UNIVERSIDAD NACIONAL DE COLOMBIA

Source code analysis on student assignments using machine learning techniques

Hugo Armando Castellanos Morales

Universidad Nacional de Colombia
Facultad de Ingeniería
Departamento de Ingeniería de Sistemas e Industrial
Bogotá, Colombia
2017

Source code analysis on student assignments using machine learning techniques

Hugo Armando Castellanos Morales

A thesis submitted to attain the degree of:
Magíster en Ingeniería de Sistemas y Computación

Advisor:
Felipe Restrepo Calle, Ph.D.
Co-Advisor:
Fabio Augusto González Osorio, Ph.D.

Research line:
Machine learning, information retrieval, programming languages
Research group:
PLaS - Programming Languages and Systems

Universidad Nacional de Colombia
Departamento de Ingeniería de Sistemas e Industrial
Facultad de Ingeniería
Bogotá, Colombia
2017

Abstract

To increase the success in computer programming courses, it is important to understand the learning process and common difficulties faced by students. Although several studies have investigated possible relationships between students performance and self-regulated learning characteristics, little attention has been given the source code produced by students in this regard. Such source code might contain valuable information about their learning process, specially in a context where practical programming assignments are frequent and students write source code constantly during the course.

This poses the following research questions: What is the relationship between the characteristics of students source code and their performance in a computer programming course?. What is the relationship between source code features and self-regulated learning characteristics (i.e., motivation and learning strategies) in a computer programming course?. How the source code and self-regulated features can predict the students' performance?

In order to answer these questions, a strategy to support the correlation analysis among students performance, motivation, use of learning strategies, and source code metrics in computer programming courses is proposed. A comprehensive case study is presented to evaluate the strategy. Additionally, an automatic grading tool for programming assignments was used, which facilitated to obtain the source code of the participants for further automatic source code analysis. Moreover, self-regulated learning characteristics were collected using the Motivated Strategies for Learning Questionnaire (MSLQ).

Results show that the main features from source code which are significantly related to students performance and self-regulated learning features are: length-related metrics, with mainly positive correlations; and Halstead complexity measures, correlated negatively. In the light of the findings of this study, it is possible to understand better students source code as an artifact that can be used to monitorize several characteristics related to self-regulated learning, course performance, and in general, their learning process. In this way, more research in the area is required to verify if these relationships could give to computing educators new ways to identify and help students with problems.

Keywords: [Motivation, learning strategies, machine learning, source code analysis, self-regulation].

Resumen

Para mejorar el éxito de los estudiantes en los cursos de programación, es importante entender el proceso de aprendizaje y las dificultades comunes que enfrentan los estudiantes. Aunque muchos estudios han investigado las posibles relaciones entre el rendimiento de los estudiantes y aspectos de la auto-regulación del aprendizaje, poca atención se le ha dado al código fuente producido por los estudiantes. El cual puede contener información valiosa acerca de su proceso de aprendizaje. Esto es especialmente cierto en contextos donde las actividades prácticas de programación son frecuentes y los estudiantes escriben código fuente constantemente durante el desarrollo del curso.

Lo anterior, plantea las siguientes preguntas de investigación: ¿Cuál es la relación entre las características del código fuente de los estudiantes y su rendimiento en un curso de programación de computadores?. ¿Cuál es la relación entre las características del código fuente y características de aprendizaje auto-regulado (motivación y estrategias de aprendizaje) en un curso de programación de computadores?. ¿Cómo el código fuente y las características de aprendizaje auto-regulado pueden predecir el rendimiento de los estudiantes?

Para responder estas preguntas, se presenta una estrategia para realizar el análisis de correlaciones entre el rendimiento de los estudiantes, motivación, el uso de estrategías de aprendizaje, y las métricas de código fuente en cursos de programación de computadores. Un caso de estudio exhaustivo es presentado para evaluar la estrategia propuesta usando datos recolectados de estudiantes. Además se usaba una herramienta de calificación automática para evaluar las practicas, lo cual facilitaba la obtención de código fuente de estudiantes para su análisis posterior. Las características de aprendizaje auto-regulado fueron obtenidas usando el cuestionario: Motivated Strategies for Learning Questionnaire Colombia (MSLQ-Colombia).

Los resultados muestran que las principales características del código fuente que estan relacionadas con el rendimiento de los estudiantes y características auto-reguladas son: las métricas de longitud, que se correlaciona positivamente; y las medidas de complejidad de Halstead, las cuales se correlacionan negativamente. Dados los resultados, es posible entender mejor el código fuente de los estudiantes como un artefacto que puede ser usado para monitorear características relacionadas con el aprendizaje auto-regulado, rendimiento en el curso, y en general, su proceso de aprendizaje. De esta forma, investigaciones adicionales son necesarias para verificar si dichas relaciones pueden dar a los educadores nuevas herramientas para identificar y ayudar a estudiantes con problemas.

Keywords: [**Motivación, estrategias de aprendizaje, aprendizaje de máquina, análisis de código fuente, auto-regulación**].

Table of contents

1. Introduction

In the last decades, computer programming has become an important subject in several engineering areas. This importance has grown together with the improvement of computer systems and its use, and the interest of the governments to attract students to Science Technology Engineering and Mathematics (STEM) careers [OCS, 2013]. Among the required student's competences are: analyze and understand how a computer program works; understand at least one programming language; be able to design, implement, test and debug a computer program, among others [Sahami and Roach, 2014].

In the academic context, while several studies have investigated possible relationships between students performance and their self-regulated learning characteristics (i.e., motivation and learning strategies used by students) [Ramírez Echeverry et al., 2014], little attention has been given to an exclusive aspect from computer programming courses, i.e., students source code. Which might contain valuable information about their learning process. This is particularly true in contexts where practical programming assignments are frequent and students write source code constantly during the course. In these scenarios, instead of understand students source code as a unique final result, it can be seen as another mean to monitorize the learning process.

Many studies have been carried out to better understand computer programming learning processes [Pears et al., 2007, Robins et al., 2003]. These works are based on source code analysis tools, including plagiarism detection [Bakker, 2014, Elenbogen and Seliya, 2008], automatic feedback generation [Singh et al., 2013] for computer programming assignments, automatic assessments [Ihantola et al., 2010], and authorship attribution (author identification) [Caliskan-Islam et al., 2014, Frantzeskou et al., 2007] tools. Many works are intended to detect cheating in the academic process. Although this is important, it is not the only possible application of source code analysis at this context, specially because plagiarism has a negative consequence on students and it does not help students directly to their improvement.

Feedback to computer programming students is reduced due to the teacher's limited time [Cheang et al., 2003]. This is particularly true in Massive Open Online courses (MOOC) where the number of students can reach hundreds of thousands [Singh et al., 2013]. One way to address the issue has been to use tools for automatic grading of programming assignments [Ihantola et al., 2010]. However, feedback is only indicating how good or bad was

the assignment submitted (i.e., source code of a computer program), but in most cases it does not indicate students how to improve. Moreover, as there are evidence that computer science related careers students have vocational orientation supported by self efficacy and peer learning [Rosson et al., 2011], it would be valuable to know more information about students self-regulated learning characteristics (i.e., motivation and learning strategies) [Zimmerman, 1998]. Current tools do not provide any information about these aspects besides technical aspects extracted from the source code (e.g. source code metrics).

The hypothesis of this work is that source code, as a human creation, contains implicit information about its author (student), that can be extracted and make it explicit and useful. Thus, this hypothesis poses the following research questions:

- Can the source code characteristics be correlated with student's performance?
- Can the source code features be correlated with the student's use of learning strategies?
- Can the source code reveal groups of students with similar characteristics (e.g., performance, source code style, motivation, the use of learning strategies)?
- How the source code can predict the student's performance?

Based on the research questions, the general objective of this thesis is to design and build a framework for analyzing source code produced by students to support the correlation analysis among coding style, students motivation, students use of learning strategies, and student performance. This can be divided into the following specific objectives:

- To develop a strategy for feature extraction from source code that captures lexical, syntactical, and semantic characteristics.
- To develop a method for non supervised analysis of a collection of student source code using machine learning algorithms.
- To develop a machine learning predictive model to do correlation analysis of coding style, students motivation, students use of learning strategies, and student performance.
- To evaluate the proposed method using actual students source code.

This work proposes the analysis of students source code and their performance, jointly with the results from a self-report questionnaire for motivation and learning strategies (i.e., MSLQ-Colombia [Ramírez-Echeverry et al., 2016]). This data is consolidated and analyzed to find relationships between individual variables, and later, using machine learning techniques, detailed relationships among characteristics are grouped in clusters. Such clusters give an indication of the main features to have into account to build a performance predictive model. This predictive model allows the quick finding of a student presenting difficulties.

Results show that the main features from source code which are related to students performance in the course are: length metrics like lines of code, which is correlated positively; and Halstead complexity measures, which are correlated negatively. Regarding to the relationship between source code features and self-regulated learning characteristics, results make evident some significant correlations of Halstead complexity measures with the use of learning strategies such as: organization of ideas, and peer learning. In addition, there are interesting results showing correlations among learning strategies like: effort regulation and critical thinking, and source code metrics including length-based metrics and complexity metrics. In addition, the predictive model results show that it is possible to predict the students' performance (final grade) through regression with a mean square error of 0,036.

The main contributions of this work are:

- A software tool for analyzing source code produced by students that supports the correlation analysis among coding style, students motivation, students use of learning strategies, and student performance.
- A better understanding of the student's learning process in computer programming subjects through the source code technical aspects, motivation and learning strategies.
- A method to enable the teacher to give and receive feedback, allowing to focus on students with problems in the learning process by detecting possible problems in early stages of a computer programming course.

In addition to the mentioned contributions, as result of this thesis the following publications were made:

1. **Hugo Castellanos**. *Personality Recognition Applying Machine Learning Techniques on Source Code Metrics*. Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation/CEUR Workshop proceedings. Kolkata, India. Dec 7-10 2016.
2. **Hugo Castellanos**, Felipe Restrepo-Calle, Fabio A. González, Jhon Jairo Ramírez Echeverry. *Understanding the relationships between self-regulated learning and students source code in a computer programming course*. Frontiers In Education 2017. Indianapolis, USA. October 18-21 2017 (under revision).

The rest of this thesis is organized as follows. Chapter 2 presents the background which consists in a conceptual framework and related works. Later, Chapter 3 describes the proposed strategy for the extraction and analysis of source code jointly with motivation and learning strategies aspects. Then, Chapter 4 presents the dataset and some descriptive statistics. Chapter 5 discusses the correlations between source code features, motivation and learning strategies aspects. Then, machine learning methods are applied in Chapters 6 and 7. In Chapter 6 clustering algorithms are applied to confirm relationships found in Chapter

4, and to find out other hidden relationships. In Chapter 7, a predictive model is proposed, tested and discussed. Finally, the conclusions of this work are presented and possible future work is described in Chapter 8.

2. Background

This chapter presents the conceptual framework and related works. It is divided into five main subjects: text analysis, source code analysis, source code metrics, self-regulation learning, and student's source code analysis.

The Section 2.1, text analysis, describes the first works on text, including its purposes, evolution, techniques, and main achievements. Later the Section 2.2, source code analysis, indicates the close relationship with text analysis, its particular purposes and current applications. The same section briefly describes the process and techniques used along with some of the challenges in the area. Later a general description on source code metrics is presented in Section 2.3, showing its importance, definition, characteristics and scales. Next, Section 2.4 summarizes the most important concepts on self-regulation learning. Finally, Section 2.5 presents related works on student's source code analysis and its applications. It is intended to present some ways in which source code analysis is used to improve the academic process.

2.1. Text analysis

Text analysis began in the XIX century with the work described in [Stamatatos, 2009]. Such work was done by Mendenhall in 1887 and used mainly the Shakespeare work to confirm authorship. Later, [Yule, 1925] did statistical studies on natural text document, including the frequency distribution of words. These were followed by work on text analysis with bayesian statistics published in [Mosteller and Wallace, 1963], which was one of the first methods that did not depend on a human expert. Since then until now works have been focused on identification and definition of features that make a text author recognizable. Among others these features are: character and word frequency, vocabulary amount, length of words, phrases and paragraphs [Rudman, 2012].

In the end of the eighties the proposed methods began to use the computer as a tool, but still depending on human experience. A clear example of this was the CUSUM method proposed in 1989 [Hardcastle, 1993], in that moment it was accepted as evidence in the court in spite of its low reliability [Holmes and Tweedie, 1995]. In a study from 1991 says "all text do not reveal multiple authorship even when we know this is the case" [de Haan and Schils, 1993]. That was because, among other reasons, text from unknown authors were used to train the

models [Stamatatos, 2009].

In the nineties, new applications emerged in several domains [Genkin and Lewis, 2005], such as:

- Large scale author identification: used to identify an author on large and different collections of text.
- Intelligence and counter-terrorism applications: consist in checking text like e-mail or documents related with terrorist activities and try to relate it with some author.
- Authenticity verification: used when a given text is supposedly written by some author but there are doubts about its truthfulness.
- Intellectual property verification: when two parties claim copyright ownership over a specific text.

The mentioned applications have been appearing with the progressive improvement in automatic systems, like machine learning and information retrieval [Ramyaa et al., 2004].

2.2. Source code analysis

Stylography is defined as "the art or method of writing, drawing, or engraving with certain style"[1]. Stylography research in natural language texts has been explored widely as discussed in the previous section. Since the nineties, there is a growing interest to explore stylography applied to software source code as well [Krsul and Spafford, 1995]. This kind of studies are motivated by similar reasons, and they also include additional fields like: plagiarism [Bakker, 2014], tracking malware authors, fraud identification [Elenbogen and Seliya, 2008], and others.

Some applications of stylography in source code include:

- Software forensics: is the process of identifying, preserving, analyzing and presenting digital evidence in a manner that is legally acceptable [Mckemmish, 1999].
- Stylometric plagiarism detection: detection of non original source code segments. Stylometry is defined by [Kumar Singh and Manimannan, 2013] as "the description of the style by quantifying aspects of the writing that can be directly measured, like sentence length (in words)".
- Copyright investigation: identify the right author comparing between two claiming parties.

[1]Definition from https://www.collinsdictionary.com/dictionary/english/stylography

Figure 2-1.: General author identification steps using stylography and stylometry

- Authorship verification: with a set of documents and authors verify that a given source was made by a specific author.

Figure **2-1** presents the general stylography application steps. Initially the source code files (documents) must be preprocessed in a way that the new format can be easily used, this depends heavily on the techniques that will be used. Later a feature extraction step is carried out, in this step those features with little impact can be removed. After that, a model is trained, and once it is done, it should be ready to perform the classification step.

In source code stylography the data set is a collection of software source code files, usually from different authors. Table **2-1**, which was adapted from [Stamatatos, 2009], shows the main features used in text stylometry. The given features are common to those used in source code analysis.

2.2.1. Preprocessing and feature extraction

To be able to analyze the data, it needs to be transformed. Such transformation is called preprocessing and consists in data objects selection (or modification) to facilitate the next steps. The exact way of preprocessing depends on several factors including: objective, techniques to be used, type of data, etc.

Authors of different research do different preprocessing levels. Fon instance, in the work of [Frantzeskou et al., 2007] the data sets are prepared to extract features training and merging all source files from the same author in just a single file.

Authors like [Shevertalov et al., 2009] initially get the structural features, such as spaces and indentation; [Caliskan-Islam et al., 2014] use all the mentioned features in Table **2-1** because they are contained in the abstract syntax tree (AST) of each file. Moreover, n-grams are used in works like [Burrows, 2010, Frantzeskou et al., 2006, Frantzeskou et al., 2007, Frantzeskou et al., 2008].

2.2.2. Training and classification

According to [Murphy, 2012] Machine learning is "a set of methods that can automatically detect patterns in data, and then use uncovered patterns to predict future data, or to perform

Table 2-1.: Stylometric features in text taken from [Stamatatos, 2009]

Features	
Lexic	Token-based (word length, sentence length, etc.) Vocabulary richness Word frecuency Word n-grams Errors
Character	Character types (letters, digits, etc.) Character n-grams (fixed length) Character n-grams (variable length) Compression methods
Syntactic	Part-of-speech (POS) Chunks Sentence and phrase structure Rewrite rules frequencies Errors
Semantic	Synonyms Semantic dependencies
Application-specific	Functional Structural Content-specific Language-specific

other kinds of decision making". Under supervised models, such methods require a training data to create a model and be able to perform classification. Usually the classification is done assigning a class to an element, but there is also possible to assign several classes to a single element.

Depending of the methods used in the feature extraction phase, different training and classification models are used. For instance, [Mckemmish, 1999] uses n-grams to build an author profile. That is, the most frequent sequence of characters (n-gram) used by an author in a file (the size of n-grams could be variable). Other authors use several different machine learning techniques like support vector machines (SVM) [Wisse, 2014], K-nearest neighbors (KNN) [Shevertalov et al., 2009], and C4.5 [Rosenblum et al., 2011].

The SCAP (Source Code Author Profile) method [Frantzeskou et al., 2007] does not use machine learning techniques, instead, a similitude measure is used to know how much similarity exists between the author profile and the example code.

2.2.3. Challenges

Source code stylography has the following challenges according to [Krsul and Spafford, 1995], [Caliskan-Islam et al., 2014] and [Shevertalov et al., 2009]:

- "The programming characteristics of programmers change and evolve" which makes difficult the studies over time.
- There are standards and conventions imposed by organizations which alters the personal stylography.
- The styles change from one language to another.
- The code reuse through code snippets alter the possible mark left in the document.

These challenges usually are avoided in the studies selecting controlled data sets.

In addition, [Brennan et al., 2012] suggest another stylography problem which is inherent to itself, and it is "adversarial stylometry". It means that an author does not want to be identified and apply techniques to change a text without alter its meaning. So far this problem has been explored only in natural language.

2.3. Source code metrics

According to [Malhotra, 2015], software metrics are used to *assess the quality of the product or process used to build it*. As stated, it is usually understood as a industry process focused on improve a product, or measure its quality. But it also provides information about the person who wrote a particular piece of code, like the expertise, knowledge, among others. And taking metrics over time can show how a person or group is improving over time.

Such metrics have the following general characteristics:

- Quantitative: metrics have a value.
- Understandable: the way the metric is calculated must be easy to understand.
- Validatable: the metric must capture the attributes which by design should.
- Economical: it must be economical to capture the metric.
- Repeatable: if measured several times the resultant values should be the same.
- Language independent: metrics should not depend on a specific language.

- Applicability: metrics should be applicable in any phase of the software development.
- Comparable: the metric should correlate with another metric capturing the same concept.

Moreover, source code metrics must have a scale which can be:

- Interval: it is given by a defined range of values.
- Ratio: it is a value which have an absolute minimum or zero point.
- Absolute: it is a simple count of the elements of interest.
- Nominal: it is a value which mainly defines a discrete scale of values, like 1–present or 0–not present.
- Ordinal: it is a categorization which is intended to order or rank, for instance levels of severity: critical, high, medium, etc.

Metrics can be classified according to the intended measure:

- Size: usually intended to estimate cost and effort. The most popular metric in this category is the source lines of code (SLOC). In object oriented languages the size can be measured by the number of classes, methods, and attributes.
- Software quality: intended to measure the quality of the software, this metric can be divided in two main categories:
  - Based on defects: consist in measure the level of defects. The main metrics in this category are: defect density defined as number of defects by SLOC; defect removal effectiveness which is defined as the defects removed in a phase divided by latent defects. If the latent defects are unknown then they can be estimated based on previous phases.
  - Usability: this kind of metrics is intended to measure the user satisfaction using the software. The satisfaction can be given be the ease to use and learn.
  - Complexity metrics: they are oriented to produce a measure on the difficulty to test or maintain a piece of source code. These metrics try to measure how complex a source code is. The complexity can be defined in several ways, but is usually intended to measure how many computational resources will be used in an algorithm execution. For example, the McCabe complexity metric [McCabe, 1976].
  - Testing: intended to measure the progress of testing over a software.
- Object oriented metrics: intended to measure object oriented paradigm, which can be divided in:

    - Coupling: measures the level of coupling (level of interdependence between classes), it is calculated counting the number of classes called by another class.
    - Cohesion: measures how many elements of a class are functionally related to each other.
    - Inheritance: measures the depth of the class hierarchy.
    - Reuse: measures the amount of times that a class is reused.
    - Size: intended to measure the size but not only in lines of code but also in the particularities of object oriented paradigm, like method count, attribute count, class count, etc.

- Evolutionary metrics: try to measure the evolution of a software based on different elements like revisions, refactorings, bug-fixes. How much lines of code are new, modified or deleted.

2.4. Self-regulation learning

An important part of a student learning process is the self-regulation, which is defined as the process done to control cognition, behavior and motivation with the purpose of reaching a goal [Boekaerts et al., 2005]. Among several models, the general structure of self-regulated learning from [Pintrich, 1999] defines the possibility to self-regulate:

- Cognition: to apply strategies to learn about a subject, including the construction of new concepts from previous knowledge.
- Motivation: including self-reward, and auto-persuasion with the purpose to improve interest in the subject.
- Behavior: to manage the learning resources, especially time, and in general, the actions which could help to improve the learning process, like study with peers.
- Environment: to generate strategies to adapt or control the environment, understanding this as the conditions in class, teacher behavior, etc.

To be able to self-regulate, a student must create strategies, which ease the learning process [Weinstein and Mayer, 1983] and help to understand the reasons of failure or success. As reported in [McKeachie, 1986] there are three main learning strategies: Cognitive, Metacognitive, and Resource management.

The cognitive strategies are:

- Reharsal: consists in reciting names from a list to be learned mainly with memorization purposes. This strategy influence attention but does not improve the integration of the new knowledge with the old one.
- Elaboration: intended to help in long term memory, creating internal connections between subjects.
- Organizational: intended to select information and create connections among the available information.

The metacognitive strategies are:

- Planning activities: consist in setting goals and creating tasks to be able to reach such goals.
- Monitoring activities: are self monitoring activities like self testing. This allows the verification of the progress made in a specific time.
- Self regulation activities: consist in the adjustment of a task according with the results in the monitoring activities.

The resource management strategies are:

- Time management: is the scheduling done by the student to do his/her tasks. Because of its nature is highly related to planning and regulation activities.
- Study environment: the student defined area of study. This includes preparing an area without distractions or with the resources to accomplish the goals.
- Self-effort management: is the capability of a student to identify when he/she needs help and how such help is obtained; for instance, studying with peers.

To be able to assess self-regulation learning, the *Motivated Strategies for Learning Questionnaire* (MSLQ) [Pintrich et al., 1991] is used in several studies, including this thesis. It is a tool that measures "the motivational orientations and their use of learning strategies". It is divided in two sections: motivation and learning strategies, which are described in Tables **2-2** and **2-3**, respectively. The row numbers in these tables will be used latter in this thesis to reference each feature.

2.4.1. Self-regulated learning on computer programming courses

Self-regulation in computer programming learning in academic context has been identified as an important matter in university context. Several studies have explored self-regulation learning on computer programming courses. For instance, [Nelson et al., 2015] studied motivational and self regulation profiles adopted by students in computer science basic levels

Table 2-2.: MSLQ Motivational features

	Feature	**Description**
0	Task value	It is the value a student gives to a certain task, in terms of how interesting or how useful is the task.
1	Anxiety	Includes students negative thoughts that affect performance.
2	Extrinsic goals	The perception to participate in a subject due to some kind of reward (grade competition).
3	Control of learning beliefs	The belief that the positive outcomes are the result of their own effort.
4	Intrinsic goals	The perception of the reasons why he/she likes the subject.
5	Self-efficacy learning	A self-evaluation of the ability to master a task.
6	Self-efficacy performance	It relates to the expectations about the performance in the course.

Table 2-3.: MSLQ Learning strategies features

	Feature	**Description**
7	Time to study	It is related to management, planning and effective use of the study time.
8	Peer learning	It refers to students learning with and from each other as fellow learners without any implied authority to any individual.
9	Meta-cognition method	Continuous adjustment of the learning activities.
10	Elaboration of ideas	Remember information in long-term memory by building connections among the items to learn.
11	Effort regulation	Ability to control the effort avoiding distractions.
12	Meta-cognition monitoring	Includes self testing to be able to better understand the subject.
13	Rehearsal	Repeating items in a list in order to memorize it.
14	Critical thinking	How the student apply old knowledge to deal with new situations.
15	Study environment	Management and organization of the place where the person studies.
16	Meta-cognition planning	Plan activities to ease the learning process.
17	Organization of ideas	To select and organize the information properly.

oriented to engineering. Their results show that students adopted profiles with limited perception to learn the course contents, inadequate orientation of learning goals and not effective behaviors for self-regulation. Moreover, [Ambrosio et al., 2012] presented a study of the profile of 190 students in introductory programming courses in Brazil and Portugal with respect to their attitudes and self-regulation behavior. Results suggest that these groups of students have a very similar profile regarding to: learning strategies, self-efficacy perception, and study organization activities. Some other studies suggest that vocational orientation to computer science careers is determined in most cases by self-efficacy and social support (like pair learning) [Rosson et al., 2011]. Overall, although motivation and learning strategies have similar traits among engineering students, these features have a strong personal and environmental component, and therefore, it is necessary to do specific studies to characterize them.

Moreover, relationships between academic performance and self-regulation learning have been reported in several studies, e.g., [Yukselturk and Bulut, 2007, DiFrancesca et al., 2016, Cheng and Keung, 2011]. In general, good performance of programming students is characteristic of students with better self-regulated learning skills [Alhazbi, 2014].

Taking into account the importance of self-regulation learning abilities in the students performance, some tools have been developed to ease the self-regulation. For example, [Manso-Vázquez and Llamas-Nistal, 2015, Ortiz et al., 2015] propose to foment these abilities through monitoring the progress of students in computer programming learning. These tools are oriented to teachers and students, which can use them to ease the planning, monitoring, and assessment of the learning process from both perspectives.

2.5. Related works on student's source code analysis

This section presents the related works on source code analysis, which focus mainly on analyzing source code developed by students in an academic context. The approximations have different purposes, including: cheat detection, feedback and assessment support, and programming expertise.

2.5.1. Cheat detection

As one of the main purposes in the academic context is the effective student learning, it is important to have techniques which avoid cheating and any form of student plagiarism. To carry out source code analysis in order to address this problem, several approximations have been proposed. They can be grouped in two approaches: authorship attribution, and source code style.

Firstly, to detect cheating by authorship attribution, i.e., verifying that a source code presented by a student is effectively his/her creation. The same techniques used in forensics applications can be used to carry out this approach. For instance, N-grams or other information retrieval techniques like the ones proposed in the works of [Burrows, 2010, Frantzeskou et al., 2006, Layton et al., 2013]. The approaches based on N-grams capture a lot of information but they do not describe the source code structure. These techniques could be highly affected when a single file has more than one author.

