It took them considerably less time to solve the 1^{st} exercise. The distribution for this first position seems normally distributed, therefore we can say that reality is not far apart from what is shown on the plot. All of the others seem to be equal in difficulty if we have to determine from the time. The skew for the 2^{nd} position is probably formed, because someone took a couple of days break from the test and that influenced the result.

Position number	Average time spent (mins.)	Most frequent item at position
2nd Pos.	1060	item 14
3rd Pos.	355	item 36
4th Pos.	348	item 85
5th Pos.	294	item 42
1st Pos.	12	item 38/item 20a

Table 5.3: The average time spent on a specific **position** in the test and the item which appears the most at each position.

Feedback Analysis

After every submitted question, the developer is asked to answer a questionnaire which gives feedback to GrepS. The questions which are asked can be seen on Figure 5.3.

Each of the questions has a set of 5 possible answers ranging from "Strongly Agree" to "Strongly Disagree". We decided to look at the overall feedback and compared the compared the answers based on the frequency of answers given to each question. The barplot on Figure 5.4 compares the frequency of each answer to each of the 3 questions posed by the questionnaire. Looking at the chart more than 30% of the people agree that it was easy for them to understand the task (question 1) and 25% strongly agreed with the same premise, whereas less than 20% disagree. Overall, more than 45% think they performed well on the tasks and less than 30% disagree that they did well. Finally, around 50% share that the tasks were interesting to solve and only 10% are on the opposite opinion.

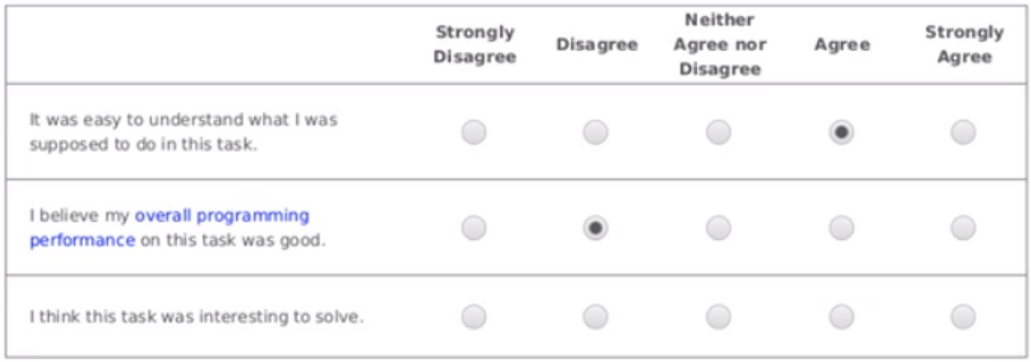


Figure 5.3: Questions asked to developer for feedback after every task

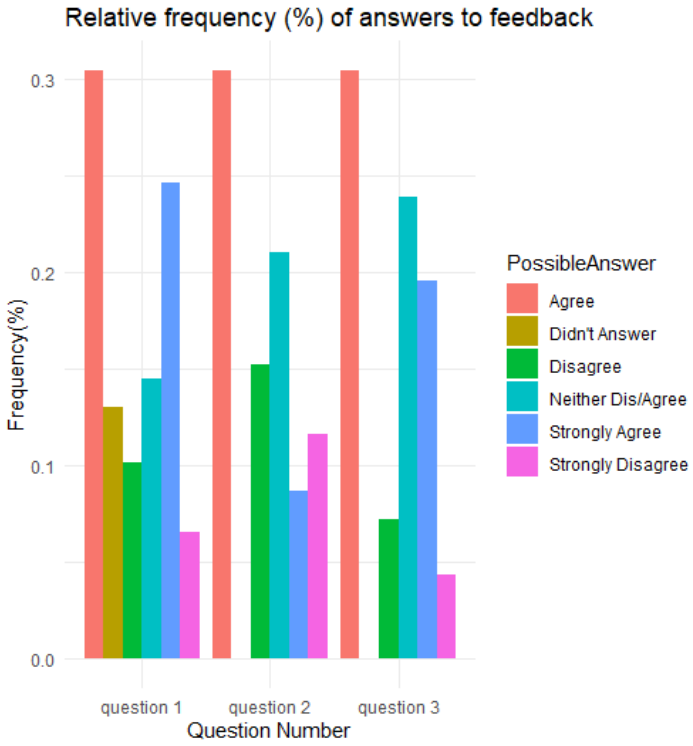
We also looked at the feedback of the questionnaires at an individual task level. The motivation for doing this is that it can help reveal flaws in the tasks which can be improved. There is 1 task, namely "item 42", which only 30% agree that they understood what was wanted to do and 40% disagree with the statement. Again, for the same task only 28% believe they did well against 56% which were not so confident with their performance. These results suggest that "item 42" is the most troublesome exercise, nevertheless, there are 2 tasks - "item 36" and "item 38" for which there were only 2 respondents. Because of the little amount of information about these tasks, we cannot derive whether one of the is the harder or not than the rest.

