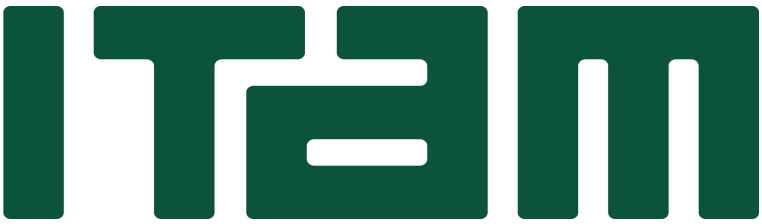
Instituto Tecnológico Autónomo de México



# ANÁLISIS DE PATRONES DE TRABAJO DE PROGRAMADORES CON DATOS DE USO DE HERRAMIENTAS

TESIS

QUE PARA OBTENER EL TÍTULO DE

MAESTRO EN CIENCIAS EN COMPUTACIÓN

PRESENTA

LUIS CARLOS CRUZ DÍAZ

ASESOR

DR. VÍCTOR MANUEL GONZÁLEZ Y GONZÁLEZ
DR. ROMAIN ROBBES

MÉXICO, D.F. 2016

"Con fundamento en los artículos 21 y 27 de la Ley Federal de Derecho de Autor y como titular de los derechos moral y patrimonial de la obra titulada "**ANÁLISIS DE PATRONES DE TRABAJO DE PROGRAMADORES CON DATOS DE USO DE HERRAMIENTAS**", otorgo de manera gratuita y permanente al Instituto Tecnológico Autónomo de México y a la biblioteca Raúl Baillères Jr., autorización para que fijen la obra en cualquier medio, incluido el electrónico, y la divulguen entre sus usuarios, profesores, estudiantes o terceras personas, sin que pueda percibir por tal divulgación una prestación"

LUIS CARLOS CRUZ DÍAZ

_____

Fecha

_____

Firma

**Abstract**

Este documento presenta una plantilla para usar en las tesis y tesinas del ITAM. Se provee de manera gratuita y sin ninguna responsabilidad bajo la licencia *creative commons BY-SA 3.0*.

**Abstract**

In this work we present a template for thesis and titulation works presented at ITAM. It is provided freely and without any responsability under the *creative commons BY-SA 3.0*.

# Contents

**1. Data and Methodology** **1**

1.1. Description of the data . . . . . 1

1.1.1. Eclipse Usage Data Collector . . . . . 1

1.1.2. Codealike and ABB . . . . . 2

1.2. Methodology . . . . . 3

1.2.1. Preprocessing . . . . . 3

1.2.2. Classification of events . . . . . 4

1.2.3. Transformation . . . . . 6

**2. Results** **9**

2.1. Activities and working patterns . . . . . 9

2.1.1. Characterizing working sessions . . . . . 9

2.1.2. Using focus data . . . . . 10

2.1.3. Identifying types of activities . . . . . 11

**References** **16**

# List of Figures

2.1. Histogram of productivity per hour according to the focus data. . 11
2.2. Histogram of productivity per day according to the focus data. . 11
2.3. Evolution of the activities throughout the sessions. . . . . 15

# List of Tables

1.1. Attributes of the UDC dataset. . . . . 2
1.2. Attributes of the ABB dataset. . . . . 3
1.3. Description of the detailed classification of events. . . . . 5
1.4. Statistics of sessions in UDC and ABB. . . . . 8

2.1. Activities found via clustering. . . . . 14

# Chapter 1

# Data and Methodology

## 1.1. Description of the data

In this section we introduce the data with which we worked in terms of the source, attributes and special characteristics. Although they come from different contexts, they represent the same kind of information.