Secondly, there are a group of works that use source code style. These also commonly use machine learning or information retrieval techniques together with a representation form. In the work of [Caliskan-Islam et al., 2014] it is used an Abstract Syntax Tree (AST) to represent the source code, including the style features, and to identify authors they apply a random forest classification. Moreover, [Elenbogen and Seliya, 2008, Joshi and Argiddi, 2013] extracted style features from the source code, including a lot of unused information, and with these features a C4.5 algorithm was used to classify the author (students). Other works also use style information but additionally perform static analysis over the source codes to use its output as a feature, and based on those features identify the authors [Hayes and Offutt, 2010]. The main advantage of these methods consists on the ease to understand and identify the discriminant features.

2.5.2. Automatic feedback and assessment

The automatic assessment is a way to evaluate and grade a student source code automatically [Ihantola et al., 2010]. It usually consists on a specific assignment, and a known set of inputs/outputs. In addition to help the teacher, it can be intended for assignment feedback. This feedback could be as simple as a grade or more elaborated like a suggestion for improvement.

There is a wide set of tools to accomplish the automatic feedback and evaluation. For example, programming contest judges like DomJudge[2]. However, this kind of tools do not provide all the features of a Learning Management System (LMS). An integration to a LMS like Moodle[3], Blackboard[4] or Sakai[5], is highly desired, because it constitutes an integral part of the learning process in many courses. This has been studied in [Radenski, 2008, Rößling et al., 2008].

To be able to give some feedback the system must have a test framework. Depending on the programming language it can be:

[2]http://www.domjudge.org
[3]https://moodle.org/
[4]http://www.blackboard.com
[5]http://www.sakaiproject.org

- Unit tests [Amelung et al., 2008]: tests defined to check a specific result in portions of code.
- Acceptance testing [Sauvé and Abath Neto, 2008]: tests defined in natural language-like scripts.
- Output comparison: refers to the traditional way that most programming contest judges use for the assessment. The system checks for an specific expected output given a known input, and indicates if the program it is correct or not.

In addition, some authors like [Malmi et al., 2005] suggest the definition and use of a resubmission policy, depending on the purpose of the exercise. Among these policies are:

- Limit number of submissions: to avoid trial and error approaches by students.
- Limit amount of feedback: intended to force students to think after a submission with errors.
- Penalties: related to the maximum number of submissions, it is intended to force the student to submit only when necessary.
- Different exercise on every submission [Brusilovsky and Sosnovsky, 2005]: intended to avoid trial and error, because the exercise changes on every submission.

The majority of the aforementioned approximations limit their automatic feedback to a correct/incorrect decision, which is a strong limitation because students do not have a way to know how to improve [Pieterse, 2013]. Contrarily, when a student receives a useful feedback related to the reasons why the submission failed, he/she will learn from it and will avoid to repeat the same mistakes again.

An approximation to generate a valuable feedback from the learning process perspective is proposed by [Singh et al., 2013]. A meta-language called Error Model Language (EML) which contains a series of rules that can be used to generate feedback automatically. It is based on the common errors that a student's source code could have. Nonetheless, this is also a limitation because an error not specified in the EML specification is an undetected error, and therefore, there is not an appropriate feedback for this.

Another approach has been the source code style analysis. With the purpose of improving coding standards and avoiding programming errors [Ala-Mutka et al., 2004], even when this is not intended to provide a grade, this method provides continuous feedback about the written code.

2.5.3. Programming expertise

Measure the programming expertise is an application that is usually used in more advanced programming related courses and in the professional practice. It is aimed at assessing how good is a programmer and which are the best practices used by him/her. An example of this approach can be found in [Kuric and Bieliková, 2014], in which are extracted some metrics related to the programming expertise, or in the case of students, the grade.

Another proposal was presented by [De Lucia et al., 2011], in which is described an experiment to identify the quality (how informative) of the comments and identifiers, and how such quality improves the style and traceability of the code.

3. Strategy to analyze student assignments source code

The success rate improvement of computer programming courses is a source of concern for teachers. Understanding how students' motivation and learning strategies are correlated to the information available in their academic works, could give the teachers a path of improvement.

The purpose of the framework presented in this thesis is to provide new ways to understand how students' motivation and learning strategies are correlated to the source code metrics. Also, to be able to find groups of metrics and motivation and learning strategies which lead to a predictive model.

The proposed framework is summarized in the Figure **3-1**. It is divided in four main steps: data collection and consolidation, source code analysis, correlation analysis, and knowledge extraction by means of the use of machine learning techniques.

3.1. Data sources and preprocessing

For the purposes of this thesis, there are three different data sources:

- Students' source code: corresponds to each one of the attempts (correct or not) to solve an assignment in a computer programming course. Each attempt is processed no matter how much of the file is changed with respect to previous attempts, or if a previous attempt was already correct. Such assignments are worked and submitted throughout the course duration.

- Student performance: it is measured through the final grades obtained by the students. These are the official grades provided by the teachers and are based mainly on the results of the programming assignments.

- Motivation and learning strategies: corresponds to data which indicates the learning strategies used by the students and their motivation in a moment during the develop-

ment of the course. This is obtained through the application of an MSLQ test to the participant students.

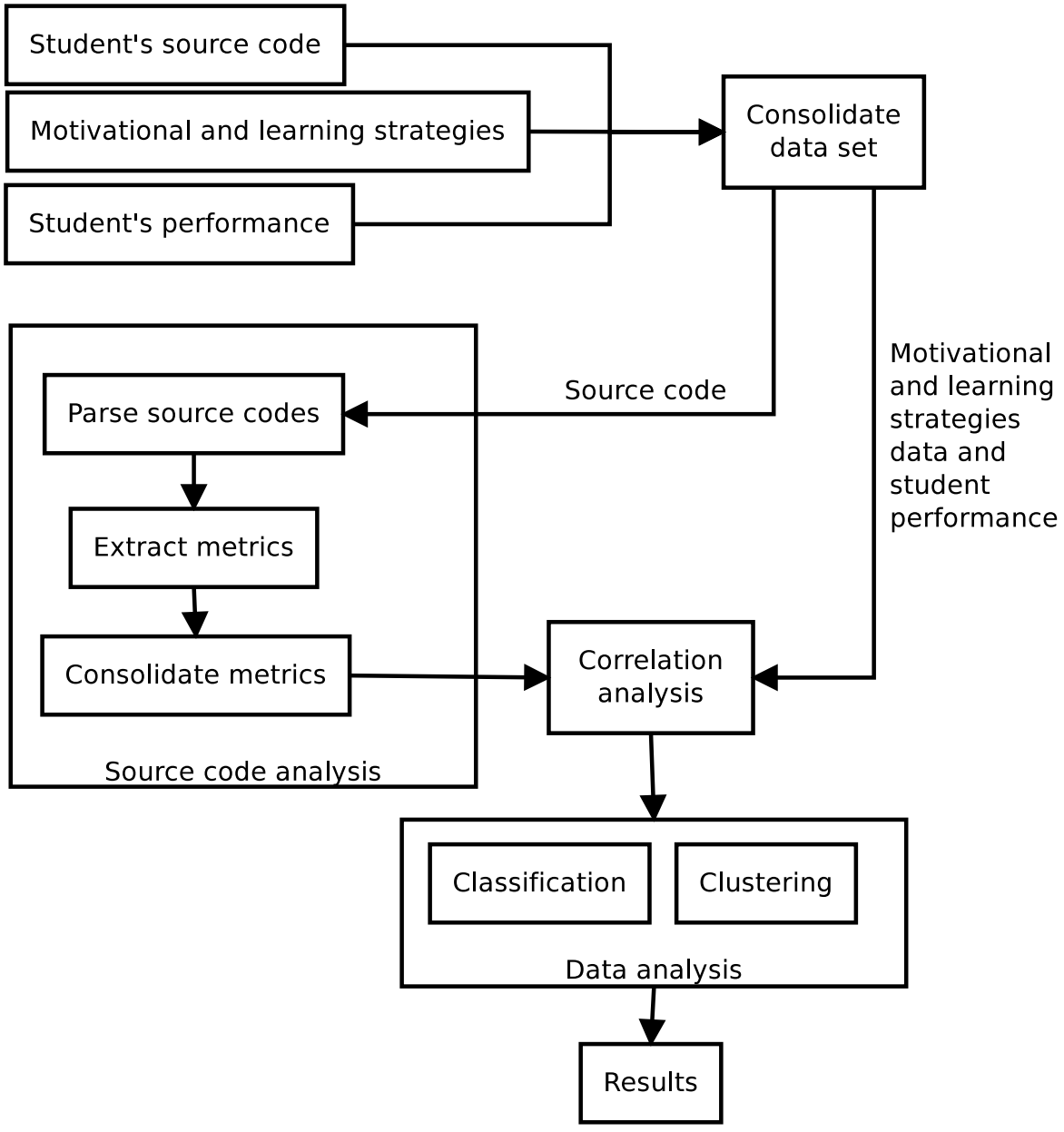


Figure 3-1.: Proposed strategy for the analysis of students source code

Once the three data sources are collected, they are consolidated in a single data set. This consolidation consists in the verification of the students who submitted programming assignments and answered the MSLQ test. This allows the students' labeling and the confirmation of various important information like:

- The student name: to be sure the students are correctly identified as a unique person. This is specially important for those who in previous semesters abandoned or did not approve the course.
- Identify the real enrolled students: to be able to process only those students who were really enrolled to the course and discard those who were only assistant to the class, who could have submitted some of the assignments.
- Identify students who abandoned: this students could have been enrolled in first place but later decided to cancel or abandon the class. This students could submit some assignments and get some grades because of this, but never finished the course.
- The complete MSLQ survey response: to be able to work with trusted data, it is verified that the students answered every single question from the MSLQ test. This is important because incomplete answers can generate unexpected results at the moment of analysis.

3.2. Source code analysis

Later, the source code analysis stage is performed. First, the consolidated data set of the students' source code is processed using a custom tool, designed to parse the source code files and to extract the source code metrics for each one of the file for every student. Such metrics are described in Table **3-1**.

Table 3-1.: Information extracted from the source code

#	**Metric**	**Description**
0	Amount of files	The total amount of files sent to DomJudge
1	Average source lines of code	Is the sum of all source code lines from all files divided by the number of files sent by a student.
2	Average class number by file	The total amount of classes in all source files divided by the number of files sent by a student.
3	Average source code lines by class	The total amount of lines of all classes divided by the total class amount.
4	Average attributes by class	The total number of attributes (static or not) divided by the total class amount.
5	Average methods by class	The total number of methods (static or not) divided by the total class amount.

Continued on next page

Table 3-1 – *Continued from previous page*

#	**Metric**	**Description**
6	Average class name length	The sum of the class name lengths divided by the number of classes.
7	Average amount of for loops	The total amount of for loops divided by the number of methods.
8	Average amount of while loops	The total amount of while loops divided by the number of methods.
9	Average amount of if clauses	The total amount of if clauses divided by the number of methods.
10	Average amount of if-else clauses	The total amount of if-else clauses divided by the number of methods.
11	Cyclomatic complexity	Indicates the cyclomatic complexity average
12	Average of static attributes	The average of static attributes contained in a class
13	Average parameters	Average number of total number of parameters divided by the total number of methods in a class
14	Average of static methods	The average of static methods per class
15	Average correct	The average number of files which were correct according to the judge
16	Average wrong	The average number of files which were wrong according to the judge
17	Average time limit	The average number of files which hit time limit according to the judge
18	Average compilation error	The average number of files which had compilation error according to the judge
19	Average execution error	The average number of files which had execution errors according to the judge
20	Average no output	The average number of files which had no output according to the judge
21	Average identifier length	The average identifier length by files, this includes names of variables, classes, parameters, etc.
22	Amount of correct files	Total amount of correct files
23	Halstead: bugs delivered	Indicates the number of possible bugs generated

Continued on next page

Table 3-1 – *Continued from previous page*

#	**Metric**	**Description**
24	Halstead: Difficulty	An index which measure the difficulty
25	Halstead: Effort	An index which measure the necessary effort to write the code
26	Halstead: Time to understand	An index which indicates the time taken to write a software
27	Halstead: volume	Indicates how much information the reader needs to get to understand the code
	Average of overriden methods	The average of methods overrided
	Deep average	The level in the hireachical tree
	Average number of interfaces	The average number of interfaces which is implemented
	Efferent coupling	Is the amount of classes which a class depends on, or the number of external types the class knows
	Specialization index	Is a relation given by the number of overriden methods, the depth in the hireachical tree and the number of methods.
	Instability	Is the relation given by the efferent coupling and the afferent coupling.
	Lack of cohesion	Is an index which indicates the cohesion level of a class, a high value indicates that the class should be splitted

As mentioned before, when the source code analysis tool generates the values for each metric, they are extracted and consolidated per student. These metrics are normalized to ease the later stages of the framework. The metrics are classified in three groups:

- *Length metrics*: contain the metrics related to some length/size measure and they are calculated as the average among: amount of files, average source lines of code, classes per file, source code lines per class, attributes per class, methods per class, class name length, and the average number of parameters.

- *Complexity metrics*: contain the metrics related with algorithm complexity and it is calculated as the average of: amount of for loops, amount of while loops, amount of if clauses, amount of if-else clauses, and the average identifier length.

- *Halstead*: contains all the Halstead metrics extracted, it was calculated as the average of: Halstead bugs delivered, Halstead difficulty, Halstead effort, Halstead time to understand or implement, Halstead volume.

3.3. Data analysis

Once the source code metrics have been consolidated, the correlations analysis can be performed. Its purpose is to identify correlations between source code metrics, motivational features and the student performance. To properly calculate the correlation, a normal distribution test is done over the values, in case of normality the Pearson correlation is used, otherwise Spearman correlation is used. This stage shows traces of possible relationships among the elements in the data sources.

The consolidated information jointly with the correlation analysis facilitates to:

- Format the data: to be used as the input in the later stages.
- Identify and analyze correlations: identify and analyze correlations among students performance, motivation, learning strategies, and their metrics extracted from the source codes. This allows to obtain an initial grouping and identification of groups of students with certain labels.
- Clustering: allows to confirm or discard the groups found in the correlations and the groups the features. Such groups also give information about the learning strategies of the best performing students.
- Predictive model: allows the use of algorithms with previously labeled data to be able to build a predictive model.

The final step in the proposed strategy is the knowledge extraction by means of the use of machine learning techniques. Firstly, clustering analysis is performed, expecting to obtain results consistent with the correlations obtained, and additional information which is not evident in the correlation analysis. The clustering algorithm was hierarchical using the Ward's method. This allows an easier interpretation and analysis. Additionally, during this stage a bi-cluster analysis is also carried out, allowing to find groups generated from two different sources (source code metrics and MSLQ data), which allows to identify groups of metrics related to motivation and learning strategies. Secondly, a predictive model is built and validated. The predictive model is trained using the final grade as a label in a Support Vector Regressor (SVR). In addition, as the grade indicates if a student approved or not, this is used as a category to train a Support Vector Machine (SVM). This allows to predict with an important accuracy if a student will approve based on the written source code and the results of the MSLQ.

This tool could be used as an indicator to identify students with problems at an early stage of the courses, allowing the teachers to take preventive or corrective measures with a student. In this way, it is possible to understand better students source code as an artifact that

can be used to monitorize several characteristics related to self-regulated learning, course performance, and in general, their learning process.

4. Data exploration

This chapter presents the description of the data used for this thesis. Such description includes: amount of students, grades, academic periods, descriptive statistics like mean and standard deviation, comparison of features among semesters, and source code metrics description.

For the experimentation done throughout the development of the present document, a total of 205 students opted in to participate voluntary. They were enrolled in Data Structures course in Universidad Nacional de Colombia, which is a second year course for computer science students. Participants were distributed in three different semesters during 2015 and 2016, i.e., 2015-I, 2015-II, and 2016-I. Each semester has a duration of 16 weeks.

Table **4-1** summarizes the students' performance in the programming course. For the three semesters, this table shows the number of students who participated in the study (n), and some descriptive statistics for the final grade, i.e., mean ($\bar{x}$) and standard deviation (σ). Students are classified in three categories: approved, not approved, and dropout. Students whose final grade was $\geq$ 3,0 were classified as approved; those whose final grade was $<$ 3,0 were classified as not approved; and students who did not have a final grade at the end of the course were categorized as drop out.

In average the approval rate of students among the three semesters was 59.8 %. The failure rate (not approved) was 21.4 %, and the drop out rate was 18.8 %. As it can be seen, the drop out rate was higher in semester 2015-I than in the other two periods. In addition, the not approved rate was higher in semester 2016-I. Although this semester was the period with more students, the standard deviation in the final grade is lower than in the previous semesters for those who approved and slightly higher than 2015-II for those who did not.

Moreover, the motivational and learning strategies data was collected using MSLQ-Colombia [Ramírez-Echeverry et al., 2016], which is an adaptation of the original questionnaire to Colombia. Table **4-2** presents the descriptive statistics ($\bar{x}$ and σ) found for each one of the self-regulated learning features in the three academic periods. The questionnaire was applied in the fifth week of each one of the semesters.

In semesters 2015-I and 2015-II the standard deviation was less than 2.0 with small dif-

Table 4-1.: Students performance in the programming course

	2015-I				**2015-II**				**2016-I**			
	n	%	**Grade**		**n**	%	**Grade**		**n**	%	**Grade**	
			$\bar{x}$	σ			$\bar{x}$	σ			$\bar{x}$	σ
Approved	28	52.8	3.58	0.53	35	68.6	3.58	0.60	59	58.1	3.69	0.43
Not approved	9	16.9	1.13	0.93	7	13.7	2.00	0.74	34	33.6	1.97	0.8
Drop out	16	30.1			9	17.6			8	7.9		
Total	53				51				101			

ferences between semesters. However, in 2016-I it became higher, which can be explained by the number of students during this semester (101 students), almost twice the amount of students from other semesters. Furthermore, among the three semesters, some features have a significant difference between them. For instance, the average in critical thinking in 2015-I is 4.54, while in 2016-I it is only 3.71; organization of ideas and rehearsal, had their biggest averages (4.09 and 4.7 respectively) in 2015-II, and in 2016-I these values decreased to 2.41 and 2.92, respectively. Moreover, the semester 2015-II presents the lowest averages in motivational features. For example, the intrinsic goals is, in average, 3.06, which is low for a self-motivation feature. Nevertheless, this semester shows the best approval rate.

Finally, the dataset is completed with all the source codes submitted by each one of the students during the courses. Each row of the dataset will correspond to a single student. The dataset columns contain the MSLQ features, the source code metrics, and student performance. All data is normalized and consolidated in such a way that a single row corresponds to a single student.

4.1. Source code metrics

The computer programming course was supported by an automatic grading tool for programming assignments, namely *DomJudge*[1] (online judge), which facilitated to obtain the source code of the participants for further automatic source code analysis. The following data was extracted from the DomJudge databases:

- The team id: In this case the field corresponds to the student user name.
- The problem id: To uniquely identify an specific assignment.
- Source code: The assignment source code.

[1] `http://www.domjudge.org`

Table 4-2.: MSLQ Data set description

		2015-I		2015-II		2016-I	
	Feature	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
Motivation	Task value	5.78	1.39	4.39	1.82	4.76	2.65
	Anxiety	4.64	1.53	3.44	1.91	3.58	2.25
	Extrinsic goals	4.77	1.42	3.39	1.81	3.62	2.38
	Control of learning beliefs	5.79	1.25	4.19	1.79	4.83	2.69
	Intrinsic goals	4.50	1.55	3.06	1.61	3.89	2.38
	Self efficacy learning	5.44	1.31	4.09	1.80	4.40	2.57
	Self-efficacy performance	5.45	1.38	3.97	1.89	4.41	2.48
Learning strategies	Time to study	3.80	1.56	4.86	1.84	3.16	2.12
	Peer learning	3.93	1.86	4.21	1.68	3.45	2.23
	Meta-cognition method	4.15	1.65	4.18	1.72	3.33	2.10
	Elaboration of ideas	4.47	1.41	4.18	1.73	3.77	2.27
	Effort regulation	5.01	1.34	4.54	1.75	3.96	2.28
	Meta-cognition monitoring	5.25	1.26	4.58	1.80	4.26	2.40
	Rehearsal	3.86	1.54	4.70	1.80	2.92	1.95
	Critical thinking	4.54	1.32	4.27	1.67	3.71	2.18
	Study environment	5.31	1.58	4.81	2.00	4.12	2.53
	Meta-cognition planning	4.50	1.57	4.13	1.60	3.46	2.12
	Organization of ideas	3.12	1.67	4.09	1.68	2.41	1.73

- Result: The outcome of the assignment. For each source file, the result can be one of the following:
  - compiler-error: The source file could not be compiled.
  - correct: The source code compiled, ran in the specified execution time, and obtained the expected output.
  - no-output: The source code compiled and ran but did not obtain any output.
  - run-error: The source code compiled an ran but there was an execution error (runtime exception).
  - timelimit: The source code compiled an ran but took more time to complete the

task than the specified in the assignment.

- wrong-answer: The source code compiled an ran but the result was not the expected one.

Once the extraction was performed, the source code files were processed with a custom source code automatic analysis tool, which uses ANTLR [Parr, 2013]. Such tool, was written from scratch using a Java 8 grammar to do an analysis and metric extraction from the source code files. This grammar was chosen for two reasons: all the source codes submitted by students were written in the Java programming language, and also due to this grammar in its version 8 is a superset of the previous versions (it supports also the versions 6 and 7 of the language).

The tool works parsing the source code file, and a specific process is done and for each sentence which contributes to a metric. The mentioned process usually involve counting, measure length and the context of several statements. For instance, its different a class identifier (class name) to a variable identifier (variable name). Once the most basic data is obtained, some formulas are applied to get the source code metrics. During the development of this chapter the formulas used to get the results are explained.

Once all the values are calculated, the information per student is consolidated, calculating the average of each metric among all the source code files. And later the data is also normalized. Such normalization is needed to ease the analysis of the data, for instance to compare the results among semesters.

Additionally, the tool stores the resulting information, in a database. Such database contains the raw data and normalized data. This allow to extract with different options the consolidated data without repeating all the process over again.

The source code processing explained above is done for each file sent by a student. The set of metrics calculated is presented in the Table **3-1**. Table **4-3** presents the standard deviation and average per source code metric, semester and final result (approved, not approved). Dropout students were not considered because the data collected for them was not complete. All values were normalized to ease the interpretation.

Although more details will be discussed in next subsections, there are some considerations that are valuable to highlight at this point. In general, the values corresponding to length related metrics (like file amount, average lines per, etc.) are greater for those who approved and this is consistent in all semesters. This behavior, however, is inverted regarding to Halstead metrics, with exception of time to understand or implement. Other kind of metrics obtain mixed results.

Table 4-3.: Average and standard deviation of source code metrics during the three semesters

	2015-I				**2015-II**				**2016-I**			
	Approved		**Not approved**		**Approved**		**Not approved**		**Approved**		**Not approved**	
	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ	$\bar{x}$	σ
File amount	0.54	0.14	0.14	0.11	0.56	0.13	0.5	0.24	0.36	0.12	0.24	0.23
Average lines per file	0.40	0.14	0.23	0.11	0.49	0.13	0.46	0.19	0.39	0.23	0.28	0.21
Average class per file	0.51	0.12	0.43	0.16	0.47	0.08	0.41	0.12	0.44	0.19	0.35	0.18
Average lines per class	0.70	0.2	0.42	0.12	0.65	0.15	0.61	0.15	0.45	0.19	0.38	0.19
Average attributes per class	0.62	0.12	0.46	0.04	0.41	0.12	0.38	0.09	0.55	0.20	0.32	0.24
Average methods per class	0.71	0.12	0.50	0.11	0.62	0.07	0.65	0.12	0.50	0.19	0.38	0.19
Average class name length	0.55	0.09	0.51	0.06	0.64	0.07	0.65	0.06	0.47	0.15	0.44	0.15
Average for loops per method	0.17	0.06	0.20	0.05	0.11	0.03	0.11	0.03	0.11	0.17	0.19	0.10
Average while loops per method	0.18	0.09	0.22	0.14	0.10	0.05	0.12	0.02	0.16	0.15	0.17	0.20
Average if per method	0.26	0.06	0.30	0.09	0.31	0.04	0.32	0.07	0.15	0.10	0.21	0.20
Average if-else per method	0.09	0.01	0.11	0.05	0.17	0.04	0.18	0.06	0.23	0.14	0.24	0.20
Cyclomatic complexity	0.36	0.12	0.39	0.18	0.38	0.11	0.32	0.08	0.20	0.11	0.25	0.20
Average static attributes	0.27	0.11	0.24	0.16	0.51	0.13	0.49	0.13	0.24	0.11	0.25	0.21
Average parameters per method	0.21	0.18	0.06	0.12	0.18	0.14	0.13	0.21	0.25	0.17	0.16	0.23
Average static methods	0.15	0.12	0.25	0.32	0.12	0.07	0.16	0.09	0.28	0.20	0.26	0.19
Average correct files	0.18	0.12	0.11	0.13	0.25	0.14	0.33	0.12	0.21	0.12	0.28	0.20
Average files with errors	0.06	0.20	0.00	0.00	0.06	0.13	0.14	0.34	0.04	0.11	0.05	0.19
Average file time limit	0.72	0.06	0.69	0.04	0.78	0.05	0.76	0.02	0.51	0.10	0.45	0.11
Average files with compilation error	0.57	0.18	0.20	0.14	0.67	0.14	0.56	0.31	0.32	0.14	0.18	0.18
Average files with execution error	0.53	0.09	0.53	0.10	0.67	0.07	0.67	0.07	0.49	0.09	0.51	0.14
Average files with no output	0.25	0.05	0.29	0.06	0.21	0.03	0.22	0.04	0.21	0.13	0.29	0.22
Average identifier length	0.13	0.10	0.09	0.14	0.21	0.17	0.11	0.04	0.19	0.19	0.14	0.23
Average amount of correct files	0.13	0.06	0.18	0.05	0.10	0.03	0.11	0.05	0.28	0.25	0.48	0.36
Halstead bugs delivered	0.02	0.00	0.24	0.32	0.01	0.00	0.02	0.01	0.01	0.01	0.10	0.23
Halstead difficulty	0.02	0.03	0.17	0.34	0.04	0.04	0.03	0.05	0.08	0.07	0.22	0.26
Halstead effort	0.05	0.10	0.16	0.34	0.02	0.03	0.03	0.06	0.07	0.12	0.14	0.24
Halstead time to understand implement	0.16	0.10	0.05	0.34	0.02	0.03	0.03	0.06	0.07	0.12	0.14	0.24
Halstead volume	0.01	0.02	0.17	0.34	0.03	0.03	0.03	0.04	0.04	0.03	0.15	0.20
Final grade	0.74	0.11	0.28	0.18	0.74	0.12	0.41	0.15	0.73	0.08	0.40	0.14

In next subsections, a figure is shown comparing the average values (red bars) and standard deviation (blue lines in the middle of the average) obtained from students who approved and those who did not approve the course for each source code metric, DomJudge result, and MSLQ feature (from Figure **4-1** to Figure **4-31**). The comparisons of the three semesters are shown from left to right for 2015-I, 2015-II and 2016-I.