5.5 Discussion

There are further matters to investigate in the task process. It turns out that the time distribution of the tasks has to be handled with care if we want to calculate the real time people spent on exercises. We assume that the break times influence greatly the skewness of the time distribution of our data, thus they are crucial if we want to understand the reality behind the process. The first step would be to include "Log in" events in the process model, however our approach does not allow to account for them, since they do not have a pre-determined position in the process, rather it changes depending on the specific case. Another drawback of the method we applied is that we do not account for sub-exercises which is valuable information on its own. The work we have done lays the foundation for potential future work like investigating further the aforementioned flaws of the project.

We assume the tasks on which people spend more than 5 hours include the breaks in them as well. If the time for breaks is identified and disregarded

Figure 5.4: Barplot which compares the relative frequency of each answer to all 3 questionnaire questions



we will probably have even a more accurate DFG. Furthermore, we do not account for tasks which have several sub-tasks in them, instead we consider them as one whole task. Maybe, if we look at specific sub-tasks we can find more insights about the user behavior. Unfortunately, to reach sounder and more solid results, we would need more data to work on. The task-specific process model is built on the journeys of only 67 developers out of all the 202 which are included in the data. Around 130 people did not engage into the test which is a lot of information rendered useless. However, if GrepS would include more data about developers which have undergone the entire set of exercises, they might find useful insights about their system and identify steps at which people face problems. If they were to do the exact opposite and focus on system events and disregard user ones they could find phases at which their system has issues. This would help them find out what to improve in order to make the user experience more pleasant for their customers.

Evaluating the complexity of the exercises based only on the time spent on them is one way of doing this. Something which could be added for a more precise evaluation are the rasch scores of the participants at each question

and observing the average change in performance. The combination of the scores with the time used will be a better estimator to the true difficulty of the test.

In the beginning we aimed at re-labelling every task node such that the different components of it can be distinguished as separate events but discovering only its beginning and end was the furthest we managed to reach. More work is needed such that every break time, sub-task, score update and feedback are added as unique activities. We could apply trace clustering to partition the users in different clusters depending on some attribute of our choice. One option is to use the feedback self-evaluation scores and divide the participants in some way. Another proposal would be to cluster the cases based on the time they have spent and compare different attributes between groups such as scores received by the system. This may be a starting point for the process of deducing some dependence between time spent and grades given to developers.

Chapter 6

Conclusion

In this project we worked with data from a company called "GrepS" who developed a software for evaluation of programming skills and gathered data about the interaction of users with it. The data contained duplicate labels for many of the different events and there were many nested activities whose sub-activities needed to be recognized as separate ones. In the beginning, we looked at the whole process from a high-level perspective, just to end up finding it has a *spaghetti-like* view.

