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

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

List of Figures

List of Tables

Chapter 1

Data and Methodology

Formal mathematics is nature's way of letting you know how sloppy your mathematics is.

– Leslie Lamport

Este trabajo presenta una plantilla para las tesis y tesinas del Instituto Tecnológico Autónomo de México para los usuarios de LaTeX. Nace de la necesidad de los matemáticos, actuarios e ingenieros (entre otras carreras) por utilizar un sistema de composición de textos adecuado para su trabajo de titulación. El objetivo es ayudar a la comunidad del ITAM a simplificar el proceso de escritura y edición de sus tesis, tesinas o casos. A continuación describimos a mayor detalle cada una de las partes de la plantilla.

1.0.1. Descripción de los archivos

1. El archivo **tesisITAM.cls** define una nueva clase de documento con el mismo nombre. Esta clase se basa en el tipo reporte, el cual se emplea comúnmente para reportes de trabajo, pequeños libros y tesis.

2. El archivo **macros.sty** define macros y operadores adicionales, como por ejemplo, el argumento mínimo argmin. Además provee un espacio para que el estudiante agregue sus definiciones propias.

3. Este archivo **introduction.tex** describe el funcionamiento de la plantilla y de los archivos.

4. El archivo principal **main.tex** es un ejemplo básico de un archivo de tesis para generar este ejemplo.

5. El archivo **portada.tex** contiene el código necesario para generar la portada. El archivo **derechos.tex** contiene el texto para cede de derechos de publicación hacia el ITAM.

1.0.2.   Características de la plantilla

La plantilla se basa en el documento tipo reporte. Todas las opciones de la clase *report* pueden ser utilizadas sin ninguna modificación (a4paper, 10pt, etc...). El espaciado del texto se establece a 1.33. Se eliminó la identación del párrafo, y el espaciado entre párrafos fue disminuido. Las sub-secciones se numeran utilizando números romanos. Se define un encabezado para cada página (salvo las que inician un capítulo) donde aparece el número del capítulo y el nombre de la sección.

I.   Paquetes importados

La plantilla utiliza los siguientes paquetes: *graphicx*, *amsopn*, *fancyhdr*, y *babel*. El paquete de manejo de idiomas babel es importado con los idiomas **english** y *spanish* por defacto. Además, se seleccionan las opciones de uso de punto decimal y de nombrar las tablas como tablas y no como cuadros.

II.   Opciones de clase

Se declara una opción adicional de nombre **tesina** para cambiar la portada a que se declare el trabajo como tesina.

III.   Campos para el autor

Para el autor, se define los siguientes campos: **title**,**author**,**degree**,**advisor**, y **year**. *Title* define el título del trabajo de titulación, este título se presenta

en la portada y al ceder los derechos de publicación. *Author* define el nombre completo del autor de la tesis. *Degree* se utiliza para establecer la carrera de la cual se va a titular, y debe incluir el texto completo (*i.e.*, Licenciatura en X o Ingeniero en C). *Advisor* recibe el nombre completo (con grado académico) del asesor, tal como se presentará en la portada. *Year* recibe el año de titulación para presentarlo en la portada.

IV. Generando la portada

El comando **maketitle** produce la portada con los campos previamente definidos. Utiliza el logo del ITAM, el cual se debe encontrar en el **PATH** de LaTeX(*e.g.*, en el fólder de Figures).

V. Cediendo derechos de publicación

El comando **publicationrights** imprime una página con el texto oficial de cede de derechos. Los nombres de la tesis (tesina) y del autor se obtienen de los campos previamente definidos.

VI. Añadiendo el resumen

Comúnmente, el resumen (o *abstract*) se debe escribir tanto en español como en inglés. Para esto, se define el ambiente **abstract** que recibe como parámetro un idioma. En el caso de que el idioma no sea ni inglés ni español, se recomienda que el idioma haya sido previamente importado como opción del paquete babel. En otro caso, el paquete imprimirá una advertencia.

VII. Citando a los grandes

La plantilla define un nuevo ambiente para las citas al principio del capítulo **chapterquote** que recibe dos parámetros: el autor y el texto. Para un ejemplo, referirse al principio de este capítulo.

1.1.   Licencia

Esta plantilla se distribuye bajo la licencia *creative commons BY-SA 3.0*. Esta licencia permite la modificación de cualquier aspecto de la plantilla siempre y cuando se respeten las siguientes condiciones:

1. Que se mantenga la atribución del trabajo original, es decir, que se mencionen los autores originales y su afiliación, tal como se hace en el archivo original.