4.1.1. Length based metrics

This kind of metrics are focused on measure the length (or amount) of a feature in the source code. The length based metrics used for this thesis are described next and are calculated per student. Later the students' results are divided in those who approve a those who did not, to be able to compare.

The amount of files, denoted as F, is the complete amount of files sent by a student, it does not matter if the code compiles or not, neither if the solution is correct or not. A comparison can be seen in the Figure **4-1**, in all the three semesters the students who approved sent in average more files than those who did not approve. In the semesters 2015-II and 2016-I the standard deviation is high, around 20 %, which means there were students who did not approve but who did several attempts trying to solve the assignments in the online judge.

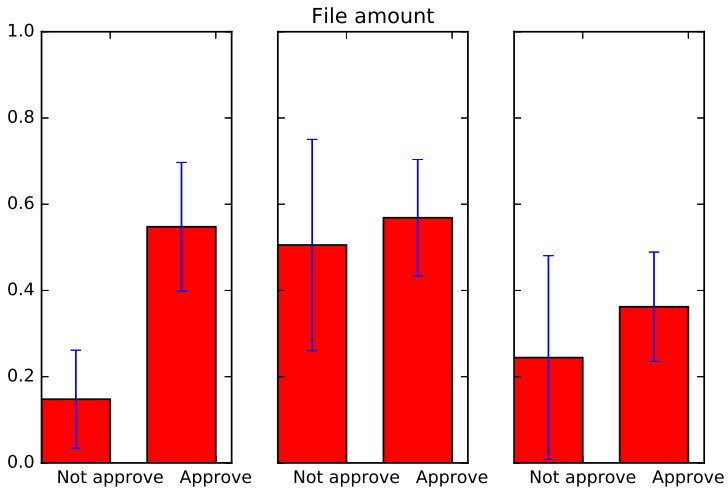


Figure 4-1.: Comparison among semesters 2015-I, 2015-II and 2016-I of file amount

Moreover, the average source lines of code (S_a), as can be seen in equation (4-1), refers to the total lines of code including empty lines, comments, and effective lines of code; where N is the total number of lines of code, and F is the total number of files. Figure **4-2** shows that the students which approved have in average more lines of code per file than those who did not approve, which is a similar behavior to the file amount.

$$S_a = \frac{N}{F} \tag{4-1}$$

The average class number by file (C_a), calculated using the equation (4-2), where c_f is amount of classes in a source code file, and F is the total number of files. In Figure **4-3** can be seen that again the students which approved have more classes in a file in average than those

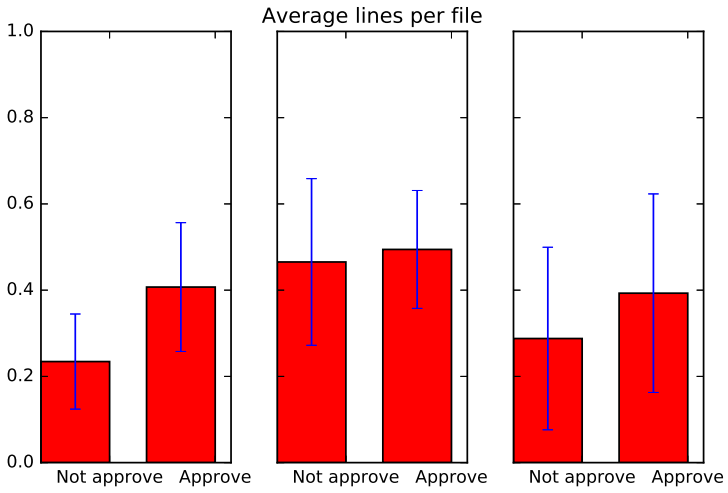


Figure 4-2.: Comparison among semesters 2015-I, 2015-II and 2016-I of average lines per file

who did not approve. In this case the difference is less evident. This can be explained by the nature of the language and the nature of the problems proposed in the assignments. Java allows one class per file, the other classes should be nested classes, which in many cases are needed, because solutions must be sent to the online judge in a single file.

$$C_a = \frac{\sum c_f}{F} \tag{4-2}$$

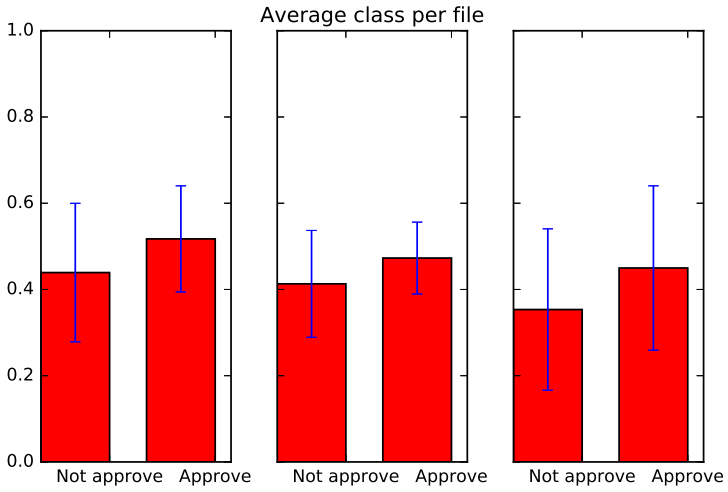


Figure 4-3.: Comparison among semesters 2015-I, 2015-II and 2016-I of average classes per file

The average source code lines by class (S_c), in equation (4-3), is calculated adding the source code lines in a class (n_c), and dividing it by the total amount of classes in all files C. In

Figure **4-4** is shown the comparison of this metric among semesters. Again a pattern in all long based metrics so far is kept, the students who approved have more average than those who did not. This behavior persists in the three semesters.

$$S_c = \frac{\sum n_c}{C} \tag{4-3}$$

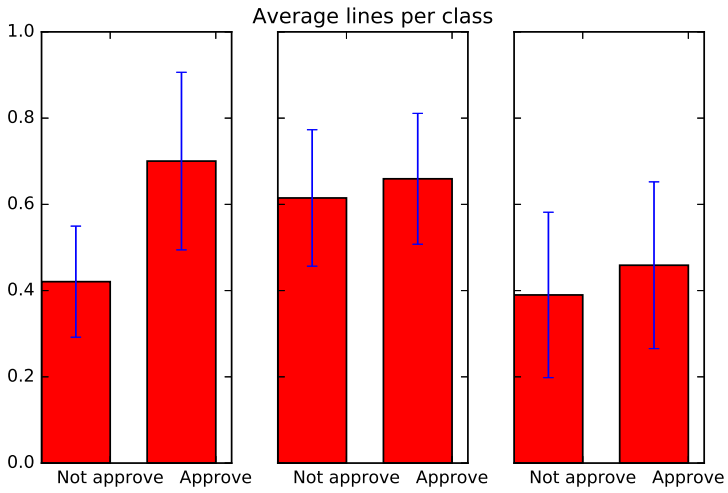


Figure 4-4.: Comparison among semesters 2015-I, 2015-II and 2016-I of average lines per class

The average attributes by class (A_c), in equation (4-4), is calculated by adding all the attributes in each one of the classes and dividing it by the total number of classes. The comparison among semester is shown in Figure **4-5**. While the trend of other long based metrics is kept, this metrics is more balanced in the semester 2015-II. In semester 2016-I the standard deviation is higher.

$$A_c = \frac{\sum a_c}{C} \tag{4-4}$$

The average methods by class (M_c), in equation (4-5), is calculated by adding all the methods in each one of the classes and dividing it by the total number of classes. Shown in the Figure **4-6**, this metric breaks the pattern seen in the other length based metrics. In semester 2015-II the students who did not approve presented more methods per class than the better performing students. However, the standard deviation is higher than in the approved group, which could mean that was not a common feature for all the students.

$$M_c = \frac{\sum m_c}{C} \tag{4-5}$$

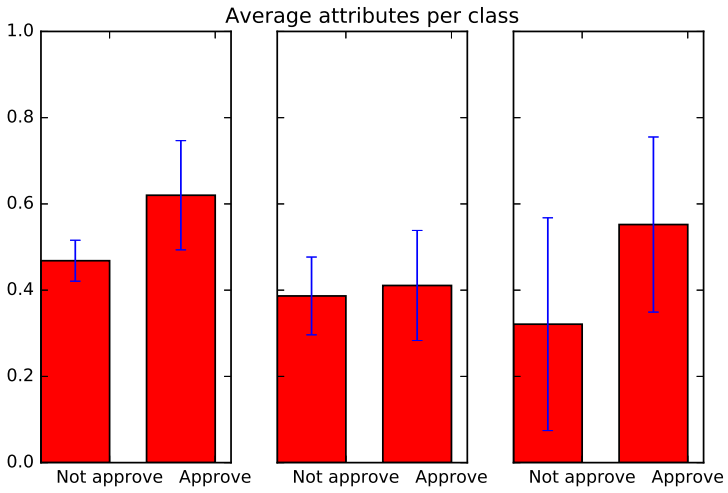


Figure 4-5.: Comparison among semesters 2015-I, 2015-II and 2016-I of average attributes per class

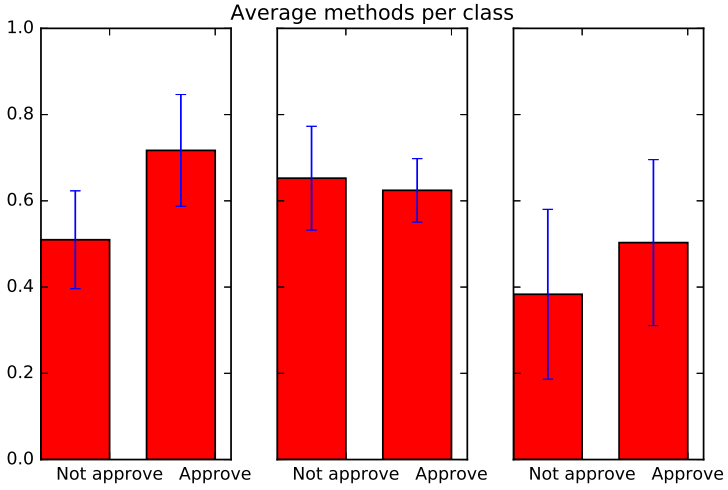


Figure 4-6.: Comparison among semesters 2015-I, 2015-II and 2016-I of average methods per class

The average amount of static attributes (L) and the average amount of static methods (O), in equations (4-6) and (4-7), are calculated adding the amount of static attributes in a class (l_c) or the amount of static methods in a class (m_c), and dividing them by the number of classes. The comparisons are shown in Figures **4-7** and **4-8**, it can be seen that the difference between approved and not approved is minimal in both metrics. In addition, the standard deviation is high enough to consider the difference insignificant.

$$L = \frac{\sum l_c}{C} \tag{4-6}$$

$$O = \frac{\sum m_c}{C} \tag{4-7}$$

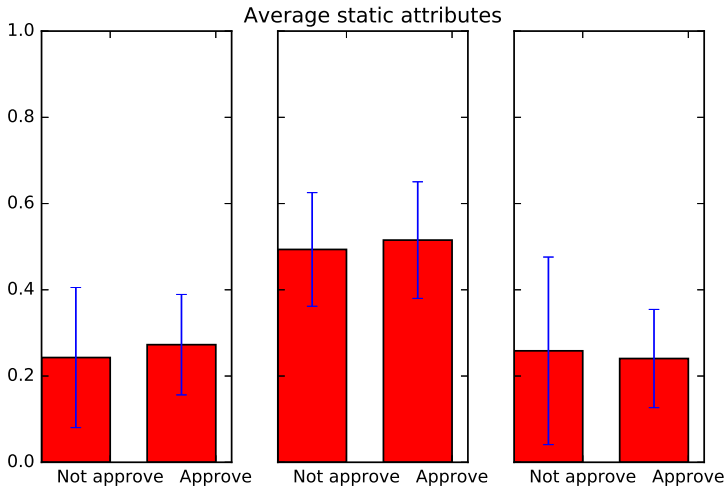


Figure 4-7.: Comparison among semesters 2015-I, 2015-II and 2016-I of average static attributes per class

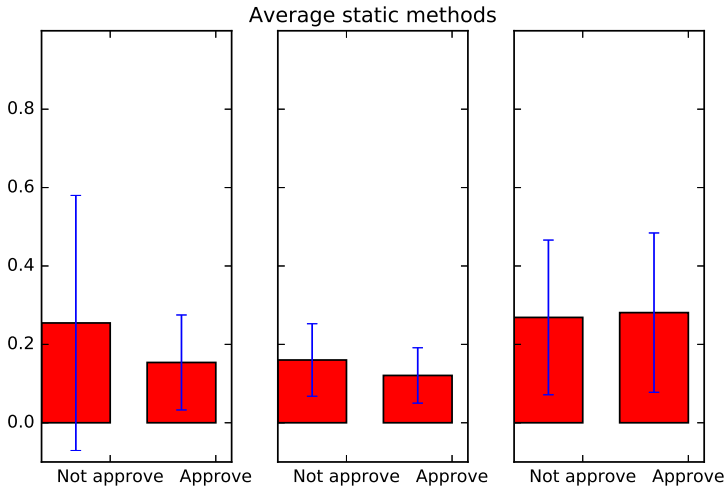


Figure 4-8.: Comparison among semesters 2015-I, 2015-II and 2016-I of average static methods per class

The average identifier length (Q), as seen in equation (4-8), is the addition of the length of all identifiers. This includes variable names, attribute names, parameter names but no the class names. And divided by the total number of classes. The comparison between approved and not approved is shown in Figure **4-9**. Students who approved have longer identifiers,

which could mean that use meaningful names. However, the standard deviation is too high, indicating that not all approved students used long identifiers and not all students who did not approve used short identifiers.

$$Q = \frac{\sum q_c}{C} \tag{4-8}$$

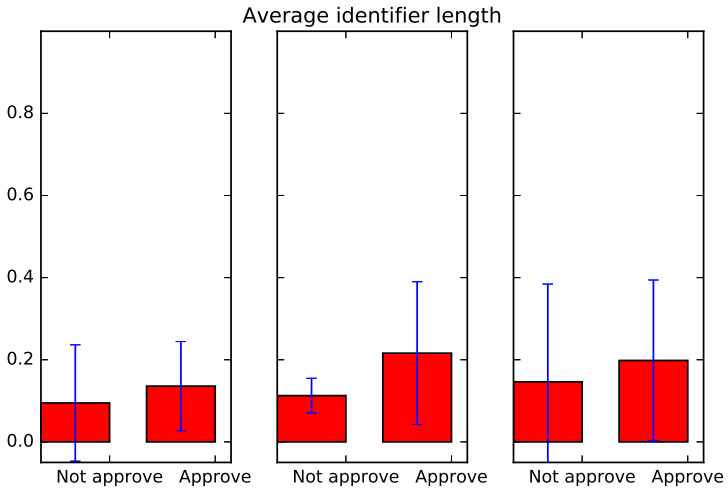


Figure 4-9.: Comparison among semesters 2015-I, 2015-II and 2016-I of average identifier length

The average class name length (K), in equation (4-9), is calculated by length of each class name, k_c, and dividing it by the total number of classes. In Figure **4-10** can be seen that the students who approved had greater average. Nevertheless, the difference is not significant as the standard deviation is bigger than the difference.

$$K = \frac{\sum k_c}{C} \tag{4-9}$$

The average parameters per method (P) is calculated obtaining the average amount of parameters of the methods and divided by the total number of methods, as seen in equation 4-10. The comparison of results for this metric is shown in Figure **4-11**. Better performing students used, in average, more parameters in their methods. Nonetheless, the standard deviation is too high, indicating that this is no specific to approved or not approved students.

$$P = \frac{\sum p_m}{M} \tag{4-10}$$

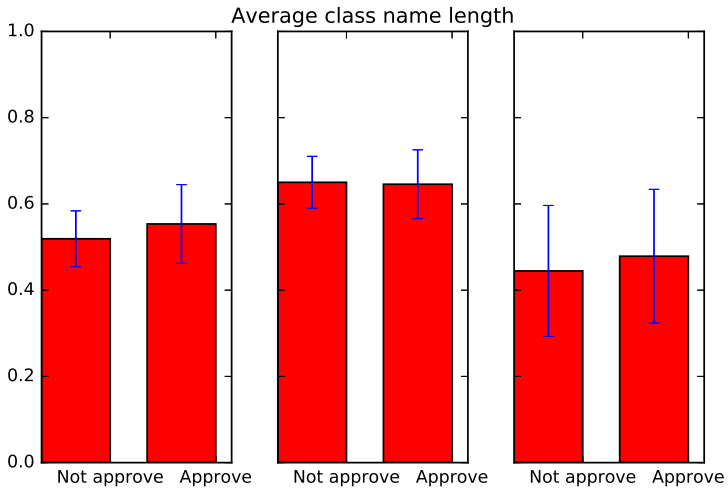


Figure 4-10.: Comparison among semesters 2015-I, 2015-II and 2016-I of average class name length

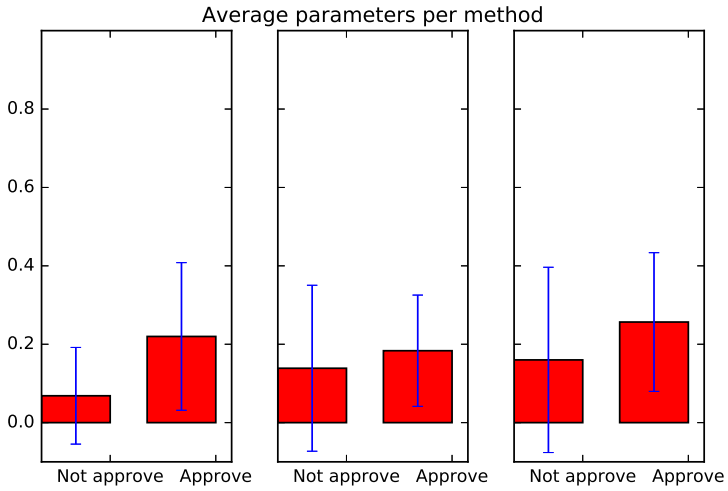


Figure 4-11.: Comparison among semesters 2015-I, 2015-II and 2016-I of average parameters per method

4.1.2. Complexity based metrics

This kind of metrics are focused on a concept called complexity [McCabe, 1976], which is an indicator of source code quality and the algorithm execution time.

The average amount of *if* (I), *if-else* (X), *for* (L) and *while* (W) statements. In equations (4-11), (4-12), (4-13) and (4-14) respectively. All the equations follow the same pattern, add the amount of statements (*if*, *if-else*, *for*, *while*) and divide it by the total amount of methods (M). The comparison between approved and not approved is shown in Figures **4-12**, **4-13**, **4-14** and **4-15** respectively. In the if and if-else clauses the difference is not significant, but

it can be seen that the average is a little bigger for those who did not approve. The same pattern can be seen in the *for-loops* and *while-loops*, but the main difference is related to the standard deviation, which is higher specially for the not approved average.

$$I = \frac{\sum i_m}{M} \tag{4-11}$$

$$X = \frac{\sum x_m}{M} \tag{4-12}$$

$$L = \frac{\sum f_m}{M} \tag{4-13}$$

$$W = \frac{\sum w_m}{M} \tag{4-14}$$

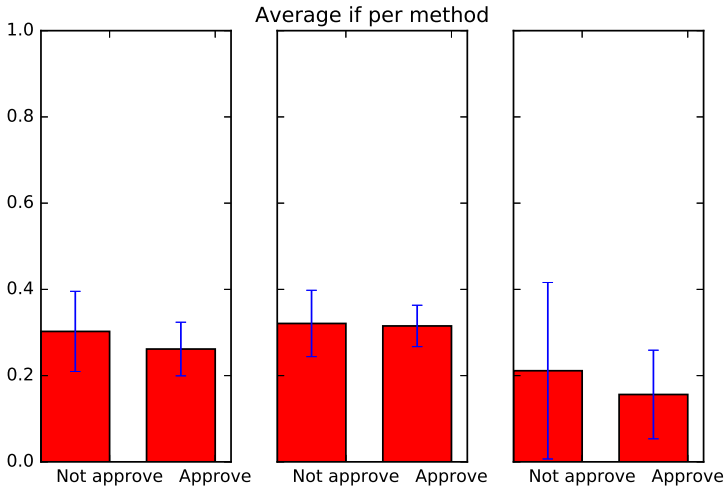


Figure 4-12.: Comparison among semesters 2015-I, 2015-II and 2016-I of average if clauses per method

In addition, the cyclomatic complexity (shown in Figure **4-16**), which can be seen as a summary of the previous discussed metrics, shows that even when it is tried to keep some elements of it lower (like the loops) the complexity in some of the codes is higher than the individual measures of the loops. Still, the values for the approved and not approved students is too close and the standard deviation indicates that there is no significant difference.

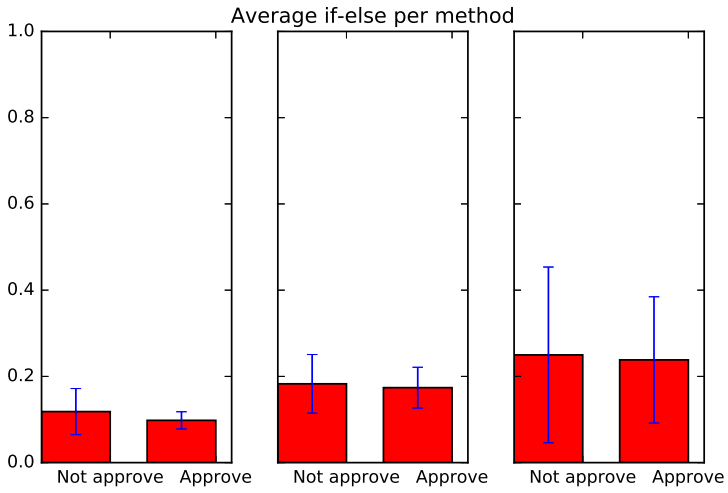


Figure 4-13.: Comparison among semesters 2015-I, 2015-II and 2016-I of average if-else clauses per method

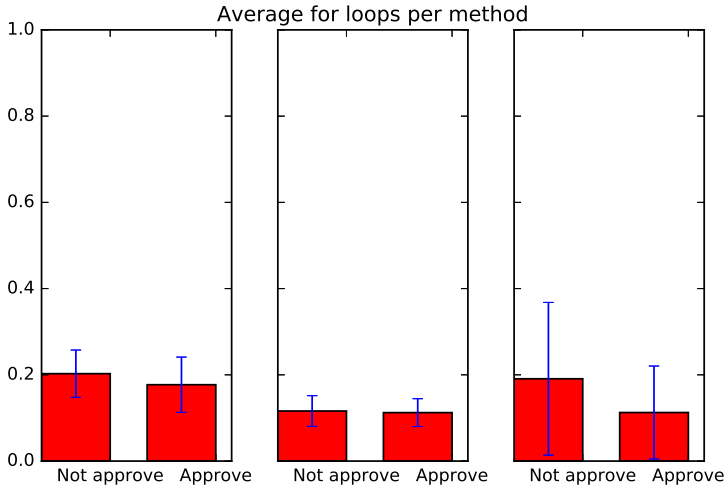


Figure 4-14.: Comparison among semesters 2015-I, 2015-II and 2016-I of average for loops per method

The Halstead metrics [Halstead, 1977] consist of another way to calculate complexity in a source code. The base to calculate these metrics are the operands (e.g., identifiers, numbers) and operators (e.g., keywords, ++, +). Vocabulary (n, described in Equation 4-15) consists in the sum of the unique operators (n_1) and operands (n_2). *Length* (N, described in Equation 4-16) is the sum of the total number of operands (N_1) and operators (N_2).

$$n = n_1 + n_2 \tag{4-15}$$

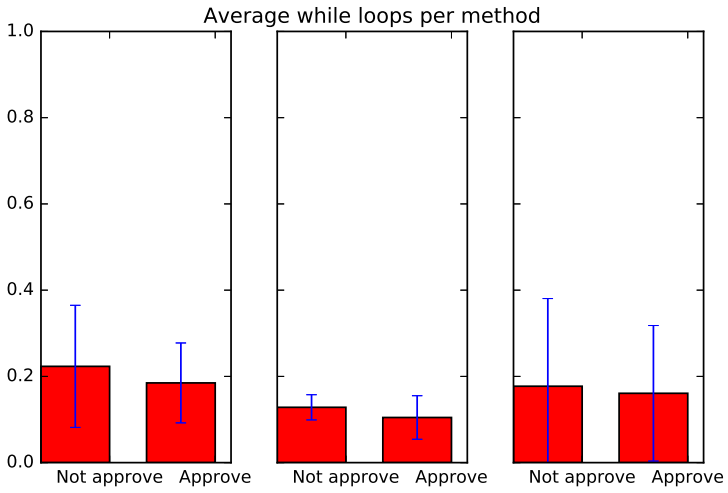


Figure 4-15.: Comparison among semesters 2015-I, 2015-II and 2016-I of average while loop per method

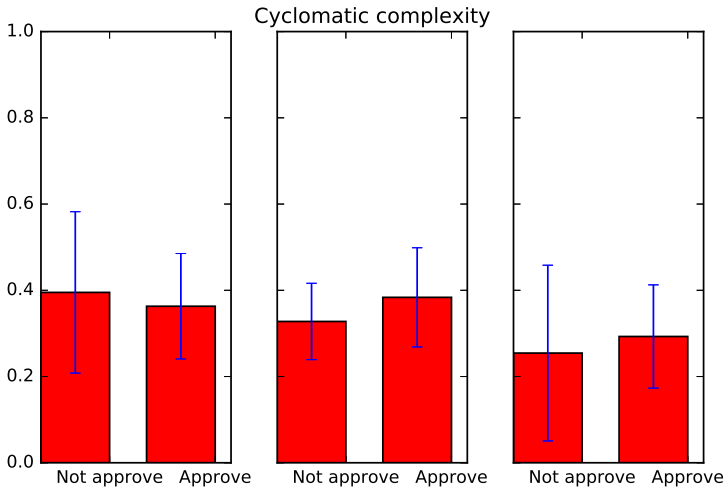


Figure 4-16.: Comparison among semesters 2015-I, 2015-II and 2016-I of average cyclomatic complexity

$$N = N_1 + N_2 \tag{4-16}$$

The Halstead *volume* (V), described in Equation 4-17, is a measure of size but it is also interpreted as the number of mental comparisons that were needed to write a program with length N. Moreover, the *difficulty* (D), shown in Equation 4-18, describes the difficulty to write a program. It is highly related to *volume* because as it increases the *difficulty* also does.

$$V = N \log_2 n \tag{4-17}$$

$$D = \frac{n_1}{2} \cdot \frac{N_2}{n_2} \tag{4-18}$$

In Figures **4-17** and **4-18** are shown the comparisons of the Halstead volume and difficulty, respectively. It can be seen that the students who did not approve have a higher average than those who did. Additionally the high standard deviation indicates that even with the high average not all the not approved students followed the same pattern. For those who did approve, the low standard deviation indicates that in this case there is a common result.