### 1.1.1. Eclipse Usage Data Collector

The Usage Data Collector (UDC) dataset is a large compendium of information about interaction data from users of Eclipse, collected from December 2008 to August 2010, with the intention to keep track of how programmers are using the IDE. The framework listens to the events triggered by the user or the system, such as: edition and navigation commands; the startup of a plug in; or the closing of the platform. To be more specific, UDC collects information about loaded bundles, commands accessed via keyboard shortcuts, actions invoked via menu or tool-bars, perspective changes, view usage and editor usage. The UDC is a large dataset that contains information of around 1,800,000 users, and has a total of 2,323,233,101 unique rows with 5 attributes each. Table 1.1 shows a description of the attributes.

We used the pre-processed version of the data that is published on Google

Table 1.1: Attributes of the UDC dataset.

| *Attribute* | *Description* |
|---|---|
| UserId | Unique number that identifies a user |
| What | The action of the event (deactivated, activated, opened, executed, etc.) |
| Kind | What kind of event was executed (workbench, view, command, etc.) |
| BundleId | Description of the event's package |
| BundleVersion | Version of the bundle's event |
| Description | Description of the event |
| Datetime | Date and time in UNIX format |

BigQuery, by Murphy-Hill et al. [6]. This is an alternative version of the original UDC dataset, which is cleaned and preprocessed, so that the transformation phase is simpler and focused on our needs. Due to the magnitude of the dataset we only worked on a fragment of it.

We took a subset of the data from 1,000 random users. We delimited the query to obtain only those events dispatched by the user, ignoring system events. We also ignored the bundle version and transformed the UNIX date into a more legible datetime format. From this query we extracted 4,321,349 unique events, which are around 0.18% of the whole dataset. Also, we only selected the attributes UserId, What, Kind, Description and Datetime.

### 1.1.2. Codealike and ABB

Codealike [1] is a tool that monitors the activity of the user and later offers analytics and insight about the programmer's productivity and working patterns. This tool is installed in the IDE (Eclipse and Visual Studio) and listens the events executed by the user similar to UDC. It captures almost the same kind of events like edition, navigation and tool usage. Shortly after the capture of events, the user can observe information derived from his activity through a website.

Corvalius (the Argentinian company that created Codealike) is collaborating with ABB, a multinational corporation operating mainly in robotics, power and

automation technology areas. The ABB's Software Engineering Research Group is using Codealike to obtain information from its developers with the objective of improving productivity and software quality. We had access to a dataset corresponding to the monitoring of 87 programmers between May and October of 2015 comprising 15,597,697 unique events.

Table 1.2: Attributes of the ABB dataset.

| *Attribute* | *Description* |
|---|---|
| Username | Unique identifier of the user |
| Timestamp | Date and time of the execution |
| Event | Identifier of the event's description |
| Category | Unique identifier of the event's category |

The actual description of the events is stored on a different file, so we use the values of the Category and Events attributes to extract the proper description. Aside from that, we use all events and attributes of the dataset. From hereafter, this dataset will be referred as ABB.

## 1.2. Methodology

We followed the steps described by the *Knowledge Discovery in Databases* process [2], a set of iterative steps composed of Selection, Preprocessing, Transformation, Data Mining and Interpretation. The Selection was covered by the last two sections and the following sections will describe the tasks performed during the Preprocessing and Transformation steps. In the latter Chapters we will give details about the Data Mining and Interpretation steps.

### 1.2.1. Preprocessing

During the Preprocessing stage we performed mainly two tasks: data cleaning and classification. The cleaning tasks are trivial, for the data is in overall good shape. However classifying the events is a complicated task that requires careful inspection and multiple iterations. The tasks involving data cleaning diverge between datasets, so we give details about this step separately. But

the classification of events follows the same process for both datasets and is described afterwards.

#### I. UDC

First, we added a field with the duration of every event (time elapsing between one event and the next one, used to determine where interruptions take place and sessions end) and an ID to identify the different working sessions present on the data. For the latter, we sorted the data by UserId and Timestamp. This was required because, by default, the user's data is mixed and we need it not only chronologically correct but also sorted by users to tag the working sessions of every user without interferences. We also filled the description for the events that indicate the activation or deactivation of the Eclipse's workbench, and they were the only ones that had this issue.