We decided to focus on the execution of the tasks because they are the main component of a test and build a DFG which gives a representation of the sequence of the tasks through which users of the test undergo. By re-labelling every task activity we mapped them to the beginning or end of a sequence of 6 tasks and constructed the DFG on Figure 5.1 and found out that 99% of the people have 5 tasks, with only 1 case having 6 tasks to solve. Furthermore, we derived that on average every task takes approximately the same time, besides the first one, which takes around 12 minutes to complete. We compared time spent per position and the time spent for specific items to check if there is a connection between the two. It turns out it is hard to make a connection between the 2^{nd} task on which people spent the most time and the time of item 14 which appeared the most in this position. These gaps come from the break time. Unfortunately, evaluating the tasks based only on the time people have spent on them is misguiding, due to the same reason of not including the time for breaks in it. Breaks were not identified because they happen at random times or more precisely, in times which the developer wishes to rest. Furthermore, we know that the transition from one exercise to another happens automatically, therefore break time is counted towards the entire time spent on a task. Identifying breaks and being able to account for them will weave more precise results on the time spent per task and consequently, evaluating the difficulty of a task. It turns out that we

need a different approach to identify breaks, since just looking at the time distribution and time difference between exercises is not enough. If breaks were included in the event log by a separate activity, then a process model algorithm could be applied to the data and yield some insightful model.

Bibliography

[1] Petri net explained kernel description. https://everything.explained.today/Petri$_n$et/. *Accessed* : 2022 − 01 − 30.

[2] Gunnar R Bergersen, Dag IK Sjøberg, and Tore Dybå. Construction and validation of an instrument for measuring programming skill. *IEEE Transactions on Software Engineering*, 40(12):1163–1184, 2014.

[3] Gaël Bernard. *Process Mining-based Customer Journey Analytics*. PhD thesis, éditeur non identifié, 2020.

[4] RP Jagadeesh Chandra Bose. Process mining in the large: preprocessing, discovery, and diagnostics. 2012.

[5] R.P. Jagadeesh Chandra Bose, Ronny S. Mans, and Wil M.P. van der Aalst. Wanna improve process mining results? In *2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM)*, pages 127–134, 2013.

[6] Joos CAM Buijs, Boudewijn F van Dongen, and Wil MP van der Aalst. Towards cross-organizational process mining in collections of process models and their executions. In *International Conference on Business Process Management*, pages 2–13. Springer, 2011.

[7] Javier de San Pedro and Jordi Cortadella. Discovering duplicate tasks in transition systems for the simplification of process models. In *International Conference on Business Process Management*, pages 108–124. Springer, 2016.

[8] Chenchen Duan and Qingjie Wei. Process mining of duplicate tasks: A systematic literature review. In *2020 IEEE International Conference on Artificial Intelligence and Computer Applications (ICAICA)*, pages 778–784. IEEE, 2020.

[9] Sander JJ Leemans, Erik Poppe, and Moe T Wynn. Directly follows-based process mining: Exploration & a case study. In *2019 International Conference on Process Mining (ICPM)*, pages 25–32. IEEE, 2019.

[10] Xixi Lu, Dirk Fahland, Frank JHM van den Biggelaar, and Wil MP van der Aalst. Handling duplicated tasks in process discovery by refining event labels. In *International Conference on Business Process Management*, pages 90–107. Springer, 2016.

[11] Heidy M Marin-Castro and Edgar Tello-Leal. Event log preprocessing for process mining: A review. *Applied Sciences*, 11(22):10556, 2021.

[12] Wil Van Der Aalst. Discovery, conformance, enhancement of business processes. In *Process mining*, page 13. Springer, 2011.

[13] Wil Van Der Aalst. Data science in action. In *Process mining*, pages 3–23. Springer, 2016.

[14] Wil MP Van der Aalst, Vladimir Rubin, HMW Verbeek, Boudewijn F van Dongen, Ekkart Kindler, and Christian W Günther. Process mining: a two-step approach to balance between underfitting and overfitting. *Software & Systems Modeling*, 9(1):87–111, 2010.

List of Figures

List of Tables

Appendix A

Original DFG

Figure A.1: The original Directly-follows graph produced in 'processanimate' library of R programming language. The difference between this and the one on Figure 5.1 are the edges which connect all of the nodes to the end one.