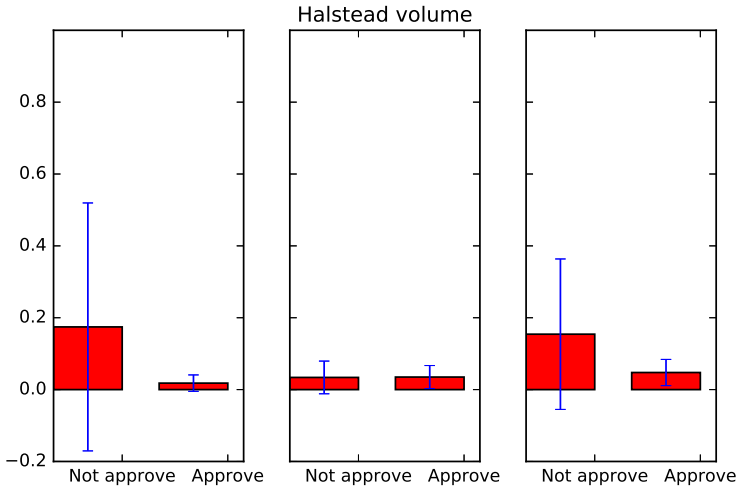


Figure 4-17.: Comparison among semesters 2015-I, 2015-II and 2016-I of Halstead volume

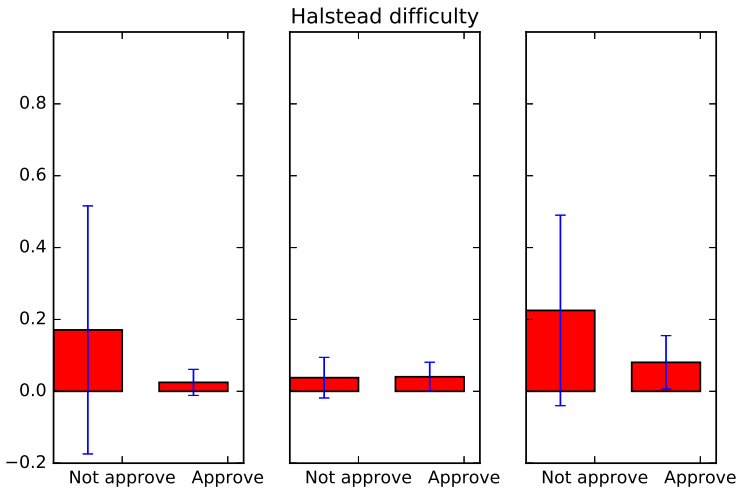


Figure 4-18.: Comparison among semesters 2015-I, 2015-II and 2016-I of average Halstead difficulty

The *effort* (E) described in Equation 4-19, indicates the effort required to write a program of high difficulty. In Figure **4-19** the comparison is shown, again the students who did not approve got a bigger value in this metric, but the standard deviation is really high.

$$E = D \cdot V \tag{4-19}$$

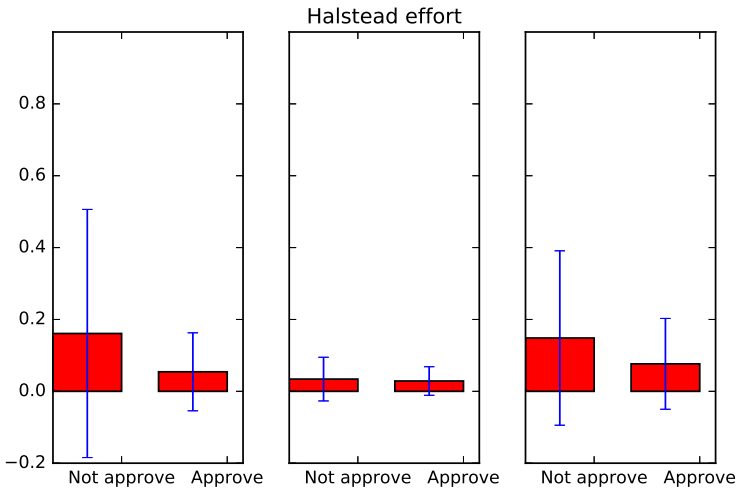


Figure 4-19.: Comparison among semesters 2015-I, 2015-II and 2016-I of average Halstead effort

Finally, the *effort* is the base to calculate the *time to understand/implement* (T) and *bugs delivered* (B), as can be seen in Equations 4-20 and 4-21, respectively. The *time* metric is related to the Stroud number [Shen et al., 1983], which is the *"number of elementary discrimination per second"*. Stroud claimed that this number ranges from 5 to 20, but the Halstead's experiments indicated empirically that the best number in this case was 18.

$$T = \frac{E}{18} \tag{4-20}$$

$$B = \frac{E^{\frac{2}{3}}}{3000} \tag{4-21}$$

In Figures **4-20** and **4-21** are shown the comparisons of Halstead time to understand or implement, and bugs delivered, respectively. As in all other Halstead metrics the students who did not approve have a bigger average value of this metrics. The standard deviation indicates that is not a "rule" but is still significant because all the Halstead metrics are an indicator of quality. Therefore, one could expect to find less quality in the code of students who did not approve the course.

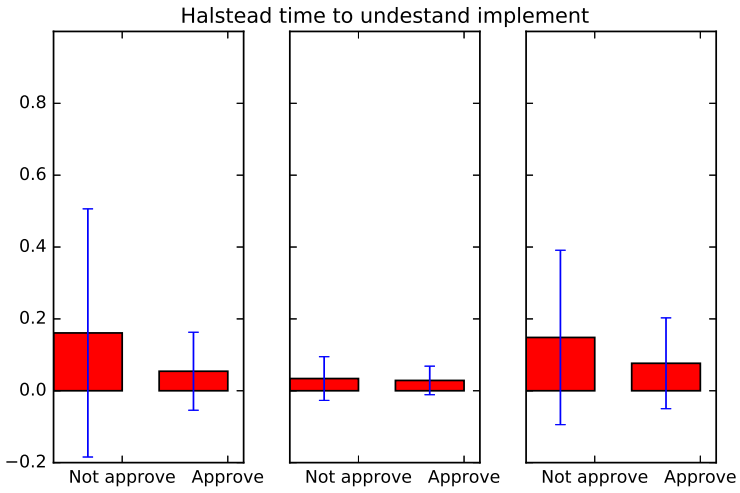


Figure 4-20.: Comparison among semesters 2015-I, 2015-II and 2016-I of average Halstead time to understand or implement

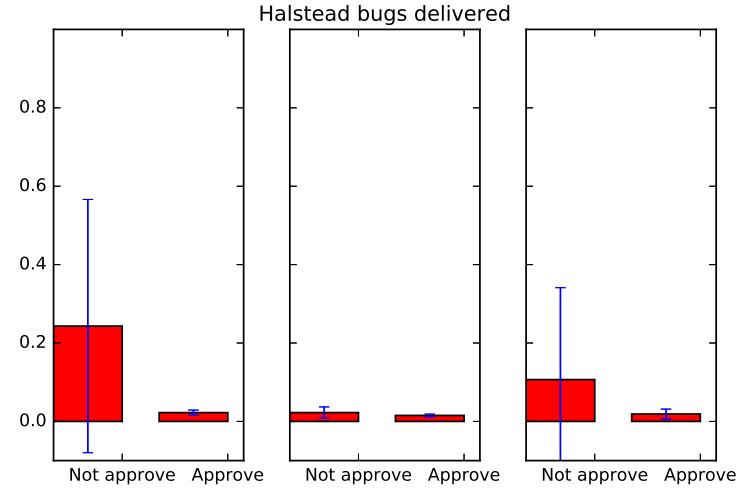


Figure 4-21.: Comparison among semesters 2015-I, 2015-II and 2016-I of average Halstead bugs delivered

4.2. Program execution results

As part of the dataset, the program execution results from Domjudge are also extracted. Given the total number of files sent (F), the total number of files: correct (R), wrong (W), which hit time limit (T), with compilation errors (E), execution errors (X), and wrong

output (O). The averages are extracted as seen in equations (4-22).

$$\begin{aligned} \bar{R} &= \frac{R}{F} & \bar{W} &= \frac{W}{F} & \bar{T} &= \frac{T}{F} \\ \bar{E} &= \frac{E}{F} & \bar{X} &= \frac{X}{F} & \bar{O} &= \frac{O}{F} \end{aligned} \tag{4-22}$$

Figure **4-22** illustrates the comparison of the average of correct files sent by students. Interestingly, in 2015-II and 2016-I the average of correct files is higher for those students who did not approve compared to the ones who approve the course. However, this could have an explanation, the judge do not punish wrong submissions, so a student which approve may have sent a lot of wrong attempts, causing a decrease in the general average. In addition, the standard deviation is high, so not all of those who approved sent a lot of attempts.

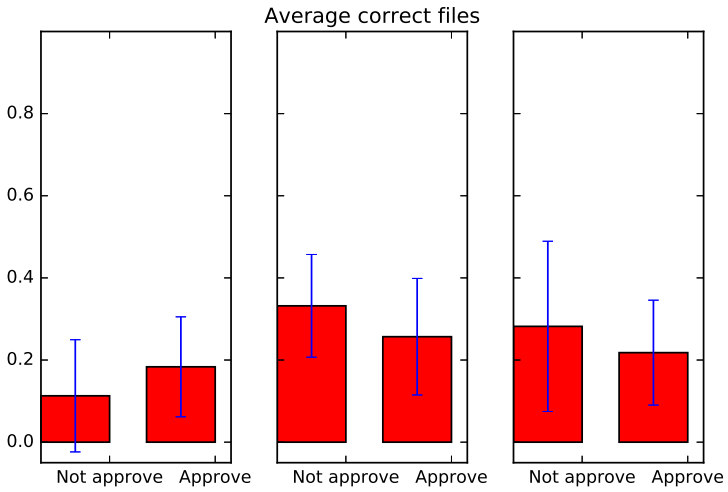


Figure 4-22.: Comparison among semesters 2015-I, 2015-II and 2016-I of average correct files

In other words, the average value is obtained based on the total amount of files sent, if the student sent few files but most of them were right could cause this result. This frequently happens with students who did not drop out early in the course on time, and decided to continue enrolled to the course without being really committed to it.

Moreover, Figure **4-23** shows the comparison of the files with errors among semesters. The standard deviation is so high that it can not be conclusive as an indicator.

In Figure **4-24**, it can be seen that approved students have in average more files which hit time limit, but the values are close to the not approved, and standard deviation is big enough

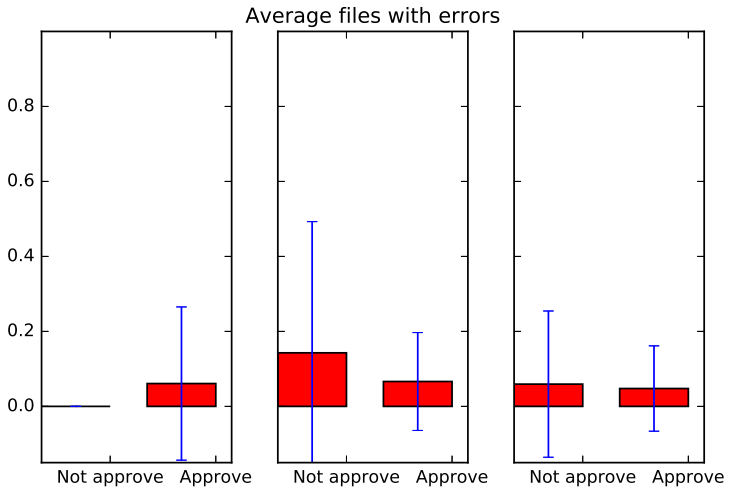


Figure 4-23.: Comparison among semesters 2015-I, 2015-II and 2016-I of average files with files

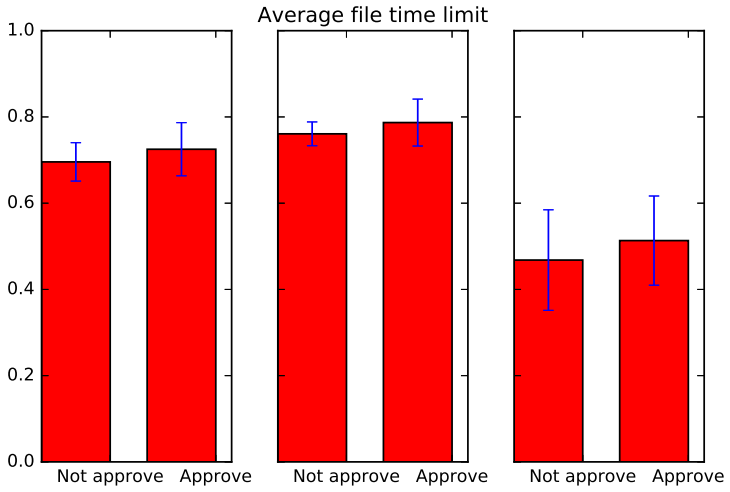


Figure 4-24.: Comparison among semesters 2015-I, 2015-II and 2016-I of average files which hit time limit

to consider the difference not significant.

Figure **4-25** presents that the higher average is for those who approved, this can be explained by the fact that there is no penalty per attempt be correct or wrong, this is also an indicator that the students who approved are more persistent.

In Figure **4-26**, is presented the comparison between students who approved and those who did not during the three semesters studied. In this case it can be seen that is not statistical difference, apparently the execution error rate is indifferent to the final result. This could be

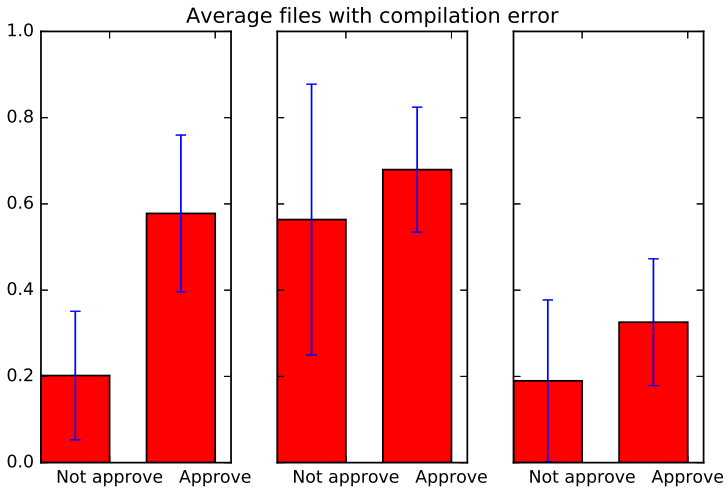


Figure 4-25.: Comparison among semesters 2015-I, 2015-II and 2016-I of average files which had compilation error

explained by the fact that there was no penalty in send a file with this characteristics.

Figure 4-26.: Comparison among semesters 2015-I, 2015-II and 2016-I of average files which had execution error

Figure **4-27** shows a higher value for those who did not approve in all the studied periods. Still, the standard deviation is high enough to not consider this as statistically significant. An important detail is seen in 2016-I, the standard deviation have a much more higher value. Possibly due to the number of students.

In Figure **4-28**, the amount of correct files comparison is presented. At first sight this may seem contradictory, not approved students got higher values in average. Firstly, results on

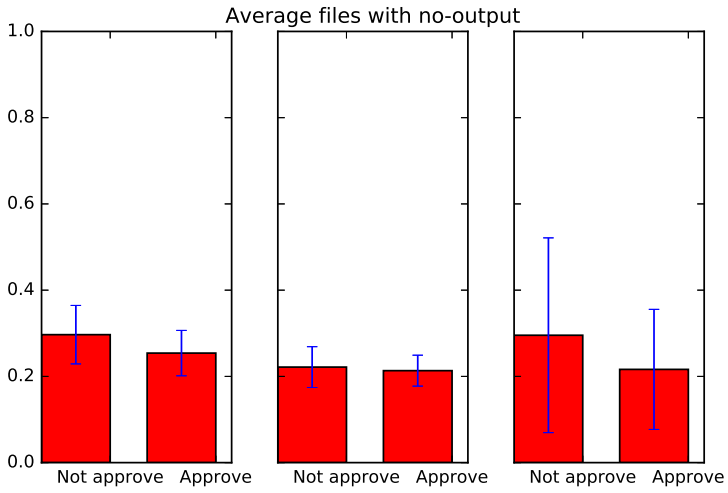


Figure 4-27.: Comparison among semesters 2015-I, 2015-II and 2016-I of average files which had no output during execution

average for the 2015-I and 2015-II semesters are very close between the better performing students and those who did not approve the course, and therefore, there is no statistically significant difference to draw conclusions. With regard to the 2016-I results, conclusions can not be drawn either because the standard deviations are very high. Secondly, it is important to clarify that the practical component of the course qualification has a value of 50 % of the final grade; the other grades correspond to two partial exams.

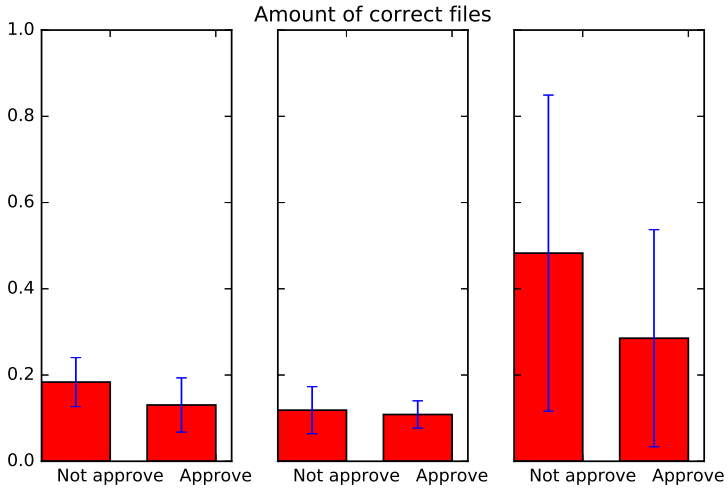


Figure 4-28.: Comparison among semesters 2015-I, 2015-II and 2016-I of total amount of correct files

Finally, Figure **4-29** presents the final grade comparison. The obtained results were the expected ones. However, some details deserve attention. The average grade in 2015-II and

2016-I was very close, around 2.0, the standard deviations show that there were students who were very close to the approval grade. In addition, in such semesters the students who did not approve did not appear to get too low grades. Moreover, it can be seen that approved students did not approve with the lower grade, but in average it was close to 4.0, and according to the standard deviation there were several students who obtained grades beyond 4.0.

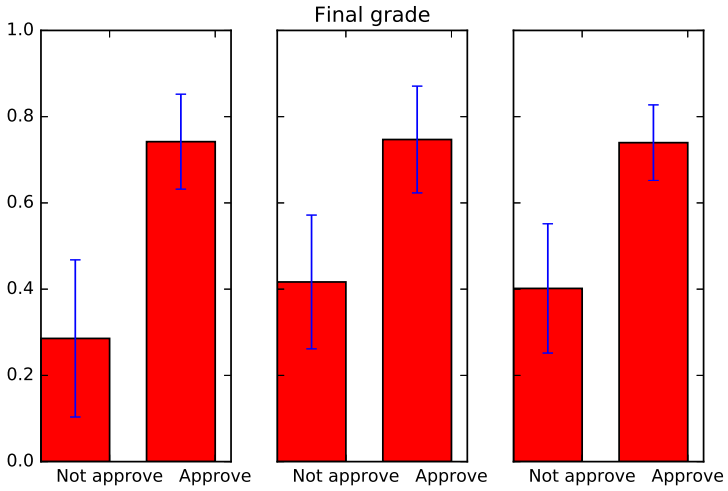
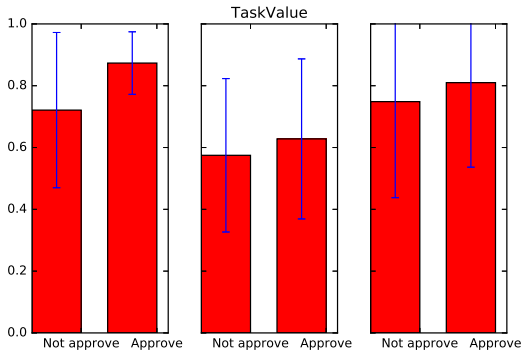


Figure 4-29.: Comparison among semesters 2015-I, 2015-II and 2016-I of final grade

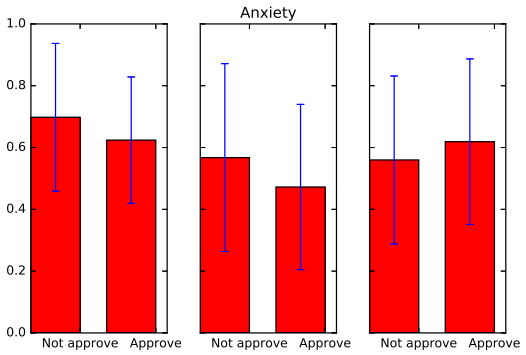
4.3. Motivational traits and learning strategies

In the three semesters a Motivated Strategies for Learning Questionnaire was applied during the fifth week of the semester, after some homework was previously done in the first weeks. The used questionnaire is an adaptation to Colombia proposed by Professor Jhon Jairo Ramirez Echeverry called MSLQ-Colombia [Ramírez-Echeverry et al., 2016]. After the test evaluation, 18 features are obtained which can be seen with a brief explanation in Tables **2-2** (7 motivational features) and **2-3** (11 learning strategies). In each item, students specify their level of agreement or disagreement on a symmetric agree-disagree Likert scale from 1 (strongly disagree) to 7 (strongly agree).

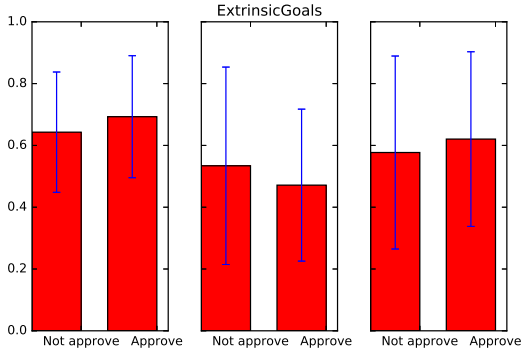
In the Figure **4-30** is shown the comparison of the seven motivational features in MSLQ. Task value (Figure **4-30**a), self efficacy learning (Figure **4-30**f) and self efficacy performance (Figure **4-30**g) presented a similar behavior; during the three periods the approved students had more average for these features. However, the students who did not approve presented also high values. The high standard deviations show that, even for the high averages, students answered very different values.



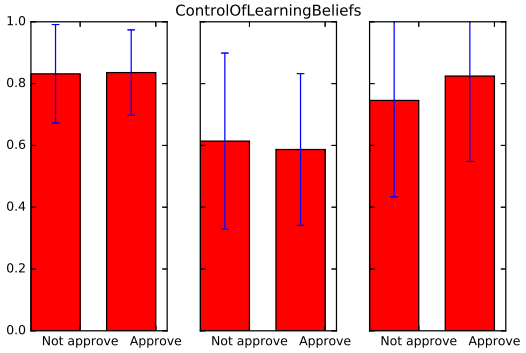
(a) Task value



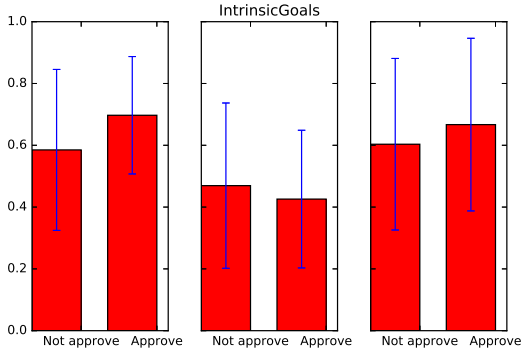
(b) Anxiety



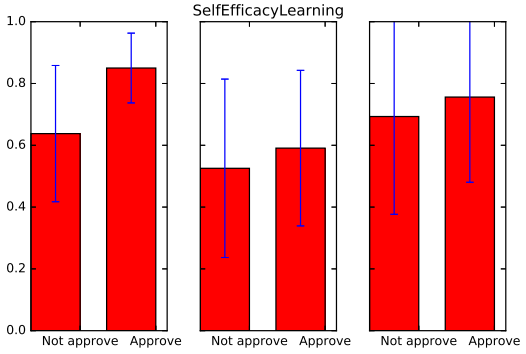
(c) Extrinsic goals



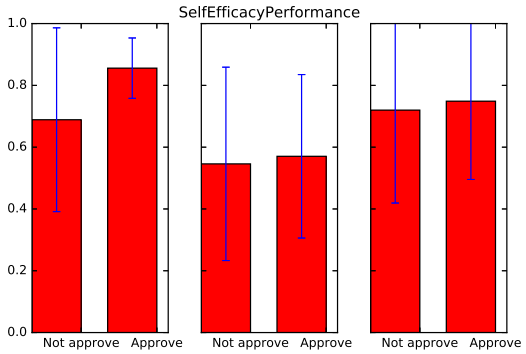
(d) Control of learning beliefs



(e) Intrinsic goals



(f) Self efficacy learning



(g) Self efficacy performance

Figure 4-30.: Comparison among semesters 2015-I, 2015-II and 2016-I

Figure **4-30**b presents the comparison of anxiety. In the periods 2015-I and 2015-II the students who did not approve had higher values of anxiety. Although it is an indicator to have in mind in future sections, the standard deviation is also high.

In addition, Figures **4-30**c and **4-30**d and **4-30**e show the comparison of extrinsic goals, control of learning beliefs and intrinsic goals, respectively, among semesters. In this case, during the periods 2015-I and 2016-I the students who did not approve had lower values, and the difference in the averages was very low. The standard deviation is high as well.