#### II. ABB

The Preprocessing for ABB is also simple. As with UDC, we added a duration in seconds and an ID to every event. Then, we extracted the Description from a second dataset, according to the Event and Category, and created a new attribute. We changed the Description to lowercases and removed curly braces and other special characters.

An interesting feature of this dataset is that plenty of data seems to be somewhat systematic, with a certain event executed every 5 minutes. We do not know in what scenarios this kind of activity is recorded so we decided to remove the observations. Finally, we ordered the data by Username and Datetime.

### 1.2.2. Classification of events

To classify the events, we look into the description to see what it can tell us about the event. We worked with two kinds of classifications:

- A detailed classification to tag the nature of every event.
- A general classification to identify between Edition and Selection events.

Having two classifications will help us provide better answers to our research questions, for some of them require doing analyses in detail and others can be seen from a general perspective. The general classification is derived from the detailed classification, so we describe the latter first in the Table 1.3.

Table 1.3: Description of the detailed classification of events.

| *Classification* | *Description* | *Examples* |
|---|---|---|
| Edit-Text | Text edition events | Copy, Paste and Delete. |
| Text-Nav | Events executed when navigating around text. | LineUp, LineDown and LineEnd. |
| High-Nav | Navigation of high level, like around classes and views. | GoToDefinition, NextTab and GoToSuperclass |
| Debug | Events executed during debugging sessions. | StepInto and StepOver. |
| Search | Events executed when searching for objects and text. | Search, Find and FindReplace. |
| Refactoring | Events executed when restructuring code. | Encapsulate, Rename and MoveField. |
| Testing | When testing (e.g. unit tests) is executed through a framework. | ExecuteTest and TestResults. |
| Control | Execution of version control tasks. | Compare, ViewChanges and CommitChanges. |
| Clean-Build | Tasks performed by the IDE to build and execute a solution. | Build and Run. |
| File | Events executed during the management of files. | Open, Close and SaveChanges. |
| Tools | Execution of specialized tools and plugins. | DatabaseDesigner, Codealike and UIDesigner. |

To assign a a detailed classification we use the description. In both datasets the description has the format *path.to.class.ListenerClass*. It contains the path to the class that works as listener of the event and executes the required task. The packages and class name give out information about the nature of the event.

With that in mind, we created a set of rules that adds a classification to every event according to the name of the class and/or the name of the path. For example, in UDC we label as Text-Nav all the events that have *ui.edit.scroll* in

the description, and in ABB we label as High-Nav all the events whose class name is *NextTab*.

Once we added the detailed type to the events, we proceeded to assign the general type. This is easily performed by setting as Selection all the events except those who has Edit-Text, Text-Nav or Refactoring in the detailed type; this set of events describe the kind of activities performed when editing code, while the rest involve navigation around classes, views code and selection of elements in the IDE. After we assigned a general and detailed type to the events we followed to the next step of the process.

### 1.2.3. Transformation

From the transformation phase and on, the activities are the same for both datasets, with some changes in the parameters. The objectives of the transformation phase are creating sessions and decomposing them into chunks, or segments of smaller time than sessions.

First, we identify interruptions using the duration or time interval of every event. This is an important tasks and it is convenient to define the two kinds of interruptions we use in this work:

- *Interruption.* We define empirically an interruption as a pause of activity of duration $\geq$ 3 minutes. This is based on previous work where we observed that short interruptions lasted usually this long [3]. Shorter values would risk classifying periods of inactivity (such as a programmer reading source code) as interruptions.
- *Prolonged interruption.* Based on additional observations from this work, we defined a prolonged interruption as one lasting for more than 12 minutes. These thresholds are also supported by the Activity Theory models of Kaptelinin and Nardi [4]. This study presented work fragmentation at two different levels: actions and activities. Interruptions originated after a period of around three minutes of sustained attention to the previous action were considered when people were switching at the level of interactions with artifacts of people. Interruptions originated after a period of twelve minutes of sustained attention to a previous activity were consid-

ered when people were switching at the level of interactions with projects or topics.