2. Que todas las modificaciones se hagan públicas y libres de acceso, sin recibir ningún tipo de retribución por el uso o distribución de la plantilla.

1.2.   Autores

Chapter 2

Results

2.1. Activities and working patterns

For the first part of this section we present a characterization of working sessions at different levels, extracting the information from the ABB and UDC 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.

In contrast, 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 (8 hours), to model a day of work. However, in many cases the extracted sessions contained an interruption of moderate length in the middle, so we set a threshold of 4 hours to split this kind of sessions. At the end, we had 1,585 sessions with an average duration of 4.7 hours ($s = 3.58$).

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 weekend, but there is a slight decrease the days Monday and Friday, and the most productive days are Tuesday and Thursday.
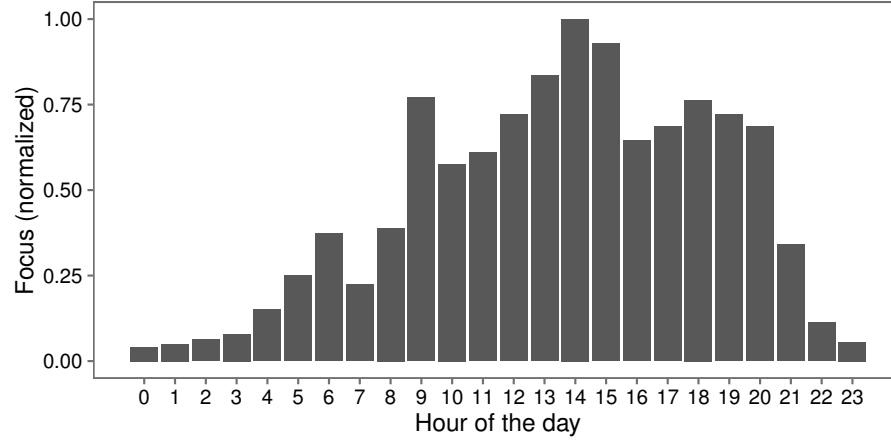
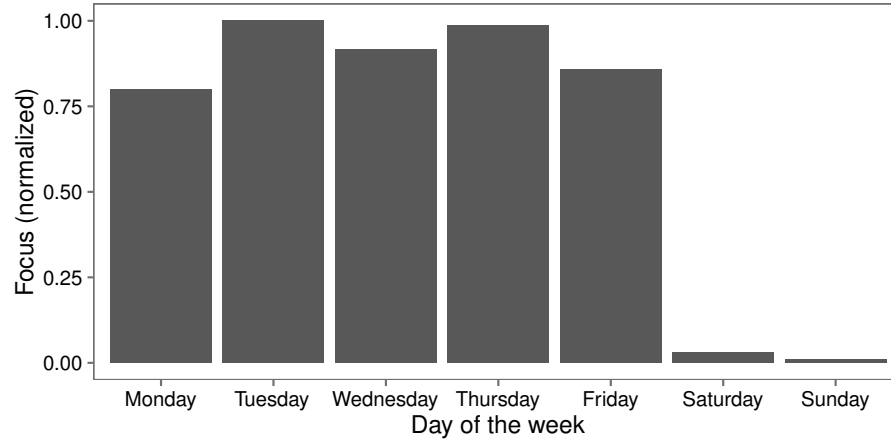Figure 2.1: Histogram of productivity per hour according to the focus data.

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.

7

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
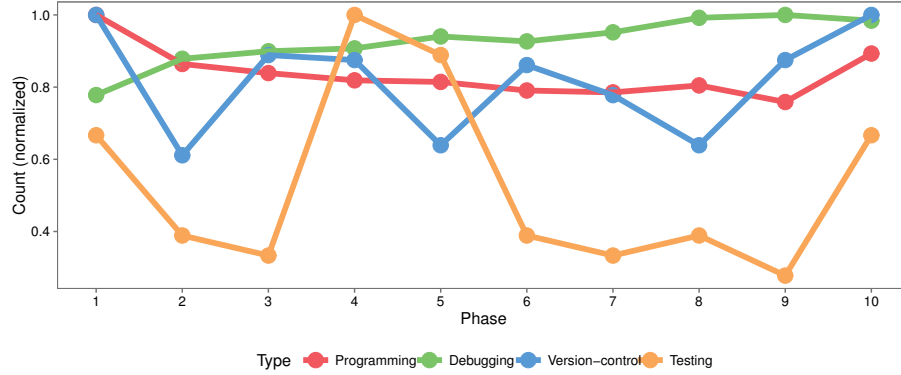
Figure 2.3: Evolution of the activities throughout the sessions.

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.