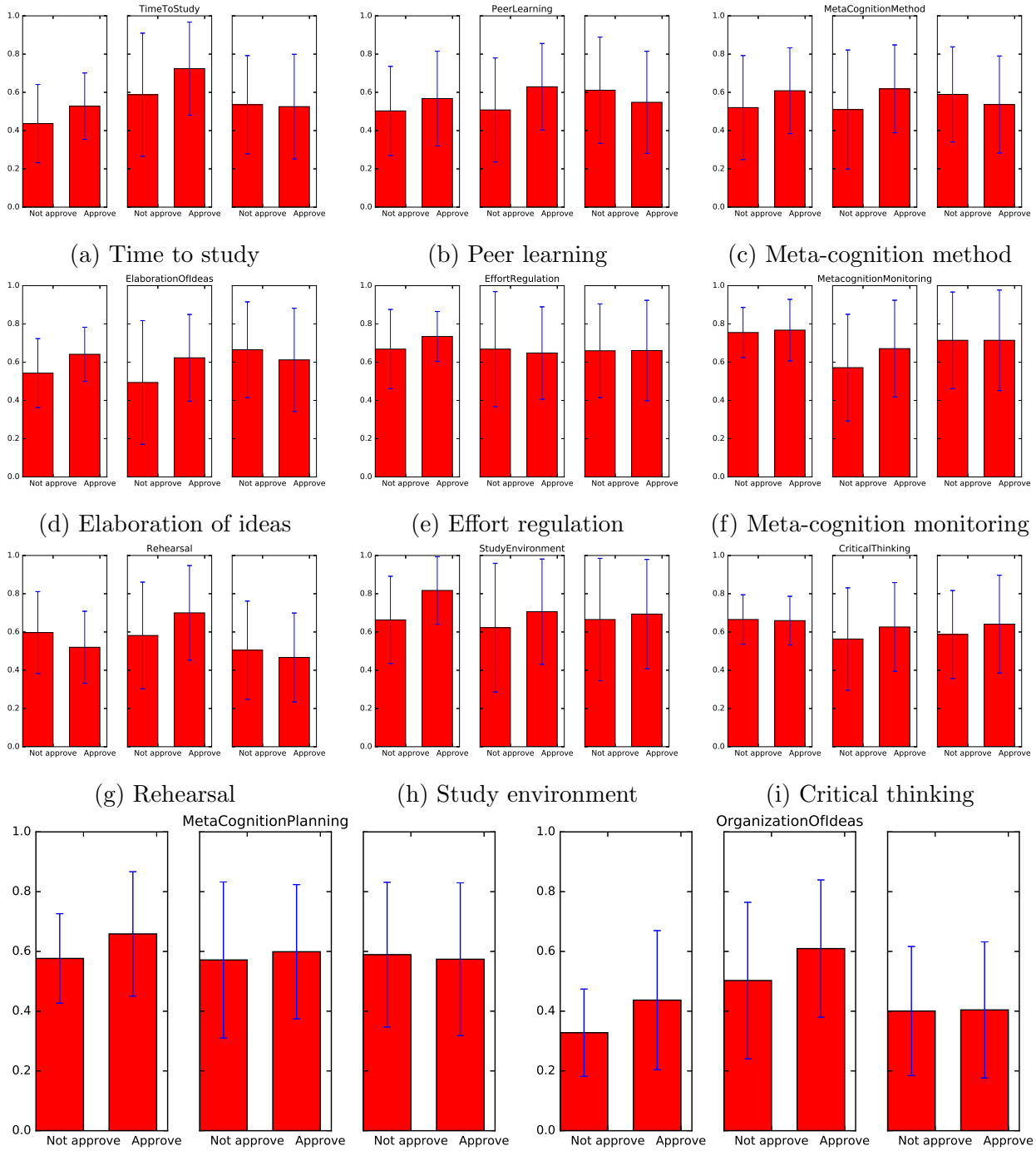
Moreover, in the Figure **4-31** is shown the comparison of the features related to learning strategies in MSLQ. In particular, the features time to study (Figure **4-31**a), peer learning (Figure **4-31**b), meta cognition method (Figure **4-31**c), and elaboration of ideas (Figure **4-31**d) show a similar behavior among semesters. In 2015-I and 2015-II approved students got higher averages, while in 2016-I the values are almost tied. The standard deviations are very high in all cases.

Regarding the features effort regulation (Figure **4-31**e), meta cognition monitoring (Figure **4-31**f), meta cognition planning (Figure **4-31**j) and critical thinking (Figure **4-31**i), it can be seen that between approved and not approved students the averages are almost tied. Again the standard deviations are high.

In the Figure **4-31**g, is shown the comparison of rehearsal among semesters. Keeping the trend of other motivational features, the standard deviation is high. The semesters 2015-I and 2016-I present higher average for students who did not approve, overcoming by little margin the approved students.

In the Figures study environment (Figure **4-31**h) and organization of ideas (Figure **4-31**k) the average values values are close for those who approve and those who did not. However, the standard deviation is higher than the differences in all semesters.

During this chapter, the dataset was presented comprehensively. This allows to get insights of the results obtained from the students. Source code metrics showed more clues about student final results than MSLQ features. In more metrics the standard deviations were usually lower comparing to the measures in MSLQ features. Furthermore, some patterns persisted through semesters. For instance, the average identifier length was higher for the students who approved. In MSLQ features this kind of analysis was not clear because the standard deviations were always high; the highest or lowest average not in all cases belong to the approved students; and, in some cases, the averages between approved and not approved in all semesters were too close.



(a) Time to study (b) Peer learning (c) Meta-cognition method

(d) Elaboration of ideas (e) Effort regulation (f) Meta-cognition monitoring

(g) Rehearsal (h) Study environment (i) Critical thinking

(j) Meta-cognition planning (k) Organization of ideas

Figure 4-31.: Comparison among semesters 2015-I, 2015-II and 2016-I, the line in blue represents the standard deviation.

Taking into account the presented results, it is difficult to draw conclusions. This motivates the necessity to explore the dataset supported by machine learning techniques as it will be

presented in next chapters of this thesis. In addition, the relationships between source code metrics (technical features) and MSLQ features is yet to be seen in the following chapters.

5. Correlations

This chapter presents and discusses the correlations results among source code metrics, motivational and learning strategies features, and students performance. In the Section 5.1, only the technical correlations are shown and relationships are presented among them, including source code metrics and students performance. Later, in the Section 5.2, the correlations between technical features (including performance as well) and the MSLQ features are explored. Finally, a discussion is done about the obtained correlations and possible reasons of the relationships found.

To properly calculate the correlation coefficients, a normal distribution test was performed over the values. In case of normality, the Pearson product-moment correlation coefficient was calculated, otherwise, the Spearman correlation coefficient was used.

It is worth mentioning that correlations among all features will be represented by means of heatmaps to facilitate the results interpretation. In this way, heatmaps clearly show which features are correlated positively or negatively. Positive and negative correlations are represented in blue and red colors, respectively. The darker the color is, the correlation is closer to 1,0 (dark blue) or $-1{,}0$ (dark red). This representation is used in this chapter from Figure **5-1** to Figure **5-4**. For detailed values of the correlation coefficients please refer to Appendix A from Table **A-1** to Table **A-6**. The correlations highlighted in bold in these tables mean that they are statistically significant, having a p-value $\leq 0{,}05$.

5.1. Technical correlations

Using the source code metrics (technical information) jointly with the students performance data, a correlations analysis was performed. This shows traces of possible relationships among the elements in the dataset.

The correlations between technical features in semester 2015-I can be visualized in the heatmap presented in Figure **5-1**. This figure presents the correlation of each technical feature against each other. That is the reason because the diagonal of the heatmap shows a correlation of 1,0. The numbers at left side (vertical axis) and on bottom (horizontal axis) of the heatmap correspond to the numbers of the metrics presented in Table **3-1** in the Chapter 3.

In addition, the number 28 corresponds to the final grade, which is the students performance. For detailed values see Table **A-1** in Appendix A.

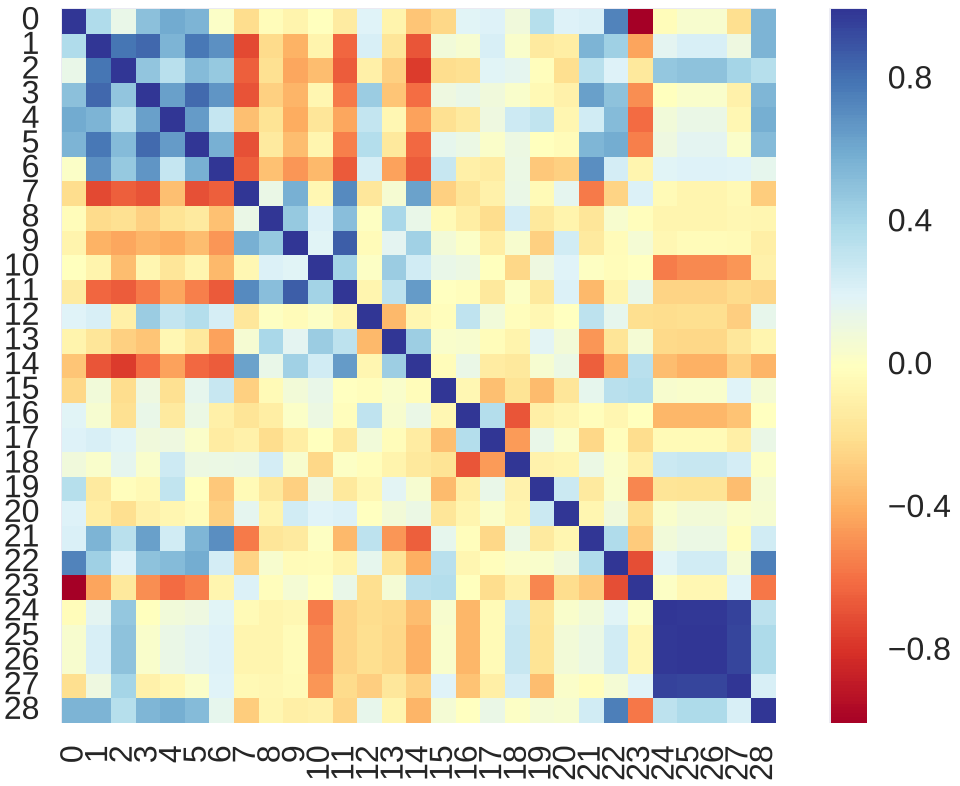


Figure 5-1.: Heatmap of correlations among technical features in semester 2015-I, the numbers of the metrics are presented in Table **3-1**. Positive correlations are presented in blue, negative correlations in red.

As it can be seen, there are some important positive correlations among metrics related with source code length (0 to 6), from now on Length metrics. Most of them have a p-value $\leq$ 0,05. Another important region is presented between Length metrics and the region related with Complexity (7 to 11), from now on Complexity metrics. This means that most of these metrics are correlated to the cyclomatic complexity metric. In addition, the metric average of static methods has a negative correlation with Length metrics.

Furthermore, another clear and visible group of correlations among technical features can be seen in metrics related with Halstead (23 to 27). These metrics expose a strong positive correlation to each other. In addition, Complexity metrics are correlated negatively with

Halstead but these correlations are not as strong as the previous ones.

Regarding the students performance (28), it is correlated positively to Halstead effort and time to understand/implement. Also, there is an interesting negative correlation between final grade and Halstead bugs delivered. These metrics also present a p-value $\leq 0{,}05$.

In the same way as in the previous figure, the heatmap shown in Figure **5-2** presents the correlation for the semester 2015-II. To see in detail the correlation coefficients see Table **A-2** (Appendix A).

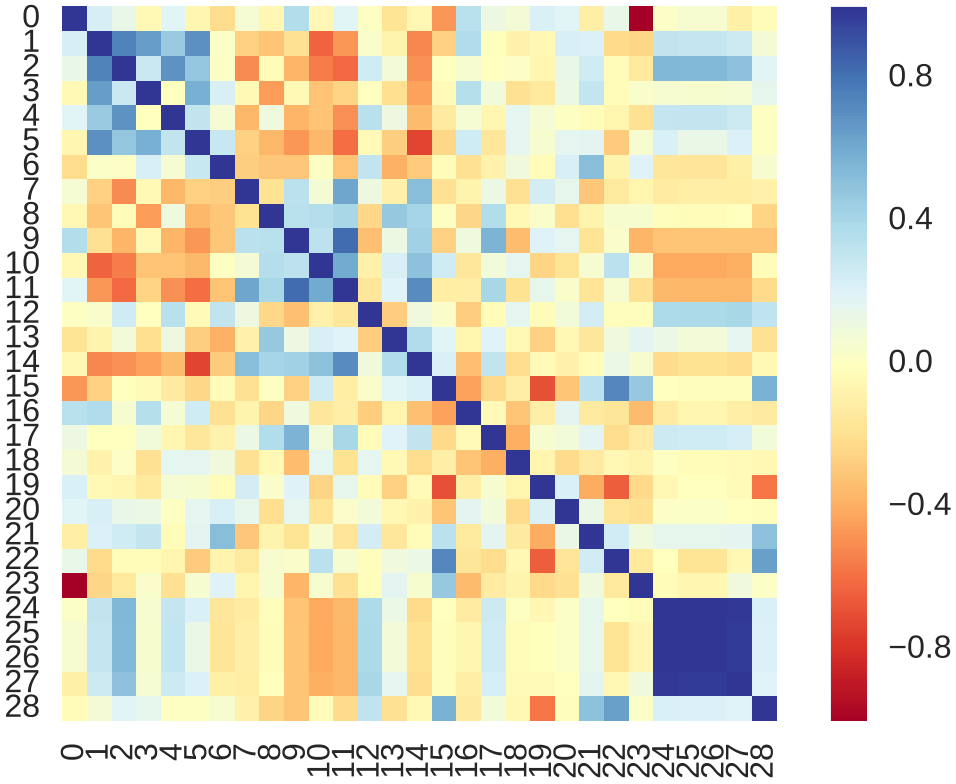


Figure 5-2.: Heatmap of correlations among technical features in semester 2015-II, the numbers of the metrics are presented in Table **3-1**. Positive correlations are presented in blue, negative correlations in red.

While it can be seen that the Length metrics group is still present showing high positive correlations with a p-value $\leq 0{,}05$. However, it is not as strong as in 2015-I. In addition, the complexity metrics are also present with less intensity. Halstead metrics appear again, but this time stronger than before. In this semester, the correlation between Halstead and

Complexity metrics is stronger as well. In this case, the p-values are $\leq 0{,}05$ and maintain the negative correlation.

Moreover, the correlations between metrics for semester 2016-I are represented in the heatmap in Figure **5-3**. To see in detail the correlation coefficients see Table **A-3** (Appendix A).

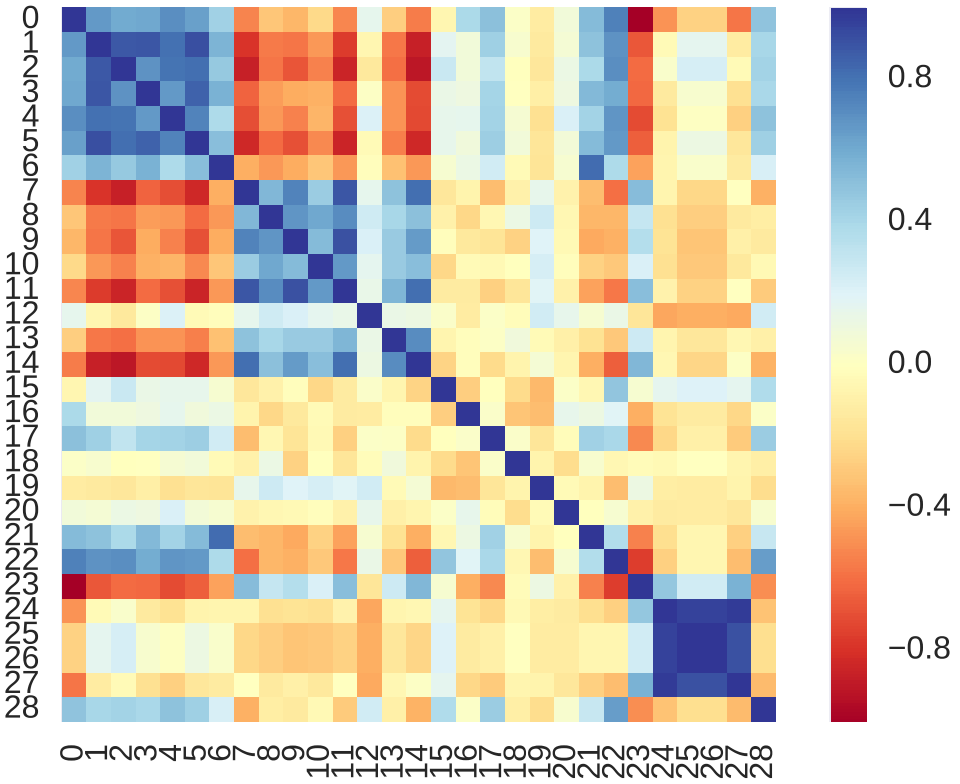


Figure 5-3.: Heatmap of correlations among technical features in semester 2016-I, the numbers of the metrics are presented in Table **3-1**. Positive correlations are presented in blue, negative correlations in red.

In this semester, the groups mentioned before are present once again with stronger correlations, specially Length metrics with respect to themselves, Length with Complexity, and Halstead metrics with themselves. The p-values for those groups indicate that the correlations are very significant. One of the possible explanations for this is that the amount of students in this semester was 101, more than double than in the other semesters.

As conclusions for this section, there are some points to highlight. First, the apparent pre-

sence of at least three groups of metrics: Length, Complexity and Halstead. This will be confirmed in the next chapter together with the existence of other group.

Another important aspect found is the persistence of the groups in the three semesters. Although the level of correlation varies, the presence of those correlations in all the semesters is an indicator of the importance of those metrics. Furthermore, the correlation of such metrics with respect to MSLQ features, which will be presented in the next section, can provide more useful information.

Finally, the correlation between the final grade and Length, Complexity and Halstead metrics show the impact of those metrics in the performance of students in the course. Notice that Halstead and most of Complexity metrics are correlated negatively, while Length metrics are correlated positively.

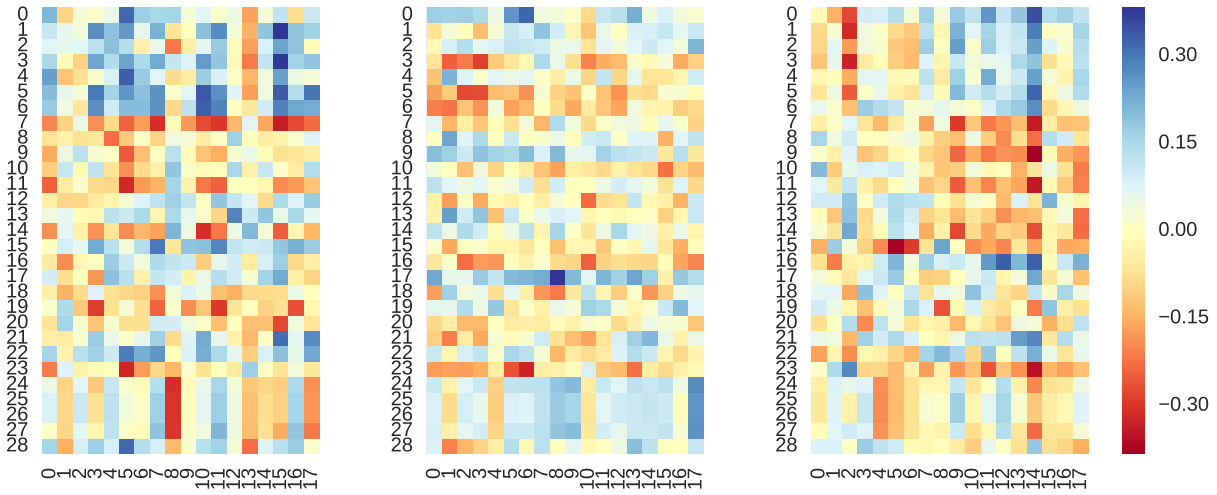
5.2. Technical and motivational correlations

This section presents and discusses correlations between technical (including source code metrics and performance) and MSLQ features. As in the previous section a heatmap for each semester is shown.

The heatmaps for all studied semesters are presented in Figure **5-4**, from left to right, the heatmaps correspond to 2015-I, 2015-II, and 2016-I. In this case, the numbers on the left correspond to the source code metrics (28 is again the students final grade) and the ones at the bottom of each heatmap correspond to the MSLQ features, numbered in Tables **2-2** and **2-3**. For detailed correlation coefficients see the Appendix A in Tables **A-4**, **A-5** and **A-6** for 2015-I, 2015-II, and 2016-I, respectively.
In semester 2015-I, the Length metrics can be easily seen in the heatmap correlated positively with several MSLQ features. This means that Length based metrics are not exclusively related to one or another MSLQ features. However, the p-values are not low enough to consider them significant. With respect to Halstead metrics, they appear to be mostly negatively correlated with motivational features. Nonetheless, similar to Length metrics, the p-values are not $\leq 0{,}05$, i.e., the correlations are not statistically significant.

In semester 2015-II, the Length metrics are not clearly seen in the heatmap, and some of them are negatively correlated with almost all MSLQ features. In this semester, Halstead metrics are more clearly visible, but in this case mostly correlated positively, and the correlations are not as high as before. Similarly to Length metrics, the p-values are not low enough.

In semester 2016-I, the Length metrics can be seen again in the heatmap, however with mixed results. They appear to be negatively correlated with motivational features, and mostly



(a) 2015-I (b) 2015-II (c) 2016-I

Figure 5-4.: Heatmap of correlations among technical and MSLQ features, numbers in the Y axis corresponds to metrics in Table **3-1**. Numbers in the X axis corresponds to Tables **2-2** and **2-3**. Positive correlations are presented in blue, negative correlations in red.

positively correlated with learning strategies features. Halstead correlations can not be seen clearly in this case, presenting mixed values, and in general, low correlation coefficients. Moreover, unlike the other two semesters, in 2016-I the Complexity metrics can be seen, they correlate negatively with several learning strategies, and have p-values $\leq 0{,}05$ (significant).

Contrarily to the correlations between technical features themselves, which showed clearly some regions that could be considered groups, the correlations between technical features and MSLQ features are not clear enough. The technical features showed inconsistent correlations with MSLQ features. However, this does not mean the lack of relationships, other methods can be used to find out how the two kind of features relate to each other. In the following two chapters of this thesis, machine learning methods will be applied to confirm certain groups, and to find hidden relationships between technical and MSLQ features, together with the students performance.

6. Clustering

Based on the correlations described in Chapter 5, different relationships are inferred between the source code metrics (technical features), and between technical features and MSLQ strategies. To explore more in detail these relationships, a hierarchical clustering using the Ward method was applied to: the technical features to all students, and technical and MSLQ features to the students grouped by semesters. The hierarchical clustering was chosen due to it is easier to visualize and interpret results than in other methods.

To apply the Ward method, it is needed that the input (I) has a positive number, thus, the formula 6-1 was used, where C is the correlation matrix.

$$I = 1 - |C| \tag{6-1}$$

With the obtained input, the hierarchical clustering with Ward method was applied and the resulting hierarchical clusters are presented and discussed in this chapter.

6.1. Hierarchical clustering of technical features

In the Chapter 5, it was inferred the existence of groups of source code metrics, due to the high correlation and low p-value between some of the metrics. Figure **6-1** shows the hierarchical clustering of technical features by means of a dendogram, note the numbers in the leaves corresponds to the same numbers in Table **3-1**.

First, three Halstead metrics appear on the left of the figure together: difficulty, effort, and time to understand/implement. The other two Halstead metrics (volume and bugs delivered) seem to be far, but a closer look reveals that Halstead volume is closer (in terms of similitude) to the first Halstead metrics than, for instance, the average amount of "if" clauses per method.

Another group that was not evident in the correlation analysis corresponds to the DomJudge information. Metrics like amount of correct files, no-output, runtime errors, files with compilation errors, and files which hit time-limit are very close. Only Total amount of files

Figure 6-1.: Hierarchical clustering of the technical features from all studied semesters using the Ward method. The numbers corresponds to metrics in Table **3-1**

and total amount of correct files are separated from the main DomJudge group.

The complexity related metrics, are distributed in the green and red branches. Still, creating groups inside such branches. The green branch includes: the "if" related clauses, the average amount of *while* clauses per method. In the red branch are close to each other cyclomatic complexity, and amount of *for* loops per method. An interesting observation can be made in that branch, the two mentioned complexity metrics are close to Halstead bugs delivered. This can be explained because Halstead itself is considered a complexity metric.

Finally, the Length based metrics are also close to each other. These are concentrated in the red branch of the hierarchy, where at least a complete sub-branch contains only Length metrics.

6.2. Hierarchical clustering of MSLQ questions

To corroborate the MSLQ features, based on the questions (items in the questionnaire). The clustering algorithm was also applied over the questions in MSLQ-Colombia. In Figure **6-2**, the clustering results are presented by means of a dendogram. To ease the reading of the figure, each one of the questions has as suffix matching the MSLQ feature which the question belongs to.

Notice that the questions of the same MSLQ feature tend to be together or close. For instance, see the leaves in the red branch. In particular, questions 4, 10, 17, 26, 27 which correspond to task value are grouped together. Questions 3, 8, 14, 19, 28 which correspond to anxiety are also together. Many of the other features tend to be together as well. Although some questions from one kind of feature appear in the middle of a group of other feature, the groups of features tend to be together or close to each other.