To identify working sessions we look for interruptions as well, but only those that last for more than 4 hours. Any segment of activity surrounded by interruptions with that duration is labeled as a session. However, in order for the segment to be valid it must be of at least 30 minutes of duration.

Then, for every session we created several time series representing the execution of events by detailed classification. Taking the minute as time unit, we counted the number of events executed on every minute and created eleven time series, where the amplitude is the number of events. The interruptions were treated differently; they are also contained in a time series of its own but the amplitude is the duration of the interruption and every observation represents the minute of occurrence.

After that, the next step is to calculate a number of metrics for every session that measure the activity of the programmer. They can be seen from two perspectives, first the metrics for the characterization of interruptions:

- *Number of interruptions*: counts all the interruptions that occur in a development session.
- *Duration of interruption*: it is the time duration in minutes of the each interruption.

And the metrics that describe the productivity and activity:

- *Productive work time:* the duration of a development session, subtracting the duration of all the interruptions present in the session, to control for inactivity.
- *Number of edits per minute:* the total number of edits events, divided by the productive work time to control for length of the session. This is an indicator of user activity during the session.
- *Number of selections per minute:* it is the total number of selection events, divided by the productive work time. Also an indicator of activity.

- *Edit ratio:* the number of edits divided by the sum of edits and selections, as used by Kersten and Murphy [5]; an efficient developer spends less time exploring code and more time editing it.
- *Proportion of events:* it is the count of events by detailed type divided by the total of events executed in the session. There are a total of 11 values for this metric.

The last task of the transformation phase is to split the sessions into smaller activity frames to do analyses of the programmer's activity in detail. For that we split the time series of every session into time segments of 3 to 5 minutes of activity that we will call chunks. It was necessary to recreate the time series and metrics for the chunks. After this, every user has a set of sessions (with their respective time series) and every session a has series of chunks.

Table 1.4: Statistics of sessions in UDC and ABB.

| *Statistic* | *UDC* | *ABB* |
|---|---|---|
| Number of sessions | 6,405 | 1,182 |
| Number of observations | 2,848,270 | 2,449,227 |
| Number of users | 621 | 69 |
| Number of chunks | 43,769 | 23,624 |
| Avg. duration | 4.11 hrs. | 7.33 hrs. |
| Avg. productive time | 64.29 min | 166.08 min |
| Avg. of interruptions | 9.52 | 25.40 |
| Avg. duration of inte. | 18.94 min | 10.74 min |

# Chapter 2

# Results

## 2.1. Activities and working patterns

For the first part of this section we present a characterization of working patterns within small segments of time (3 to 5 minutes) and larger frames of activity (¿4 hours). We extracted the information from both datasets. The ABB data corresponds to full-time developers, so we assume that they follow certain working patterns and have common activities. For that, we expect to see an homogeneous behavior among the developers that we can easily characterize and find structures via machine learning techniques.

In contrast, the UDC dataset contains more variety and different profiles of programmers, so it might be troublesome to find specific patterns.

### 2.1.1. Characterizing working sessions

We defined a working session as a lapse of recorded activity surrounded by interruptions of large duration ($\geq$ 4 hours), to model a day of work. In this case, our definition of interruption is any lapse of time without events recorded, but based on observational studies we set a threshold of 3 minutes to label

From the transformation phase, every session is composed of time series and other attributes. There is a time series to represent the interruptions, where

the amplitude is the duration of every interruption, and 11 more series whose amplitude is the amount of events of every type per minute. Every data point of the time series is equal to a minute of activity in the data, meaning that all the time series have the same length. In average this length is of 121.61 minutes ($s = 147.81$). This is also refereed as productive time, or the total time that the programmer spent working on the IDE without being interrupted.

A lot of the time of a working session is wasted in interruptions, or time segments of at least 3 minutes without recorded activity. In average every session has 18 interruptions (sd=21.17) with an average duration of 10.78 minutes. As for the total time consumed in interruptions, the average is of 190 minutes (sd=137.19), which is more than half the average duration of a session.

### 2.1.2. Using focus data

Complementary to the dataset, we had access to the focus data of a smaller group of programmers from ABB. This dataset contains the focus level (a value between 0 and 1) per minute that Codealike uses in its tool as an inference of the concentration of the user. The focus is calculated according to two functions: one to increase the value when the programmer makes use of the IDE, and a second one as a decay function, that is applied when the programmer is inactive. Therefore, when the user is active the value decreases and the contrary when he is not using the IDE.

This can be used as a summarization of the activity the user and to get very general information. For example, the Figure 2.1 shows the total focus per hour from all the programmers. We can see that the 14 hours seems to be the more productive time, for is the hour with the higher value of focus or the time with more usage of the IDE. There is a decrease at the 16 hrs., and increases a again to later abruptly drop at 21 hrs. The activity is lower between the 21 and 8 hours.

With this information we can also see the evolution of productivity throughout the days of the week. The Figure 2.2 shows the sum of focus level (normalized) for every day of the week. There is not a great difference aside from the week-end, but there is a slight decrease the days Monday and Friday, and the most productive days are Tuesday and Thursday.

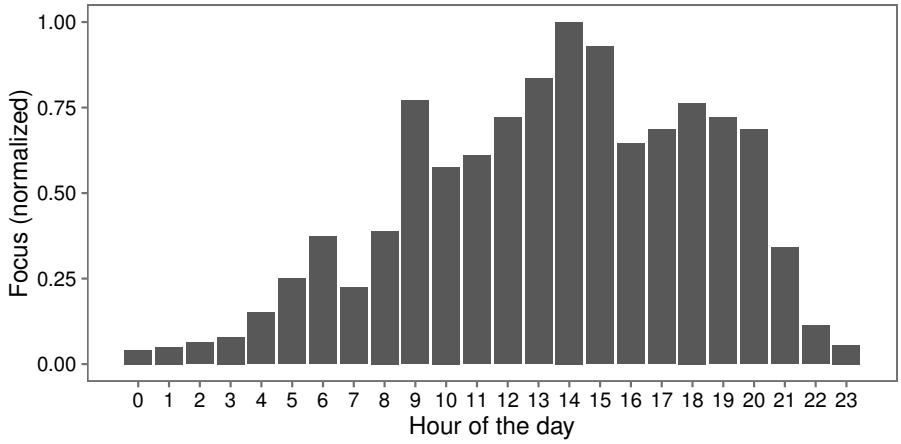



Figure 2.1: Histogram of productivity per hour according to the focus data.

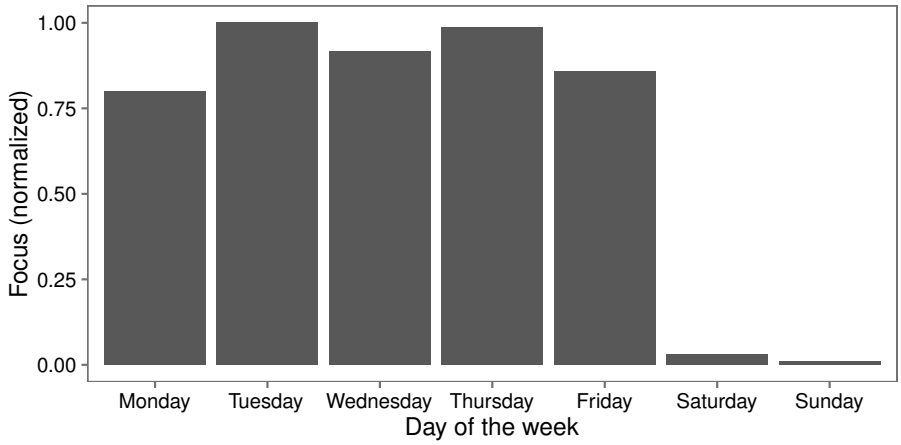



Figure 2.2: Histogram of productivity per day according to the focus data.