6.3. Bi-clustering of technical and MSLQ features

To facilitate the further analysis of correlations between source code metrics and MSLQ features, a bi-clustering algorithm was applied to this data. Clustering results are shown in Figures **6-3** (2015-I), **6-4** (2015-II), and **6-5** (2016-I). The figures presents a bi-clustering analysis of the correlations by means of a heatmap for each semester. The clustering of source code metrics is represented at the left side of each heatmap. In addition, a color in the left indicates the group of metrics each row belongs according to the hierarchical clustering results. The clusters corresponding to the MSLQ features are presented on top of each

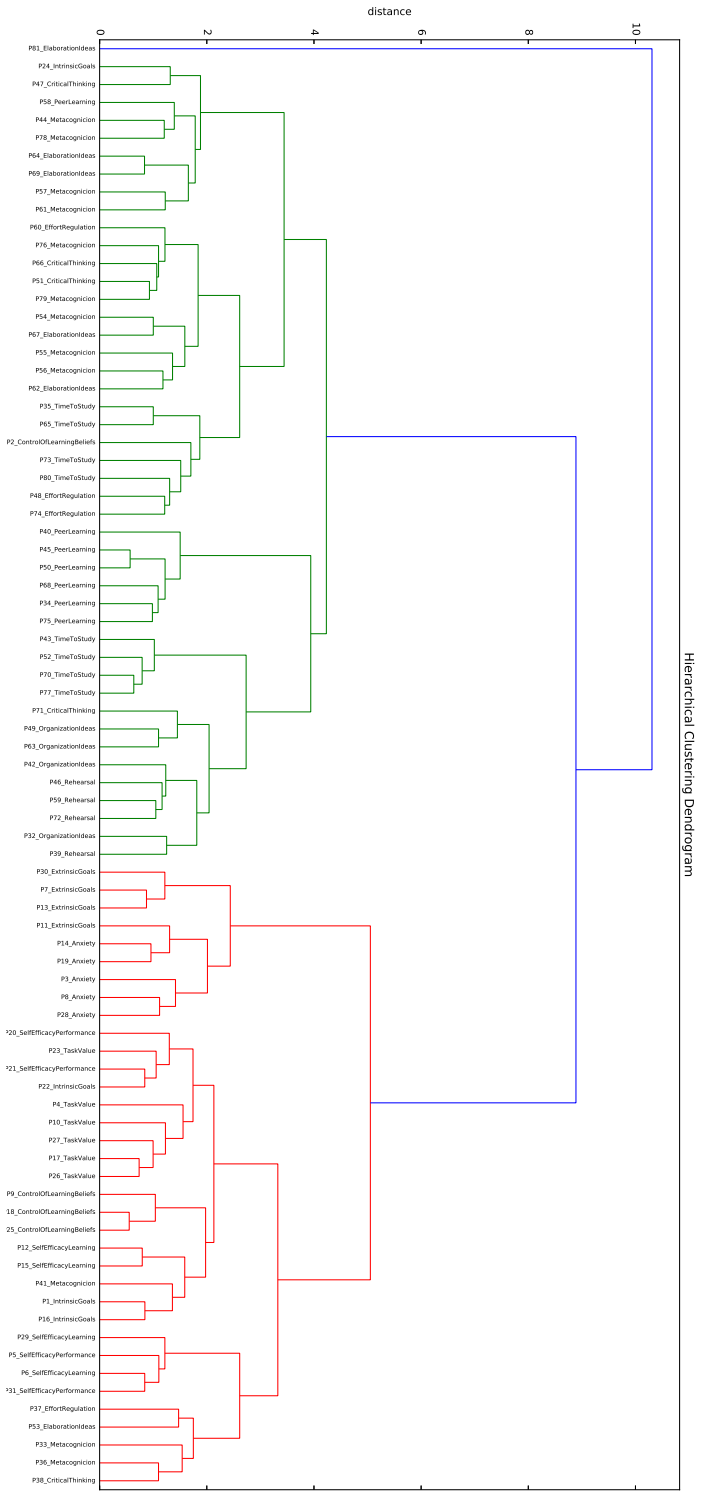


Figure 6-2.: Hierarchical clustering of the MSLQ features from all studied semesters using the Ward method

heatmap. Furthermore, the numbers on the vertical axis represent the source code metrics (right side of the figure), which correspond to the numbers of the metrics in Table **3-1**. The horizontal axis represents the MSLQ features (bottom of the figure), which are represented by the corresponding numbers in Tables **2-2** and **2-3**. Finally, the heatmap contains four highlighted regions of correlations which will be discussed next.

As mentioned before, clustering results show that several source code metrics tend to group together. For instance, this is the case in four of the Halstead metrics. Metrics related to length (like the *average number of lines of code*, and *average methods by class*, among others) are close to each other as well (see the gray color at the left side of heatmap in Figures **6-3**, **6-4** and **6-5**). Therefore, the source code metrics were grouped as follows:

- Length metrics (Gray): the amount of files, average source lines of code, the average amount of classes by file, average lines by class, averages attributes by class, average methods by class, average static attributes, and average static methods.
- Complexity metrics (Purple): average amount of *for* and *while* loops, average amount of *if* and *if-else* clauses, cyclomatic complexity, average identifier length, average class name length, and method parameter average.
- DomJudge result (Orange): average number of correct, wrong, time-limit, compiler error, execution error, and no-output solutions.
- Halstead (Green): bugs delivered, difficulty, effort, time to understand or implement, and volume.

The correlation study is done over the source code metrics (technical features) and the MSLQ features. Results are shown by means of the (previously mentioned) heatmap in Figure **6-3**, corresponding to semester 2015-I. Positive and negative correlations are represented in blue and red colors, respectively. The darker the color is, the correlation is closer to 1,0 (dark blue) or $-1{,}0$ (dark red). In addition, it is worth noting that regions of interest are highlighted using colors in the heatmap seen in Figure **6-3**, which corresponds to the main areas which will be discussed below (black at left, green on top, red in the middle, and orange the remaining one).

The black rectangle covers an area which contains mainly Length metrics, which corresponds to gray color on the left side. It also shows a positive correlation between these metrics and three self-regulation items: Control of learning beliefs (3), Study environment (15), and Organization of ideas (17). This may suggest that a student with source code characterized by high values in length-related metrics may believe that a positive outcome in the course depends of his/her own effort. Also, it could suggest a good management in the use of the

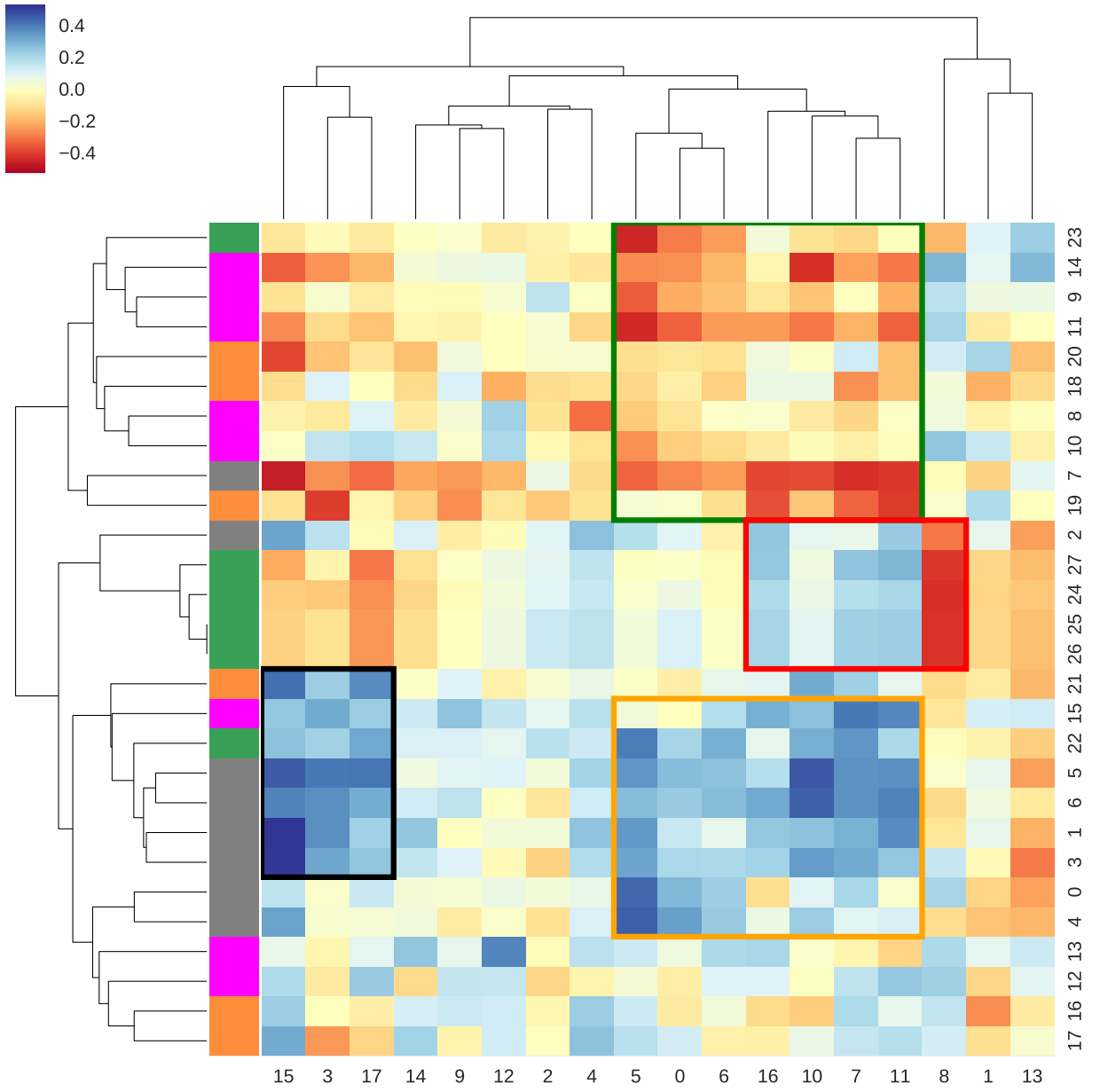


Figure 6-3.: Hierarchical bi-clustering between technical and MSLQ features for 2015-I. Numbers in the Y axis corresponds to metrics in Table **3-1**. Numbers in the X axis corresponds to Tables **2-2** and **2-3**. Positive correlations are presented in blue, negative correlations in red. Groups of metrics are differenciated on the left: Length (Gray), Complexity (Purple), DomJudge (Orange), and Halstead (Green).

aforementioned learning strategies.

The green region of interest is mainly composed of negative correlations involving two groups

of metrics (i.e., Complexity and DomJudge). Complexity has a strong negative correlation with self-efficacy learning (5). This fact may suggest little self-evaluation of the ability to master the course tasks in students whose code present high values in the metrics related to Complexity. Furthermore, there are two other motivational features involved, i.e., Task value (0) and Self-efficacy performance (6). As these features give an indication of the course importance to the students, this may suggest these students need more motivation during the semester.

The red highlighted region shows mainly Halstead metrics which are mostly correlated positively with learning strategies features. Time of study (7), elaboration of ideas (10), effort regulation (11) and meta-cognition planning (16) have a positive correlation with Halstead metrics. However, peer learning (8) have a negative correlation with this metric. This is particularly interesting because it may be indicating that learning with and from peers improve the value of these metrics, creating simpler code (in Halstead metrics the lower, the better).

Finally, the orange rectangle covers again correlations between metrics in the Length group and the same MSLQ features than the green region of interest. Unlike the green rectangle, in this case the correlation are mostly positive, particularly in the case of self-efficacy learning (5). This may mean that length-related metrics indicate that students increase the value of this kind of metric due to practicing (sending more files/attempts) or being more explicit/verbose in code (e.g., long and explicative identifiers). This, in particular, should be further studied to confirm this relationship and also the importance of frequent programming practicing in this kind of courses. In addition, learning strategies features have mostly a positive correlation, which could indicate a dependency between the use of these learning methods and the length metrics in the source code.

Moreover, in the Figure **6-4**, which shows the heatmap corresponding to semester 2015-II, three areas or interest are highlighted: black on top, green in the middle, and orange at the bottom. It is worth noting that during this semester the correlations were not as strong as in the other semesters. Still, some interesting results can be observed.

The black region contains mainly negative correlations. The strongest correlations correspond to Length metrics with motivational features.

The green region in the heatmap middle, corresponds mainly to DomJudge results and Complexity metrics, the strongest correlation appears with anxiety (1). Being this positive, it suggests that the amount of sent files and the higher Complexity values may be an indicator of anxiety in the student.

The orange region contains mostly positive correlations, two particular points are worth to

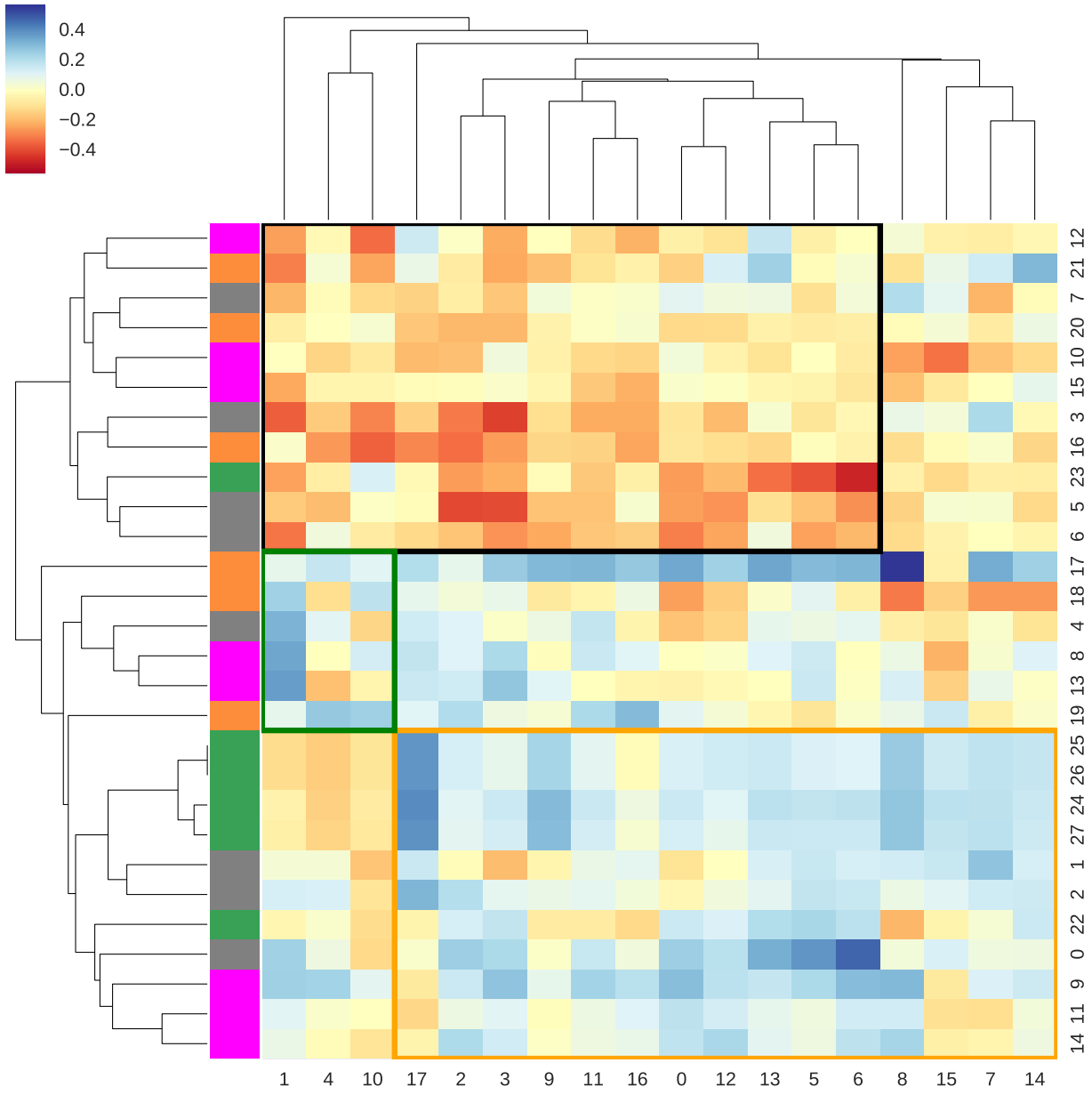


Figure 6-4.: Hierarchical bi-clustering between technical and MSLQ features for 2015-II. Numbers in the Y axis corresponds to metrics in Table **3-1**. Numbers in the X axis corresponds to Tables **2-2** and **2-3**. Positive correlations are presented in blue, negative correlations in red. Groups of metrics are differenciated on the left: Length (Gray), Complexity (Purple), DomJudge (Orange), and Halstead (Green).

mention in this region. Halstead metrics (corresponding to green color at the left) correlate positively with organization of ideas (17), and meta-cognition method (9). In addition, the amount of files has a strong correlation with self-efficacy learning (5), self efficacy performan-

ce (6), and critical thinking (13). In the case of self-efficacy learning in this region, it shows a positive correlation, even though it is not as strong as in 2015-I. This is an indicator of the consistent relationship between Length metrics with the mentioned motivational feature.

Finally, it is worth to mention that the metric Average time limit (17) has a strong correlation specially with learning strategies features. This is an indication about how was the approximation taken during 2015-II by the students to improve their learning strategies.

Figure **6-5** presents the heatmap corresponding to semester 2016-I. Two areas or interest are highlighted, black on top, green in the middle, and orange at the bottom. During this semester stronger correlations appear again.

In the black region, Length metrics are the most present group, which correlates positively with learning strategy features, and negatively with extrinsic goals (2) which is a motivational feature. This feature indicates that the students are expecting a reward (or in this case a good grade). Moreover, the strongest correlation is observed with critical thinking. This suggests the use of old knowledge to solve the assignments from the course.

DomJudge results are also observable in the black region. Although most of the correlations with self-regulation features are low, the correlations with meta-cognition monitoring (12) and critical thinking (14) have higher values, specially *average wrong*. Again, this correlation suggest the use of old knowledge to solve the current assignments, and additionally, this may suggest that the students use the submitted tasks as a method of self-testing.

In the orange region, is mainly composed by Complexity metrics (purple), most of the region corresponds to negative correlations. Being the only positive when correlated to anxiety. The strongest correlations between anxiety and Complexity are average of static attributes (12), average parameters (13), and average static methods (14). This is consistent with the results observed in 2015-II. The strongest negative correlations is between critical thinking (14) and Complexity metrics: average amount of *while* loops (8), average amount of *if* clauses (9), cyclomatic complexity (11), and average of static methods (14). This suggests that the use of old knowledge in the current task reduces code complexity. Additionally, Halstead bugs delivered also appears with a strong negative correlation. While no other Halstead metric appear in this region, the appearance of these metrics here is consistent with the hierarchical clustering seen in Figure **6-1** (2015-I).

The green region is mostly composed by Halstead metrics showing negative correlations. Control of learning beliefs (3) is the only self-regulated feature with positive correlations, indicating that students with low Halstead metrics expect a positive outcome. Moreover, the negative correlations are stronger with respect to intrinsic goals (4) and self-efficacy

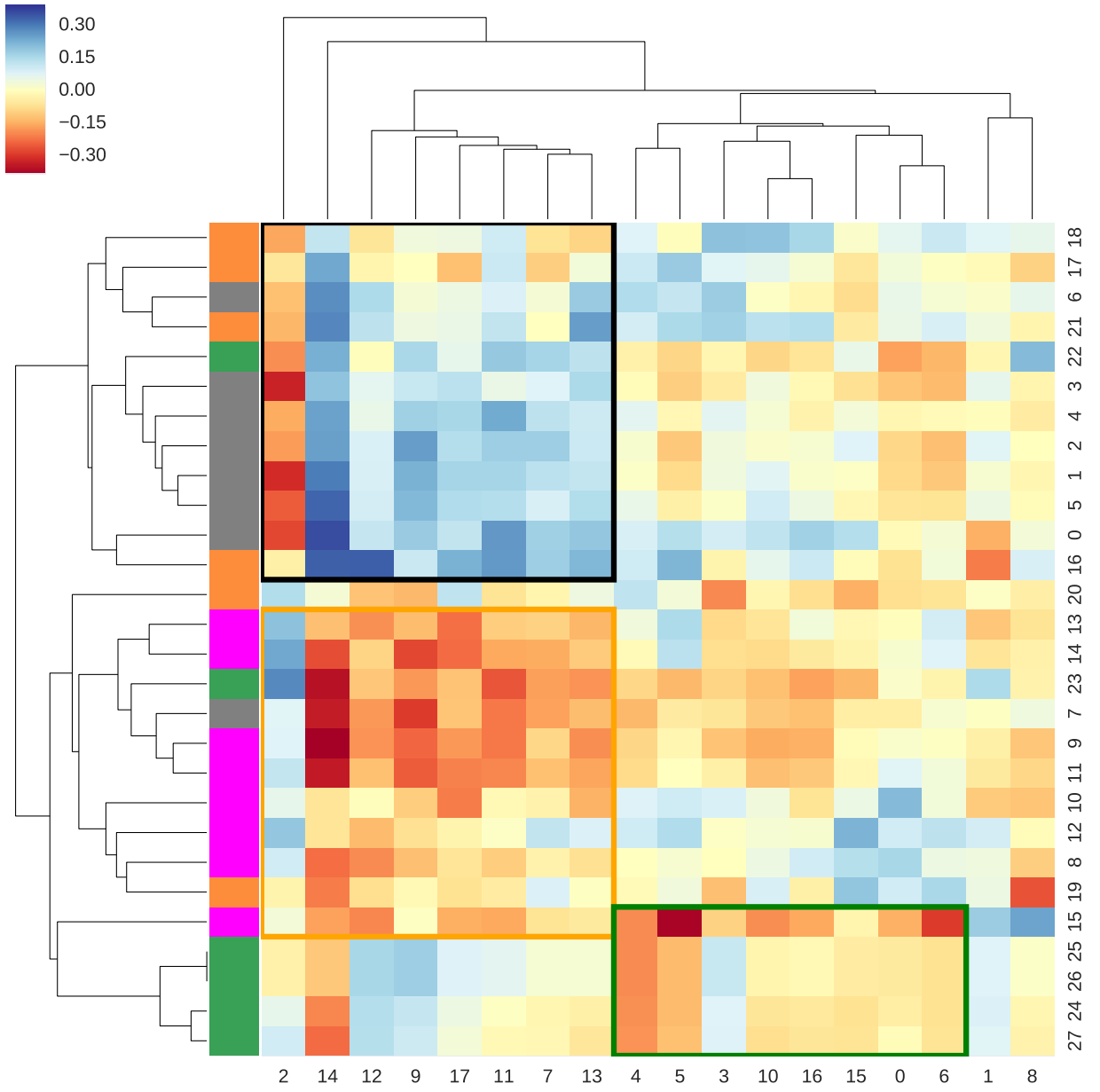


Figure 6-5.: Hierarchical bi-clustering between technical and MSLQ features for 2016-I. Numbers in the Y axis corresponds to metrics in Table **3-1**. Numbers in the X axis corresponds to Tables **2-2** and **2-3**. Positive correlations are presented in blue, negative correlations in red. Groups of metrics are differenciated on the left: Length (Gray), Complexity (Purple), DomJudge (Orange), and Halstead (Green).

learning (5), suggesting that students with low Halstead values like the subject and self-evaluate themselves constantly to improve. Furthermore, this region shows a better use of some learning strategies for those students with lower Halstead metrics in their source codes.

6.4. Correlations of groups of source code metrics, motivational learning strategies, and students performance

Using the groups obtained for the source code metrics, the correlation coefficients were again calculated between these technical groups (source code metrics) and MSLQ features, including also students performance (i.e., final grade). Results are presented in Table **6-1**. The first column includes the number of the MSLQ feature (see Tables **2-2** and **2-3**), and also the final grade. To focus only on statistically significant results, presented correlations correspond only to those coefficients with a p-value $\leq 0{,}1$. Correlations with an asterisk (*) in the Table corresponds more significant results where the p-value is $\leq 0{,}05$. In this way, features and semesters without any significant correlation were removed from the Table.

Table 6-1.: Correlation coefficients between MSLQ features (and students performance$-FG$) and groups of technical source code metrics

	Length			Compl.	DomJudge results			Halstead		
MSLQ	2015-I	2015-II	2016-I	2016-I	2015-I	2015-II	2016-I	2015-I	2015-II	2016-I
2			−0,22							
3		−0,27*			−0,33					
4							0,23*			−0,19
5	0,40*									
7	0,33							0,32		
8						0,30		−0,45*	0,27	
9				−0,21						
10	0,32									
11	0,30		0,17					0,33		
12		−0,28								
13			0,18							
14			0,23*	−0,22*						−0,19
15	0,53*									
17				−0,19				−0,32	0,38*	
Final Grade	0,62*		0,44*							−0,30*

Correlations with an asterisk (*) means a p-value $\leq 0{,}05$

As it can be seen in Table **6-1**, source code metrics related to length are highly correlated with MSLQ features and student performance in the course. In particular, effort regulation (11) and final grade (FG) show a positive correlation with the same tendency in two semesters (2015-I and 2016-I). These results suggest that length metrics have a correlation

with the students' performance in the class, which could be considered as a indicator to be monitored during the development of programming courses. Moreover, this kind of metrics has the biggest number of significant correlations with motivational and learning strategies, many of them with p-value $\leq 0{,}05$ (marked with *).

Moreover, Halstead metrics have during two semesters a significant correlation in peer learning (8) and organization of ideas (17). It can be seen that Halstead metrics have only one correlation with motivational features (4, intrinsic goals) and the rest are with learning strategies. This suggests that students exposing source code with high values in Halstead metrics (often associate to low quality code) may need to develop better learning strategies. This is consistent with the results observed in the Figure **6-3**, which suggests that students whose source codes present high Halstead metrics need to improve their use of different learning strategies; for example, the organization of ideas.

The complexity metrics have a negative correlation with three learning strategies: metacognition method (9), critical thinking (14), and organization of ideas (17). As a high complexity metric is a bad indicator in software quality, this correlation may be understood as an indicator of which learning strategies a student needs to improve. Therefore, this could be a way to identify students which may need additional help. Furthermore, such improvements may lead to improving Halstead metrics which are also correlated with organization of ideas (17).

The DomJudge results present correlations with control of learning beliefs (3) in 2015-I, peer learning (8) in 2015-II, and intrinsic goals (4) in 2016-I, suggesting mainly a relationship with motivational features. It is consistent with the data shown in Table **4-2**, where control of learning beliefs obtained the biggest average in 2015-I, and the maximum peer learning result was obtained in 2015-II. Intrinsic goals, however, did not obtain the biggest average in 2016-I, but it might be caused due to the standard deviation, as in that semester it was higher than the others due to the large number of students in that semester.

Finally, students performance, i.e., final grade shows correlations with length and Halstead metrics. These correlations have a p-value $\leq 0{,}05$, which is a strong indicator of the importance of those source code metrics in the students performance. As the correlation with Halstead metrics is negative, and the correlations with length metrics are positive, this could mean that good performing students write code with some of the following characteristics: easily readable, which causes a low value in Halstead metrics; with meaningful identifier names, which are more descriptive, causing a bigger value in length-related metrics. Therefore, measuring these metrics during the course could indicate which students may have problems to approve and may need additional assistance, which could be given based on other metrics. For instance, checking if the source codes have a high complexity, which suggests problems

in some of the learning strategies.

6.5. Spectral bi-clustering of technical and MSLQ features

To support the bi-clusters found previously, spectral bi-clustering was used. This is a partitional method. Contrary to the previous bi-clustering, which used the consolidated self-regulated features, here all the MSLQ items are used.

The spectral bi-clustering method needs the selection of the number of clusters for both set of features (metrics and self-regulated features). To calculate such numbers, the silhouette score was obtained based on the results of a K-Means algorithm with $K \in [2, 20)$, independently for technical features and MSLQ features. The input data used in this algorithm are the absolute values of the correlations, because a high correlation value are those close to 1 or -1. The silhouette value must be in a range of $[-1, 1]$. Values closer to 1 indicate a good match among the elements of the cluster. Therefore, K, is selected based on the higher silhouette value.

Figure **6-6** shows the silhouette measure for source code metrics. The horizontal axis represents the number of clusters, and the vertical axis represent the silhouette value. According to the figure, the number of clusters over source code metrics is 5.