### 2.1.3. Identifying types of activities

In order to identify the types of activities the programmers perform, we split every working session into smaller activity time frames of 3 to 5 minutes of length. For each of this new chunks of the sessions we recreated the time series and calculated the proportion of events per type, which is the count of events of type $t$ divided by the total count of events in that chunk. After this, we obtained a total of 31,727 chunks.

The nature (or the type) of a chunk is defined by the amount of events of certain type in comparison with others. For example, for the activity of programming we expect a high proportion of events of edition, and probably also navigation around text and classes. The challenge is to know what threshold set to differentiate between two types of activities that might use the same kind of events, i.e. reading and programming could both use text navigation events. Also, if we try to define a set of activities according to experience and related work, we might be ignoring certain characteristic activities of this group of programmers or forcing one to exist. For this reasons we decided to implement clustering algorithms with the chunks, taking the proportion of events as the attributes. So, every observation contained 11 different attributes whose values are between 0 and 1.

As we didn't know how many types of activities there were, we discarded the algorithms that require the number of clusters as parameter, like K-means and K-medians. From the rest of algorithms, we chose Mean shift because: (1) in contrast with Affinity propagation (our other option) the time complexity is lower, and (2) the Mean shift technique performs one last step to unify very similar clusters. The importance of the second point will be discussed later.

An issue with Mean shift is that the size of the observations has a big impact in the results: a small population can offer more variety of clusters, and a big population could be clustered around one or two centers. Mean shifts creates the clusters by estimating a density function over a region of size given by the bandwidth, a parameter of the algorithm. So, the bandwidth also has an effect on the results.

It is difficult to know which are the appropriate parameters and sample size because, for we do not know what are the results. To tackle this, we took the following approach:

1. From the observations, select a random sample size $n$.
2. Execute the Mean shift technique with the random sample.
3. Store the resulting clusters.
4. Repeat $k$ times steps 1 to 3.

5. Filter the clusters to eliminate very close centers according to the Euclidean distance and a threshold $d$.

6. Discard clusters that only appeared in one execution.

The fifth step is done by measuring the distance of a cluster with the rest and selecting the group of clusters with a distance lesser or equal to $d$. Then, we calculated the median of the attributes of all the clusters in that group and substituted the group with the new cluster created from the medians.

With this approach we were able to observe very common clusters corresponding to frequent activities like programming and debugging, and also activities that only occur on special occasions or that are not very common within a working session.

With $n = 300$ and $k = 150$, the Table 2.1 shows the resulting clusters. We selected the corresponding activity by inspecting the closeness of the center to the attributes: the higher the value, the most events of that type. For example the Debugging activity has a value of 0.90 (from a range between 0 to 1) for the debug type; the rest of the activities have a value $< 0.20$ for that type of event. Next, we describe each of the activities discovered via clustering:

- *Debugging.* The most common activity has a big value for the debug type (0.90). It is composed of events meant to execute and control debugging sessions.

- *Programming.* It has the greatest values for the edit and text-nav types (0.62 and 0.24, respectively). It contains text edition and text navigation events and it's the second most common activity.

- *Navigation.* This activity has a high value for the high-nav type (0.85). It is often related to comprehension tasks.

- *Version-control.* It has a high value for the control type (0.89).

- *File-management.* It has a high value for the file type (0.86).

- *Tool-usage.* This activity represents the execution and usage of tools within the IDE with different objectives i.e. the management of databases,

Table 2.1: Activities found via clustering.

| *Activity* | *Amount of chunks* | *Users* | *Times found* |
|---|---|---|---|
| Debugging | 40.62 % | 59 | 150 |
| Programming | 35.53 % | 58 | 150 |
| Navigation | 18.67 % | 61 | 140 |
| Version-control | 1.85 % | 33 | 150 |
| File-management | 1.60 % | 45 | 2 |
| Tool-usage | 0.59 % | 28 | 149 |
| Search | 0.49 % | 32 | 14 |
| Refactoring | 0.34 % | 13 | 17 |
| Testing | 0.30 % | 17 | 58 |

user interface design and architecture diagrams design. It has a high value for the tools type (0.91)

- *Search*. It has the higher value for the search type (0.91) and to a lesser extent for edition and text-nav (0.10 and 0.11 respectively).
- *Refactoring*. It has a high value for the refactoring type (0.90) and to a lesser extent for edition, text-nav and high-nav (0.13, 0.15 and 0.13 respectively)
- *Testing*. It has a high value for the testing type (0.95) and to a lesser extent control and file types (0.35 and 0.19 respectively).