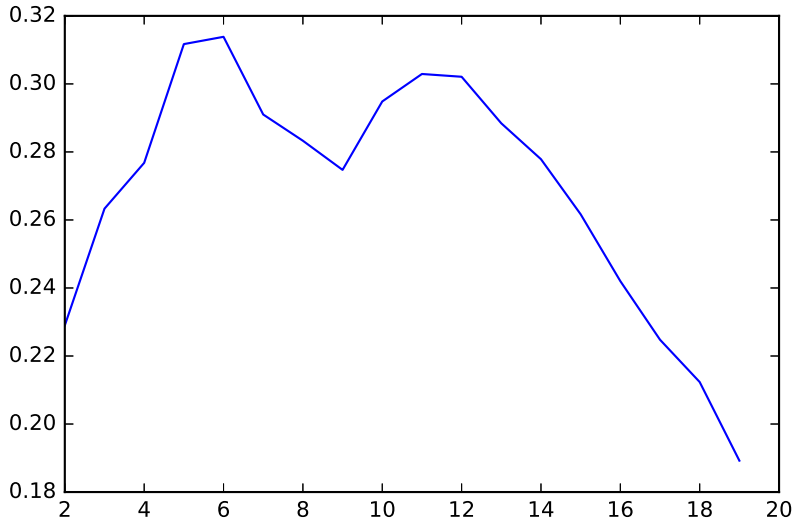


Figure 6-6.: Silhouette measures of source code metrics

In the Figure **6-7**, is shown the silhouette measure for MSLQ features. According to the figure, the best number of clusters over the MSLQ features is 2, which has the higher value.

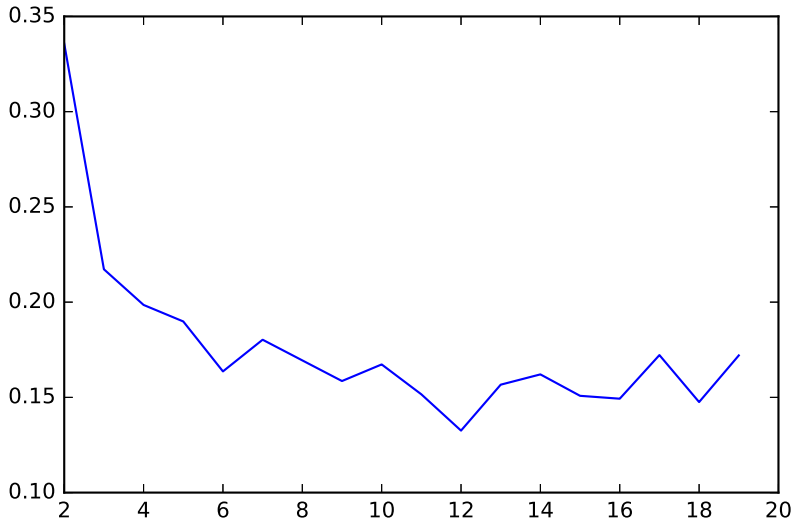


Figure 6-7.: Silhouette measures from self-regulated features

Once the number of clusters have been chosen, the spectral bi-clustering algorithm is applied. In the Figure **6-8** at top is shown the original data, at the bottom is the data after the bi-clustering. The horizontal axis represents the MSLQ questions, and the vertical axis includes the source code metrics. According to the number of cluster chosen, there are 10 clusters. To choose the most important clusters, all the values from each cluster are added, the results are presented in Table **6-2**. This indicates that the most important clusters are number 4, 5 and 1 (in descendant order).

It was found that the following source code metrics were present in the clusters 4 and 5 (top 2):

- Average while loops per method
- Average correct files
- Average identifier length
- Cyclomatic complexity
- Average of static attributes
- Average of static methods

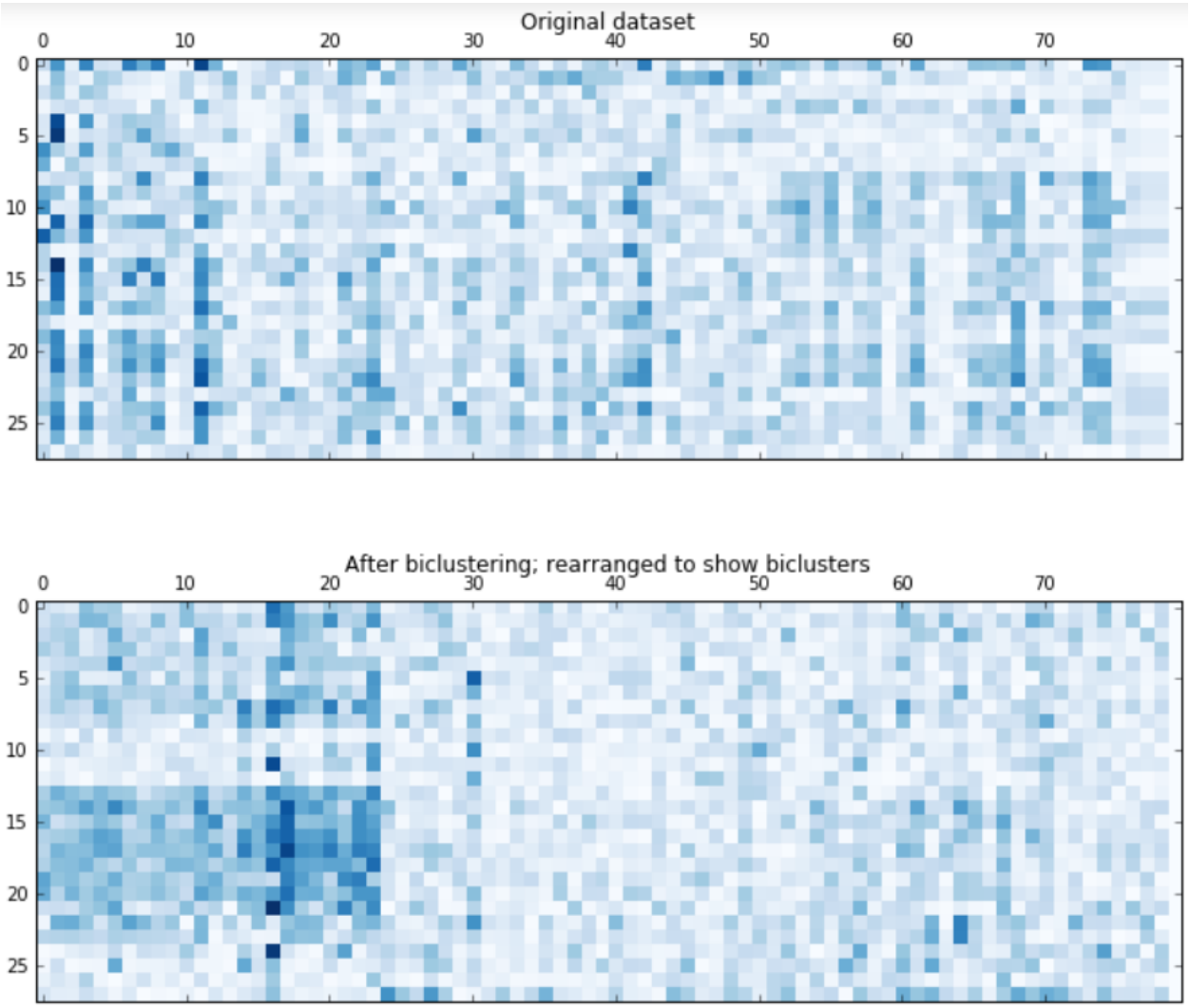


Figure 6-8.: Before and after biclustering, note the grouping in the darker blues at left.

- Halstead difficulty

In the cluster 4, 24 questions from MSLQ appear. The most frequent features are: peer learning and elaboration of ideas, both with four questions each. This is consistent with the results found in 2015-I were Halstead had a negative correlation with peer learning, and a positive correlation with elaboration of ideas.

In the cluster 5, 53 question from MSLQ appear. The mos frequent features are: time to study, task value, and organization of ideas. That is consistent with the results found in semester 2015-I, where task value and elaboration of ideas have a negative correlation with complexity metrics.

Another observation concerning the clusters 4 and 5 has to do with the appearance of Com-

Table 6-2.: Result of adding internal cluster values (top 3 values are highlighted in bold)

Cluster	Value
0	33.49
1	**38.90**
2	11.50
3	21.02
4	**50.29**
5	**41.76**
6	13.04
7	17.33
8	7.60
9	24.65

plexity metrics and Halstead difficulty. Such metrics appear in both clusters. As seen in the analysis of hierarchical bi-clusters, those metrics are negatively correlated to most learning strategies and with task value in the motivational features. This could support the importance of the mentioned metrics in the better use of learning strategies. In addition, results suggest that most self-regulated features are highly influenced by the metrics found in both clusters.

Moreover, in the cluster 1, which is the third in importance, the source code metrics found were:

- Average class lines
- Average amount of if clauses
- Average of wrong files
- Average of execution error files
- Average no-error files
- Amount of correct files
- Halstead effort
- Halstead time to understand/implement.

In the cluster 1, 54 questions from MSLQ appear. The most frequent features are: time to study, task value and anxiety. The source code metrics found are related to Length metrics (average class lines, amount of correct files) which are related with better performance. According to the results in Figure **6-3**, a strong correlation exists with at least 10 self-regulated

features. DomJudge metrics also appear in this clusters. It is interesting to see the relationship with questions related to task value (5 questions from 6 in the MSLQ). As seen before, task value has positive correlations with Length metrics and negative correlations with Complexity and close to zero with Halstead.

7. Predictive model

In this chapter, two predictive models will be presented. Such models are proposed based on the results found in the previous chapters, where the existence of groups of metrics and self-regulated features were found. Such predictive models apply machine learning algorithms to find out: if a student will approve the course based on the students' source code metrics and the answers of the MSLQ. It is worth to mention that the input values were normalized.

7.1. Classification model

For classification, a support vector machine (SVM) was used. The experiment was performed using a ten-fold cross validation, which consists in dividing the data set in ten subsets, nine of them are used to train the model and the remaining one is used to test. In the SVM, a RBF (Radial basis function) kernel was used to obtain the best results. The cross validation was applied to all the variations of parameters such as: penalty (C) and gamma (γ) within certain range. C ranges between 2^{-2} and 2^{50}, and gamma was from 2^{-4} to 2^{6}. For each one of these validations, the confusion matrix was calculated together with the error measures (i.e., accuracy, precision, and recall).

The target values were defined as approved (1) and not approved (0). As the amount of drop out students were too low, they were not considered. The features vector is comprised of the source code metrics and MSLQ data for each student.

The behavior of precision with different values of γ in the best C value during the cross validation is presented in the figure **7-1**. The horizontal axis is in a logarithmic base 2 scale. As can be observed, the best precision is $\gamma = 2^{-3} = 0{,}125$, which means a precision of 0,765. After this value, precision begins to decrease, but it precision stays close to 0,725.

Thus, the cross validation indicates that the best values were $C = 2^{1} = 2$ and $\gamma = 2^{-3} = 0{,}125$. Table **7-1** shows the confusion matrix obtained given the mentioned values. It can be seen that most of the students approved, and because of that, it was easier and more accurate to classify correctly a student who approve the course.

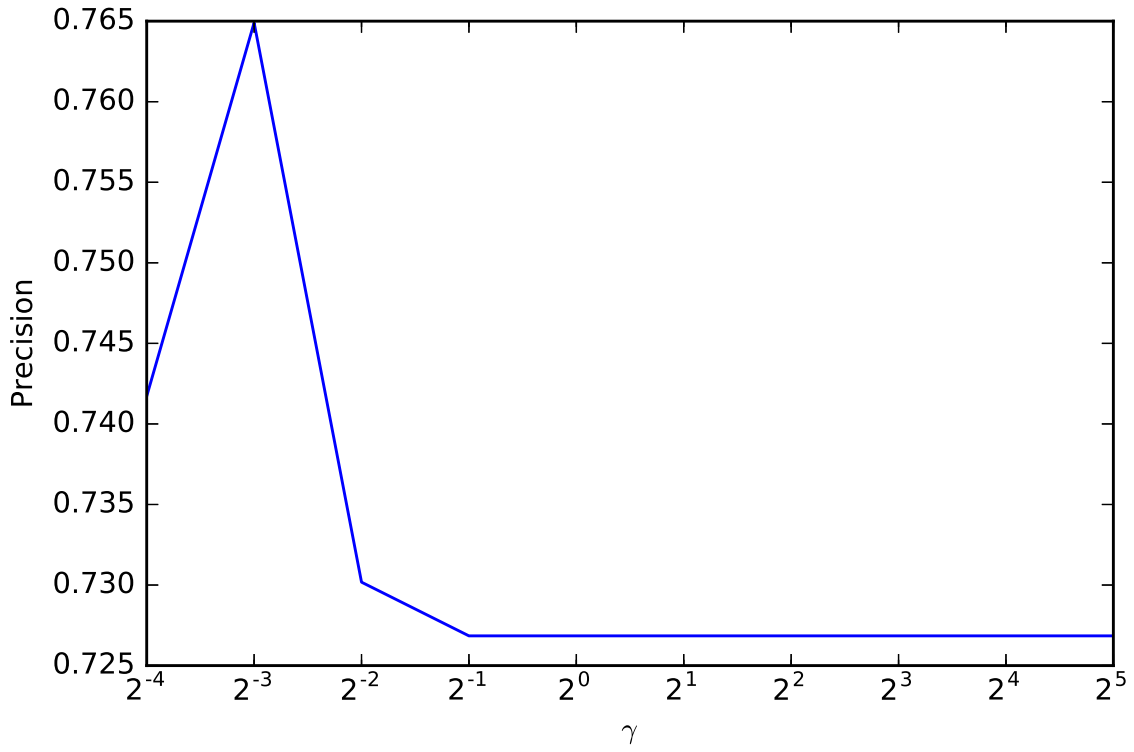


Figure 7-1.: Gamma value (γ) versus Precision in best C value during cross validation

Table 7-1.: Confusion matrix classification model

		Predicted class	
		Not Approved	**Approved**
Actual class	**Not approved**	13	28
	Approved	16	93

Based on the confusion matrix, three error measures were obtained. These are presented in Table **7-2**. The accuracy, which indicates the amount of correctly classified instances over the total amount, was 0,70. Moreover, the precision, which indicates how many elements are correctly classified over all the elements which were classified into a class, a value of 1,0 indicates that all elements were correctly classified. The resulting value in this for precision was 0,76. Finally, the recall is the sensitivity of the model. The value obtained in this case was 0,85.

Being the precision and recall close to each other, and at the same time, being close to 1,0; this indicates that the model seems to have obtained good performance in the cross validation. However, checking the details in the confusion matrix, most of the students who did

Table 7-2.: Error measures for classification model

Measure	**Value**
Accuracy	0,70
Precision	0,76
Recall	0,85

not approve were misclassified (28 students). To explain this, the approval rate has to be reviewed, in Table **4-1**, can be observed that from 172 students in the course during the three semesters, 71 % approved, and 29 % did not. In addition, in Section 4.2 it was mentioned that the practical component of the course represented the 50 % of the final grade, this means, that in this case the model only have half of the information required to predict the approval of a student. Still, the results of the confusion matrix suggest that the practical component is very important to approve the course, even if it is not determinant to get an approval grade.

7.2. Regression model

For regression, a support vector machine (SVM) was used. The experiment was done using a ten-fold cross validation, using a RFB kernel. In regression, it has an additional parameter ϵ, which specifies a distance where no penalty is given to a predicted point. In the cross validation, the parameters C and γ vary in the same way than for the classification model. The ϵ value varies in the range $(2^{-200}, 2^{-1})$.

The behavior of Mean Square Error (MSE) with different values of γ during the cross validation is presented in the figure **7-2**. It can be observed the best MSE is with $C = 2^{-2} = 4$, and from this value, it begins to increase up to $\gamma = 2$ and remains constant.

As result of the cross validation, the best parameters were $C = 2^{-2} = 0{,}25$, $\gamma = 2^{-2} = 0{,}25$ and $\epsilon = 2^{-10} = 0{,}00097$ with a MSE of 0,036184 and standard deviation 0,01, which, in terms of final grade in the course, represents a MSE of 0,18 and a standard deviation of 0,05.

According to the results, regression results are better at prediction than classification. The MSE and standard deviation values are enough to cross the approval boundary. There were 39 students with final grade between 2,8 and 3,2. This is a close to the 44 misclassified.

The results obtained in both models suggest that features used in the prediction models effectively have discriminant values. In addition, it supports the results obtained in Chapters 5 and 6, where high correlations were found between source code metric groups and

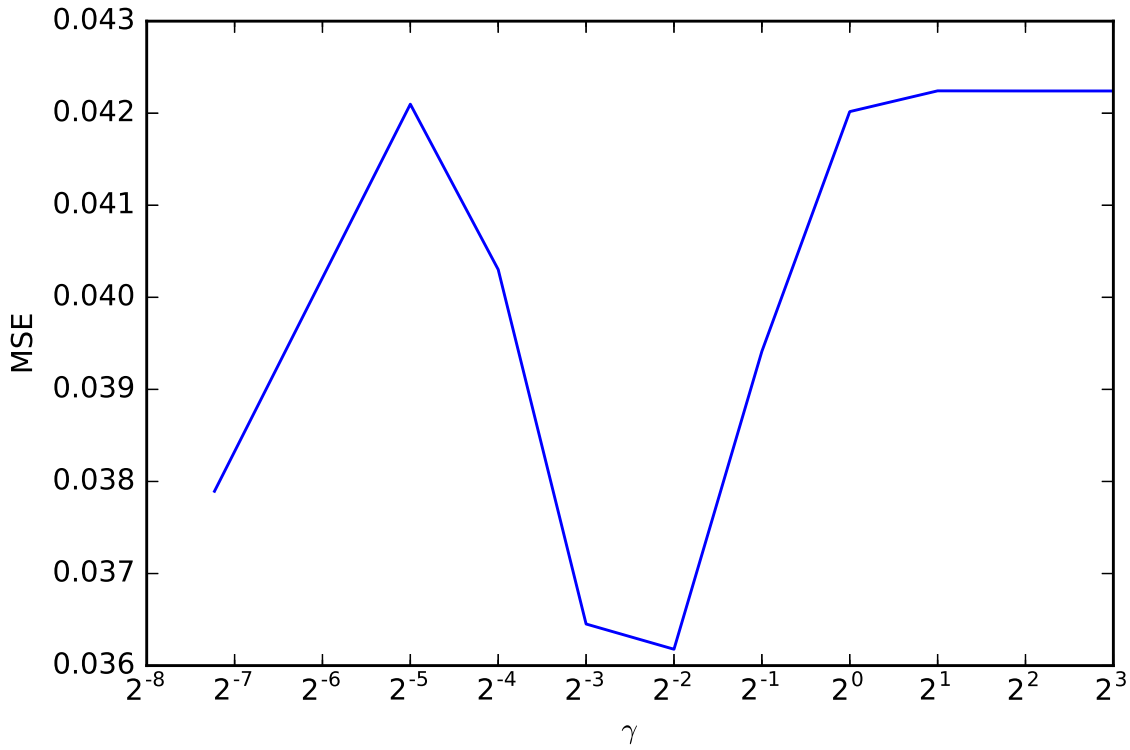


Figure 7-2.: γ values versus Mean Square Error (MSE) during cross validation in the best C value during cross validation

self-regulated features. The fact that the practical assignments only represent 50 % of the final grade could have affected the classification results. However, even with this impact, the results show that the source code metrics and the self-regulated features are useful features to predict student performance.

8. Conclusions and Future Work

8.1. General conclusions

Previous studies have focused in finding correlations between the self-regulated learning features and students performance in computer programming course. Although correlations were found in these studies, they provide few clues to give meaningful feedback to students on how to improve in their programming assignments in order to get better final grades. This thesis provides an initial evidence that source code metrics in computer programming courses not only have correlations with the students performance but also with their self-regulated learning characteristics. This suggests that source code metrics could be a source of information about the students motivation and their adequate use of learning strategies. However, further investigations are necessary to explain the cause-effect relationships of these correlations.

The general conclusions of this work can be summarized as follows:

1. There is evidence indicating that some source code metrics in a computer programming course are correlated with student performance, i.e., length-based metrics and Halstead.

2. Results suggest that source code metrics could be a source of information about students self-regulated characteristics, including some motivational features and the use of learning strategies.

3. The correlations found lead to identify groups of source code metrics which in turn have relationship with self-regulated characteristics and performance in the course.

4. Prediction through regression, obtain a low MSE. This indicates that applying regression over source code metrics and self-regulated features can make a close prediction to the final grade.

8.2. Contributions

In the light of the findings of this study, it is possible to understand better students source code as an artifact that can be used to monitor several characteristics related to self-regulated learning, course performance, and in general, their learning process. In this way, more research in the area is required to verify if these relationships could give to computing

educators new ways to identify and help those students with problems. In the future, the proposed strategy could be used as a base to build a tool which enables the teacher to give feedback to specific students early in the academic period.

The main contributions of this work can be summarized as follows:

1. A software tool for analyzing source code produced by students that supports the correlation analysis among coding style, students motivation, students use of learning strategies, and student performance in a computer programming course.

2. A better understanding of the students' learning process in computer programming subjects through the source code technical aspects, motivation and use of learning strategies.

3. The foundations to additional research on education in computer programming courses based on source code metrics and self-regulation characteristics.

4. The foundation to enable the teacher to give and receive feedback, allowing to focus on students with problems in the learning process by detecting possible problems in early stages of a computer programming course.

8.3. Publications

As a result of this thesis, the following papers have been published:

1. **Hugo Castellanos**. *Personality Recognition Applying Machine Learning Techniques on Source Code Metrics*. Working notes of FIRE 2016 - Forum for Information Retrieval Evaluation/CEUR Workshop proceedings. Kolkata, India. Dec 7-10 2016.

2. **Hugo Castellanos**, Felipe Restrepo-Calle, Fabio A. González, Jhon Jairo Ramírez Echeverry. *Understanding the relationships between self-regulated learning and students source code in a computer programming course*. Frontiers In Education 2017. Indianapolis, USA. October 18-21 2017 (accepted).

8.4. Future work

Several interesting correlations and groups were found when the clustering algorithms were applied. However, additional studies are needed to confirm the behavior of the self-regulated characteristics bases on the source code metrics extracted. Specially, those which indicates problems in the students' motivation and use of learning strategies.

As future work, the source code metrics considered in this work can be extended to other kinds. Due to the nature of the programming course studied in this paper, some metrics could not be applied. However, in more advanced courses, other metrics related to object-oriented programming could be considered, such as: coupling, cohesion, reuse, testing, etc. In addition, source code metrics related to readability will be considered as well.

Finally, more research is needed to verify the ways a teacher can identify students with problems based on the relationships found. Identifying students with problems may be not enough, additional research in automatic feedback based on the results of this thesis, could help all students. This could further improve the performance of the students in computer programming courses in general.

Bibliography

[Ala-Mutka et al., 2004] Ala-Mutka, K., Uimonen, T., and Jarvinen, H.-M. (2004). Supporting students in c++ programming courses with automatic program style assessment. *Journal of Information Technology Education*, 3(1):245–262.

[Alhazbi, 2014] Alhazbi, S. (2014). Using e-journaling to improve self-regulated learning in introductory computer programming course. In *Global Engineering Education Conference (EDUCON), 2014 IEEE*, pages 352–356. IEEE.

[Ambrosio et al., 2012] Ambrosio, A. P., Almeida, L., Franco, A., Martins, S., and Georges, F. (2012). Assessment of self-regulated attitudes and behaviors of introductory programming students. In *Frontiers in Education Conference (FIE), 2012*, pages 1–6. IEEE.

[Amelung et al., 2008] Amelung, M., Forbrig, P., and Rösner, D. (2008). Towards generic and flexible web services for e-assessment. In *ACM SIGCSE Bulletin*, volume 40, pages 219–224. ACM.

[Bakker, 2014] Bakker, T. (2014). *Plagiarism Detection in Source Code*. PhD thesis, Universiteit Leiden.

[Boekaerts et al., 2005] Boekaerts, M., Maes, S., and Karoly, P. (2005). Self-Regulation Across Domains of Applied Psychology: Is there an Emerging Consensus? *Applied Psychology*, 54(2):149–154.

[Brennan et al., 2012] Brennan, M., Afroz, S., and Greenstadt, R. (2012). Adversarial stylometry. *ACM Transactions on Information and System Security*, 15(3):1–22.

[Brusilovsky and Sosnovsky, 2005] Brusilovsky, P. and Sosnovsky, S. (2005). Individualized exercises for self-assessment of programming knowledge: An evaluation of quizpack. *Journal on Educational Resources in Computing (JERIC)*, 5(3):6.

[Burrows, 2010] Burrows, S. D. (2010). *Source Code Authorship Attribution*. PhD thesis, RMIT University.

[Caliskan-Islam et al., 2014] Caliskan-Islam, A., Harang, R., Liu, A., Narayanan, A., Voss, C., Yamaguchi, F., and Greenstadt, R. (2014). De-anonymizing Programmers via Code Stylometry.

[Cheang et al., 2003] Cheang, B., Kurnia, A., Lim, A., and Oon, W.-C. (2003). On automated grading of programming assignments in an academic institution. *Computers & Education*, 41(2):121–131.

[Cheng and Keung, 2011] Cheng, C. and Keung, E. (2011). The role of self-regulated learning in enhancing learning performance.

[de Haan and Schils, 1993] de Haan, P. and Schils, E. (1993). The qsum plot exposed. In *Proceedings of the 14th ICAME Conference*.

[De Lucia et al., 2011] De Lucia, A., Di Penta, M., and Oliveto, R. (2011). Improving source code lexicon via traceability and information retrieval. *IEEE Transactions on Software Engineering*, 37(2):205–227.

[DiFrancesca et al., 2016] DiFrancesca, D., Nietfeld, J. L., and Cao, L. (2016). A comparison of high and low achieving students on self-regulated learning variables. *Learning and Individual Differences*, 45:228–236.

[Elenbogen and Seliya, 2008] Elenbogen, B. S. and Seliya, N. (2008). Detecting Outsourced Student Programming Assignments. *Journal of Computing Sciences in Colleges*, 23(3):50–57.

[Frantzeskou et al., 2008] Frantzeskou, G., MacDonell, S., Stamatatos, E., and Gritzalis, S. (2008). Examining the significance of high-level programming features in source code author classification. *Journal of Systems and Software*, 81(3):447–460.

[Frantzeskou et al., 2007] Frantzeskou, G., Stamatatos, E., Gritzalis, S., Chaski, C. E., and Howald, B. S. (2007). Identifying Authorship by Byte-Level N-Grams: The Source Code Author Profile (SCAP) Method. *International Journal of Digital Evidence Spring*, 6(1).

[Frantzeskou et al., 2006] Frantzeskou, G., Stamatatos, E., Gritzalis, S., and Katsikas, S. (2006). Source code author identification based on N-gram author profiles. *IFIP International Federation for Information Processing*.

[Genkin and Lewis, 2005] Genkin, A. and Lewis, D. D. (2005). Author Identification on the Large Scale. *In Proc. of the Meeting of the Classification Society of North America*.

[Halstead, 1977] Halstead, M. H. (1977). *Elements of Software Science (Operating and Programming Systems Series)*. Elsevier Science Inc., New York, NY, USA.

[Hardcastle, 1993] Hardcastle, R. (1993). Forensic linguistics: An assessment of the cusum method for the determination of authorship. *Journal of the Forensic Science Society*, 33(2):95–106.