Once we had every chunk of a session labeled with the corresponding activity, we did an analysis of the evolution of the activities throughout a working session. For that, we split every session into 10 phases and counted the activities by type on each of the phases from all the sessions. The Figure 2.3 shows a line plot with the evolution of four selected activities (Programming, Debugging, Version-control and Testing) throughout the 10 phases of the sessions.

We can observe that Programming starts high and gradually decreases towards the end of the session, and the contrary occurs with Debugging, whose highest point is at the 9th phase. There is a negative correlation between these two activities ($r = -0.76$), possibly meaning that as the programmer advances

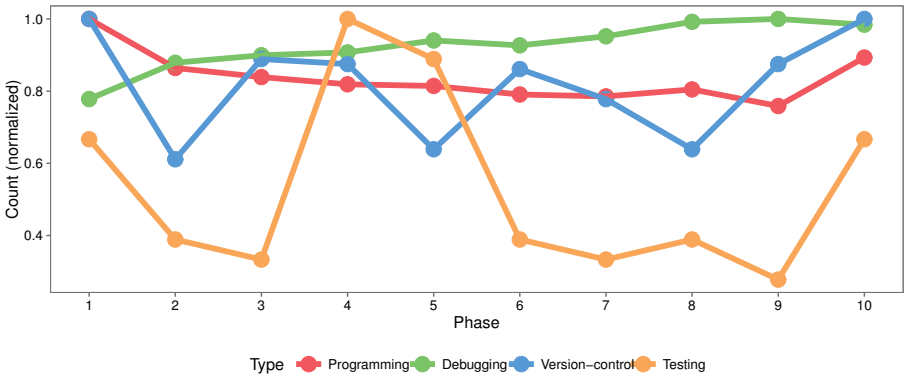



Figure 2.3: Evolution of the activities throughout the sessions.

with the task the programming necessities are reduced but increases the need for debugging and checking the correctness of the program. In contrast, the Navigation activity (not shown in the plot) has a positive correlation with programming ($r = 0.74$).

The Version-control activity seems to have an irregular behavior, being high at the beginning and ending of the session. The reason could be that at the beginning the programmer obtains the latest version of the program, while at the end he commits the final version after the working session. In the middle there are some peaks that might be different checkpoints that the programmer reaches as he has progress in the task.

As for the Testing activities, there s an obvious peak at the middle of the session and lower activity in the rest.

# Bibliography

[1] Corley, C. S., F. Lois, and S. Quezada (2015). Web usage patterns of developers. In *Software Maintenance and Evolution (ICSME), 2015 IEEE International Conference on*, pp. 381–390. IEEE.

[2] Fayyad, U., G. Piatetsky-Shapiro, and P. Smyth (1996). From data mining to knowledge discovery in databases. *AI magazine 17*(3), 37.

[3] González, V. M. and G. Mark (2004). "constant, constant, multi-tasking craziness": managing multiple working spheres. In *Proceedings of CHI 2004*, pp. 113–120.

[4] Kaptelinin, V. and B. A. Nardi (2007). Acting with technology: Activity theory and interaction design. *First Monday 12*(4).

[5] Kersten, M. and G. C. Murphy (2006). Using task context to improve programmer productivity. In *Proceedings of FSE 2006*, pp. 1–11.

[6] Snipes, W., E. Murphy-Hill, T. Fritz, M. Vakilian, K. Damevski, A. R. Nair, and D. Shepherd (2015). *Analyzing Software Data*, Chapter A Practical Guide to Analyzing IDE Usage Data. Morgan Kaufmann.