[Hayes and Offutt, 2010] Hayes, J. H. and Offutt, J. (2010). Recognizing authors: An examination of the consistent programmer hypothesis. *Software Testing Verification and Reliability*, 20(4):329–356.

[Holmes and Tweedie, 1995] Holmes, D. I. and Tweedie, F. J. (1995). Forensic Stylometry: A Review of the {CUSUM} Controversy. *Revue Informatique et Statistique dans les Science Humaines*, pages 19–47.

[Ihantola et al., 2010] Ihantola, P., Ahoniemi, T., Karavirta, V., and Seppälä, O. (2010). Review of recent systems for automatic assessment of programming assignments. In *Proceedings of the 10th Koli Calling International Conference on Computing Education Research*, pages 86–93. ACM.

[Joshi and Argiddi, 2013] Joshi, R. R. and Argiddi, R. V. (2013). Author Identification : An Approach Based on Style Feature Metrics of Software Source Codes. 4(4):564–568.

[Krsul and Spafford, 1995] Krsul, I. and Spafford, E. H. (1995). Authorship Analysis : Identifying The Author of a Program . 2 Statement of the Problem . 1 Introduction 4 Survey of Related Work. In *8th National Information Systems Security Conference*.

[Kumar Singh and Manimannan, 2013] Kumar Singh, A. and Manimannan, G. (2013). Literary Analysis using CUSUM Technique on Bharathiar Writings. *IOSR Journal of Mathematics*, 8(4):42–50.

[Kuric and Bieliková, 2014] Kuric, E. and Bieliková, M. (2014). Estimation of Student's Programming Expertise. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, number 1, page 35. ACM.

[Layton et al., 2013] Layton, R., Watters, P., and Dazeley, R. (2013). Local n-grams for author identification: Notebook for PAN at CLEF 2013. In *CEUR Workshop Proceedings*, volume 1179.

[Malhotra, 2015] Malhotra, R. (2015). *Empirical Research in Software Engineering: Concepts, Analysis, and Applications*. CRC Press.

[Malmi et al., 2005] Malmi, L., Karavirta, V., Korhonen, A., and Nikander, J. (2005). Experiences on automatically assessed algorithm simulation exercises with different resubmission policies. *Journal on Educational Resources in Computing (JERIC)*, 5(3):7.

[Manso-Vázquez and Llamas-Nistal, 2015] Manso-Vázquez, M. and Llamas-Nistal, M. (2015). A monitoring system to ease self-regulated learning processes. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 10(2):52–59.

[McCabe, 1976] McCabe, T. J. (1976). A complexity measure. *IEEE Transactions on software Engineering*, (4):308–320.

[McKeachie, 1986] McKeachie, W. J. (1986). Teaching and Learning in the College Classroom. A Review of the Research Literature (1986) and November 1987 Supplement. *National Center for Research to Improve Postseconotoxy Teaching and Learning*.

[Mckemmish, 1999] Mckemmish, R. (1999). What is Forensic Computing? *Australian Institute of Criminology.*, (118).

[Mosteller and Wallace, 1963] Mosteller, F. and Wallace, D. (1963). mosteller.pdf. *Journal of the American Statistical Association*, 58(302):275–309.

[Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.

[Nelson et al., 2015] Nelson, K. G., Shell, D. F., Husman, J., Fishman, E. J., and Soh, L.-K. (2015). Motivational and self-regulated learning profiles of students taking a foundational engineering course. *Journal of Engineering Education*, 104(1):74–100.

[OCS, 2013] OCS (2013). Science, technology, engineering and mathematics in the national interest: A strategic approach.

[Ortiz et al., 2015] Ortiz, O., Alcover, P. M., Sánchez, F., Pastor, J. Á., and Herrero, R. (2015). M-learning tools: The development of programming skills in engineering degrees. *IEEE Revista Iberoamericana de Tecnologias del Aprendizaje*, 10(3):86–91.

[Parr, 2013] Parr, T. (2013). *The definitive ANTLR 4 reference*. Pragmatic Bookshelf.

[Pears et al., 2007] Pears, A., Seidman, S., Malmi, L., Mannila, L., Adams, E., Bennedsen, J., Devlin, M., and Paterson, J. (2007). A survey of literature on the teaching of introductory programming. *SIGCSE Bulletin*, 39(4):204–223.

[Pieterse, 2013] Pieterse, V. (2013). Automated assessment of programming assignments. In *Proceedings of the 3rd Computer Science Education Research Conference on Computer Science Education Research*, pages 45–56. Open Universiteit, Heerlen.

[Pintrich, 1999] Pintrich, P. R. (1999). The role of motivation in promoting and sustaining self-regulated learning. *International Journal of Educational Research*, 31(6):459–470.

[Pintrich et al., 1991] Pintrich, P. R., Smith, D. A. F., Garcia, T., and McKeachie, W. J. (1991). A Manual for the Use of the Learning Questionnaire Motivated Strategies for (MSLQ). *Mediterranean Journal of Social Sciences*, 6(1):156–164.

[Radenski, 2008] Radenski, A. (2008). Digital cs1 study pack based on moodle and python. In *ACM SIGCSE Bulletin*, volume 40, pages 325–325. ACM.

[Ramírez-Echeverry et al., 2016] Ramírez-Echeverry, J. J., García-Carrillo, A., and Olarte Dussán, F. A. (2016). Adaptation and Validation of the Motivated Strategies for Learning Questionnaire -MSLQ- in Engineering Students in Colombia. *International Journal of Engineering Education*, 32-4.

[Ramírez Echeverry et al., 2014] Ramírez Echeverry, J. J., Olarte Dussan, F. A., and García Carrillo, A. (2014). Estrategias de aprendizaje usadas por estudiantes de ingeniería eléctrica e ingeniería electrónica de primer semestre. *Educación en ingeniería*, 9(18):216–227.

[Ramyaa et al., 2004] Ramyaa, C., Rasheed, K., and He, C. (2004). Using Machine Learning Techniques for Stylometry. *Conference on Machine Learning*, (Proceedings of International Conference on Machine Learning).

[Robins et al., 2003] Robins, A., Rountree, J., and Rountree, N. (2003). Learning and teaching programming: A review and discussion. *Computer Science Education*, 13(2):137–172.

[Rosenblum et al., 2011] Rosenblum, N., Zhu, X., and Miller, B. P. (2011). Who wrote this code? Identifying the authors of program binaries. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*.

[Rößling et al., 2008] Rößling, G., Joy, M., Moreno, A., Radenski, A., Malmi, L., Kerren, A., Naps, T., Ross, R. J., Clancy, M., Korhonen, A., et al. (2008). Enhancing learning management systems to better support computer science education. *ACM SIGCSE Bulletin*, 40(4):142–166.

[Rosson et al., 2011] Rosson, M. B., Carroll, J. M., and Sinha, H. (2011). Orientation of undergraduates toward careers in the computer and information sciences: Gender, self-efficacy and social support. *ACM Transactions on Computing Education (TOCE)*, 11(3):14.

[Rudman, 2012] Rudman, J. (2012). The State of Non-Traditional Authorship Attribution Studies - 2012: Some Problems and Solutions. *English Studies*, 93(3):259–274.

[Sahami and Roach, 2014] Sahami, M. and Roach, S. (2014). Computer science curricula 2013 released. *Communications of the ACM*, 57(6):5–5.

[Sauvé and Abath Neto, 2008] Sauvé, J. P. and Abath Neto, O. L. (2008). Teaching software development with atdd and easyaccept. *ACM SIGCSE Bulletin*, 40(1):542–546.

[Shen et al., 1983] Shen, V. Y., Conte, S. D., and Dunsmore, H. E. (1983). Software science revisited: A critical analysis of the theory and its empirical support. *IEEE Transactions on Software Engineering*, (2):155–165.

[Shevertalov et al., 2009] Shevertalov, M., Kothari, J., Stehle, E., and Mancoridis, S. (2009). On the use of discretized source code metrics for author identification. In *Proceedings - 1st International Symposium on Search Based Software Engineering, SSBSE 2009*.

[Singh et al., 2013] Singh, R., Gulwani, S., and Solar-Lezama, A. (2013). Automated feedback generation for introductory programming assignments. *ACM SIGPLAN Notices*, 48(6):15–26.

[Stamatatos, 2009] Stamatatos, E. (2009). A survey of modern authorship attribution methods. *Journal of the American Society for Information Science and Technology*, 60(3):538–556.

[Weinstein and Mayer, 1983] Weinstein, C. E. and Mayer, R. E. (1983). The teaching of learning strategies. In *Innovation abstracts*, volume 5, page n32. ERIC.

[Wisse, 2014] Wisse, W. (2014). *Authorship Identification and Verification of JavaScript Source Code*. PhD thesis, Delft University of technology.

[Yukselturk and Bulut, 2007] Yukselturk, E. and Bulut, S. (2007). Predictors for student success in an online course. *Educational Technology & Society*, 10(2):71–83.

[Yule, 1925] Yule, G. U. (1925). A Mathematical Theory of Evolution, Based on the Conclusions of Dr. J. C. Willis, F.R.S.

[Zimmerman, 1998] Zimmerman, B. J. (1998). Developing self-fulfilling cycles of academic regulation: An analysis of exemplary instructional models.

A. Correlation tables

In this appendix, the correlation tables mentioned in Chapter 5 are presented. Tables **A-1**, **A-2** and **A-3** correspond to the correlation coefficients of technical features (source code metrics). The correlation values are in the range [−1,0, 1,0], being 1,0 the maximum correlation value, 0 no correlation at all, and −1,0 the maximum inverse correlation. The values in bold are those which have a p-value $\leq$ 0,05, meaning that these correlations have statistical significance. The numbers on top and at the left of the tables are the same presented in Table **3-1** with the addition of the number 28 which means final grade (students performance).

Moreover, Tables **A-4**, **A-5**, and **A-6**, present the correlation coefficients between technical features and self-regulated features. In these tables the correlation values have the same range mentioned before, and the values in bold have the same meaning as in the previous tables. The numbers on top of the tables corresponds to technical features (presented in Table **3-1**) with the addition of the number 28 which means final grade. The numbers at the left of the tables corresponds to self-regulated features, and are the same presented in Tables **2-2** and **2-3**.

Table A-1.: Correlations among source code metrics in 2015-I

Metric	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	FG
0	**1.00**	**0.38**	0.15	**0.51**	**0.61**	**0.57**	0.03	-0.21	-0.02	-0.07	-0.01	-0.13	0.20	-0.07	-0.31	-0.23	0.19	0.21	0.09	**0.35**	0.21	0.22	**0.75**	**-1.00**	-0.02	0.05	0.05	-0.20	**0.57**
1	**0.38**	**1.00**	**0.79**	**0.84**	**0.56**	**0.78**	**0.70**	**-0.72**	-0.22	**-0.38**	-0.08	**-0.62**	0.23	-0.16	**-0.68**	0.09	0.06	0.23	0.05	-0.13	-0.12	**0.56**	**0.44**	**-0.42**	0.18	0.23	0.23	0.12	**0.57**
2	0.15	**0.79**	**1.00**	**0.49**	**0.35**	**0.53**	**0.47**	**-0.65**	-0.19	**-0.42**	-0.34	**-0.65**	-0.10	-0.27	**-0.76**	-0.21	-0.19	0.19	0.17	-0.02	-0.20	**0.35**	0.21	-0.15	**0.48**	**0.51**	**0.51**	**0.41**	**0.36**
3	**0.51**	**0.84**	**0.49**	**1.00**	**0.64**	**0.83**	**0.68**	**-0.68**	-0.26	**-0.37**	-0.06	**-0.55**	**0.46**	-0.31	**-0.59**	0.11	0.14	0.10	0.05	-0.05	-0.09	**0.64**	**0.50**	**-0.49**	-0.00	0.04	0.04	-0.09	**0.56**
4	**0.61**	**0.56**	**0.35**	**0.64**	**1.00**	**0.66**	0.30	-0.33	-0.17	**-0.40**	-0.16	**-0.42**	0.31	-0.05	**-0.43**	-0.19	-0.14	0.11	0.27	0.31	-0.06	0.25	**0.54**	**-0.61**	0.09	0.14	0.14	-0.05	**0.59**
5	**0.57**	**0.78**	**0.53**	**0.83**	**0.66**	**1.00**	**0.58**	**-0.69**	-0.14	-0.34	-0.07	**-0.55**	**0.36**	-0.14	**-0.61**	0.16	0.13	0.04	0.12	-0.00	-0.03	**0.56**	**0.59**	**-0.54**	0.12	0.18	0.18	0.04	**0.53**
6	0.03	**0.70**	**0.47**	**0.68**	0.30	**0.58**	**1.00**	**-0.64**	-0.32	**-0.47**	**-0.36**	**-0.66**	0.24	**-0.43**	**-0.65**	0.29	-0.10	-0.12	0.13	-0.29	-0.26	**0.71**	0.24	-0.06	0.19	0.20	0.20	0.20	0.16
7	-0.21	**-0.72**	**-0.65**	**-0.68**	-0.33	**-0.69**	**-0.64**	**1.00**	0.14	**0.58**	-0.05	**0.72**	-0.15	0.07	**0.64**	-0.26	-0.17	-0.09	0.13	-0.04	0.17	**-0.56**	-0.25	0.21	-0.04	-0.07	-0.07	-0.04	-0.28
8	-0.02	-0.22	-0.19	-0.26	-0.17	-0.14	-0.32	0.14	**1.00**	**0.48**	0.21	**0.52**	0.01	**0.40**	0.14	-0.03	-0.11	-0.21	0.24	-0.14	-0.08	-0.16	0.05	-0.01	-0.07	-0.06	-0.06	-0.05	-0.06
9	-0.07	**-0.38**	**-0.42**	**-0.37**	**-0.40**	-0.34	**-0.47**	**0.58**	**0.48**	**1.00**	0.19	**0.87**	-0.03	0.18	**0.43**	0.08	0.03	-0.11	0.05	-0.26	0.26	-0.13	-0.03	0.07	-0.05	-0.03	-0.03	-0.03	-0.10
10	-0.01	-0.08	-0.34	-0.06	-0.16	-0.07	**-0.36**	-0.05	0.21	0.19	**1.00**	**0.43**	0.02	**0.46**	0.26	0.14	0.12	-0.00	-0.23	0.11	0.20	0.01	-0.02	0.01	**-0.55**	**-0.51**	**-0.51**	**-0.47**	-0.09
11	-0.13	**-0.62**	**-0.65**	**-0.55**	**-0.42**	**-0.55**	**-0.66**	**0.72**	**0.52**	**0.87**	**0.43**	**1.00**	-0.06	**1.00**	**-0.36**	0.00	-0.01	-0.14	0.02	-0.15	0.21	**-0.36**	-0.08	0.14	-0.25	-0.24	-0.24	-0.21	-0.24
12	0.20	0.23	-0.10	**0.46**	0.31	**0.36**	0.24	-0.15	0.01	-0.03	0.02	-0.06	**1.00**	**-0.36**	-0.06	-0.01	0.33	0.09	-0.01	-0.05	0.00	0.33	0.16	-0.20	-0.21	-0.20	-0.20	-0.27	0.15
13	-0.07	-0.16	-0.27	-0.31	-0.05	-0.14	**-0.43**	0.07	**0.40**	0.18	**0.46**	0.33	**-0.36**	**1.00**	**0.45**	0.05	0.05	-0.02	-0.08	0.18	0.08	**-0.47**	-0.17	0.07	-0.23	-0.23	-0.23	-0.16	-0.07
14	-0.31	**-0.68**	**-0.76**	**-0.59**	**-0.43**	**-0.61**	**-0.65**	**0.64**	0.14	**0.43**	0.26	**0.66**	-0.06	**0.45**	**1.00**	-0.02	0.14	-0.12	-0.15	0.06	0.13	**-0.65**	**-0.40**	**0.35**	**-0.34**	**-0.38**	**-0.38**	-0.25	**-0.38**
15	-0.23	0.09	-0.21	0.11	-0.19	0.16	0.29	-0.26	-0.03	0.08	0.14	0.00	-0.01	0.05	-0.02	**1.00**	-0.05	-0.33	-0.17	**-0.35**	-0.16	0.16	**0.35**	**0.36**	0.05	0.04	0.04	0.20	0.07
16	0.19	0.06	-0.19	0.14	-0.14	0.13	-0.10	-0.17	-0.11	0.03	0.12	-0.01	0.33	0.05	0.14	-0.05	**1.00**	**0.37**	**-0.67**	-0.11	-0.07	-0.01	-0.06	-0.01	**-0.36**	**-0.36**	**-0.36**	-0.32	0.00
17	0.21	0.23	0.19	0.10	0.11	0.04	-0.12	-0.09	-0.21	-0.11	-0.00	-0.14	0.09	-0.02	-0.12	-0.33	**0.37**	**1.00**	**-0.46**	0.15	0.04	-0.23	-0.01	-0.21	-0.04	-0.04	-0.04	-0.11	0.13
18	0.09	0.05	0.17	0.05	0.27	0.12	0.13	0.13	0.24	0.05	-0.23	0.02	-0.01	-0.08	-0.15	-0.17	**-0.67**	**-0.46**	**1.00**	-0.08	-0.06	0.13	0.04	-0.09	0.27	0.29	0.29	0.24	0.02
19	**0.35**	-0.13	-0.02	-0.05	0.31	-0.00	-0.29	-0.04	-0.14	-0.26	0.11	-0.15	-0.05	0.18	0.06	**-0.35**	-0.11	0.15	-0.08	**1.00**	0.28	-0.13	0.04	**-0.52**	-0.17	-0.18	-0.18	-0.34	0.08
20	0.21	-0.12	-0.20	-0.09	-0.06	-0.03	-0.26	0.17	-0.08	0.26	0.20	0.21	0.00	0.08	0.13	-0.16	-0.07	0.04	-0.06	0.28	**1.00**	-0.06	0.10	-0.21	0.05	0.08	0.08	0.03	0.06
21	0.22	**0.56**	**0.35**	**0.64**	0.25	**0.56**	**0.71**	**-0.56**	-0.16	-0.13	0.01	**-0.36**	0.33	**-0.47**	**-0.65**	0.16	-0.01	-0.23	0.13	-0.13	-0.06	**1.00**	**0.38**	-0.28	0.09	0.13	0.13	-0.01	0.25
22	**0.75**	**0.44**	0.21	**0.50**	**0.54**	**0.59**	0.24	-0.25	0.05	-0.03	-0.02	-0.08	0.16	-0.17	**-0.40**	**0.35**	-0.06	-0.01	0.04	0.04	0.10	**0.38**	**1.00**	**-0.70**	0.19	0.25	0.25	0.07	**0.76**
23	**-1.00**	**-0.42**	-0.15	**-0.49**	**-0.61**	**-0.54**	-0.06	0.21	-0.01	0.07	0.01	0.14	-0.20	0.07	**0.35**	**0.36**	-0.01	-0.21	-0.09	**-0.52**	-0.21	-0.28	**-0.70**	**1.00**	0.02	-0.05	-0.05	0.20	**-0.57**
24	-0.02	0.18	**0.48**	-0.00	0.09	0.12	0.19	-0.04	-0.07	-0.05	**-0.55**	-0.25	-0.21	-0.23	**-0.34**	0.05	**-0.36**	-0.04	0.27	-0.17	0.05	0.09	0.19	0.02	**1.00**	**0.99**	**0.99**	**0.95**	0.33
25	0.05	0.23	**0.51**	0.04	0.14	0.18	0.20	-0.07	-0.06	-0.03	**-0.51**	-0.24	-0.20	-0.23	**-0.38**	0.04	**-0.36**	-0.04	0.29	-0.18	0.08	0.13	0.25	-0.05	**0.99**	**1.00**	**1.00**	**0.95**	**0.39**
26	0.05	0.23	**0.51**	0.04	0.14	0.18	0.20	-0.07	-0.06	-0.03	**-0.51**	-0.24	-0.20	-0.23	**-0.38**	0.04	**-0.36**	-0.04	0.29	-0.18	0.08	0.13	0.25	-0.05	**0.99**	**1.00**	**1.00**	**0.95**	**0.39**
27	-0.20	0.12	**0.41**	-0.09	-0.05	0.04	0.20	-0.04	-0.05	-0.03	**-0.47**	-0.21	-0.27	-0.16	-0.25	0.20	-0.32	-0.11	0.24	-0.34	0.03	-0.01	0.07	0.20	**0.95**	**0.95**	**0.95**	**1.00**	0.23
28	**0.57**	**0.57**	**0.36**	**0.56**	**0.59**	**0.53**	0.16	-0.28	-0.06	-0.10	-0.09	-0.24	0.15	-0.07	**-0.38**	0.07	0.00	0.13	0.02	0.08	0.06	0.25	**0.76**	**-0.57**	0.33	**0.39**	**0.39**	0.23	**1.00**

Table A-2.: Correlations among source code metrics in 2015-II

Metric	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	FG
0	**1.00**	0.24	0.14	-0.04	0.19	-0.05	-0.21	0.06	-0.05	**0.37**	-0.05	0.19	0.01	-0.17	-0.05	**-0.47**	**0.34**	0.12	0.07	0.22	0.19	-0.10	0.14	**-1.00**	0.03	0.06	0.06	-0.09	-0.03
1	0.24	**1.00**	**0.75**	**0.66**	**0.47**	**0.70**	0.03	-0.26	-0.31	-0.19	**-0.63**	**-0.48**	0.04	-0.07	**-0.52**	-0.26	**0.38**	-0.00	-0.08	-0.04	0.23	0.21	-0.21	-0.24	0.31	0.30	0.30	0.27	0.08
2	0.14	**0.75**	**1.00**	0.29	**0.69**	**0.48**	0.02	**-0.51**	-0.02	**-0.37**	**-0.55**	**-0.62**	0.26	0.08	**-0.49**	-0.01	0.06	0.00	0.02	-0.06	0.14	0.26	-0.03	-0.14	**0.55**	**0.55**	**0.55**	**0.50**	0.19
3	-0.04	**0.66**	0.29	**1.00**	-0.01	**0.58**	0.23	-0.04	**-0.45**	-0.04	-0.31	-0.25	0.01	-0.20	**-0.43**	-0.04	**0.35**	0.09	-0.20	-0.14	0.13	0.31	-0.02	0.04	0.06	0.05	0.05	0.06	0.16
4	0.19	**0.47**	**0.69**	-0.01	**1.00**	0.32	0.07	**-0.36**	0.10	**-0.37**	-0.31	**-0.49**	**0.34**	0.12	**-0.35**	-0.13	0.07	-0.06	0.17	0.07	0.01	-0.02	-0.07	-0.19	0.30	0.31	0.31	0.27	0.01
5	-0.05	**0.70**	**0.48**	**0.58**	0.32	**1.00**	0.29	-0.26	**-0.36**	**-0.47**	**-0.36**	**-0.60**	-0.03	-0.27	**-0.73**	-0.23	0.26	-0.15	0.17	0.05	0.17	0.18	-0.29	0.05	0.22	0.14	0.14	0.22	0.01
6	-0.21	0.03	0.02	0.23	0.07	0.29	**1.00**	-0.27	-0.30	-0.30	0.01	-0.31	0.31	**-0.39**	-0.29	-0.03	-0.20	-0.08	0.10	-0.03	0.23	**0.52**	-0.08	0.21	-0.15	-0.17	-0.17	-0.10	0.06
7	0.06	-0.26	**-0.51**	-0.04	**-0.36**	-0.26	-0.27	**1.00**	-0.18	**0.34**	0.08	**0.62**	0.11	-0.09	**0.52**	-0.19	-0.08	0.13	-0.19	0.25	0.15	-0.30	-0.14	-0.06	-0.12	-0.11	-0.11	-0.11	-0.09
8	-0.05	-0.31	-0.02	**-0.45**	0.10	**-0.36**	-0.30	-0.18	**1.00**	**0.35**	**0.37**	**0.41**	-0.23	**0.47**	**0.42**	0.02	-0.24	**0.37**	-0.04	0.04	-0.20	-0.08	0.05	0.05	-0.02	-0.02	-0.02	-0.00	-0.25
9	**0.37**	-0.19	**-0.37**	-0.04	**-0.37**	**-0.47**	-0.30	**0.34**	**0.35**	**1.00**	**0.33**	**0.82**	**-0.33**	0.12	**0.43**	-0.26	0.10	**0.57**	**-0.34**	0.20	0.17	-0.18	0.04	**-0.37**	-0.31	-0.30	-0.30	-0.31	-0.30
10	-0.05	**-0.63**	**-0.55**	-0.31	-0.31	**-0.36**	0.01	0.08	**0.37**	**0.33**	**1.00**	**0.61**	-0.09	0.23	**0.50**	0.26	-0.15	0.09	0.16	-0.25	-0.18	0.06	**0.34**	0.05	**-0.41**	**-0.41**	**-0.41**	**-0.39**	-0.03
11	0.19	**-0.48**	**-0.62**	-0.25	**-0.49**	**-0.60**	-0.31	**0.62**	**0.41**	**0.82**	**0.61**	**1.00**	-0.15	0.20	**0.71**	-0.11	-0.11	**0.41**	-0.18	0.15	0.03	-0.17	0.05	-0.19	**-0.35**	**-0.37**	**-0.37**	**-0.36**	-0.21
12	0.01	0.04	0.26	0.01	**0.34**	-0.03	0.31	0.11	-0.23	**-0.33**	-0.09	-0.15	**1.00**	-0.28	0.10	0.04	-0.28	-0.03	0.17	-0.03	0.09	0.25	-0.01	-0.01	**0.39**	**0.39**	**0.39**	**0.41**	**0.32**
13	-0.17	-0.07	0.08	-0.20	0.12	-0.27	**-0.39**	-0.09	**0.47**	0.12	0.23	0.20	-0.28	**1.00**	**0.38**	0.19	-0.07	0.20	-0.05	-0.26	-0.05	-0.15	0.10	0.17	0.13	0.08	0.08	0.17	-0.19
14	-0.05	**-0.52**	**-0.49**	**-0.43**	**-0.35**	**-0.73**	-0.29	**0.52**	**0.42**	**0.43**	**0.50**	**0.71**	0.10	**0.38**	**1.00**	0.22	**-0.33**	0.32	-0.21	-0.04	-0.08	-0.03	0.13	0.05	-0.22	-0.18	-0.18	-0.21	-0.04
15	**-0.47**	-0.26	-0.01	-0.04	-0.13	-0.23	-0.03	-0.19	0.02	-0.26	0.26	-0.11	0.04	0.19	0.22	**1.00**	**-0.43**	-0.22	-0.10	**-0.69**	-0.31	**0.34**	**0.73**	**0.47**	-0.00	-0.01	-0.01	-0.01	**0.57**
16	**0.34**	**0.38**	0.06	**0.35**	0.07	0.26	-0.20	-0.08	-0.24	0.10	-0.15	-0.11	-0.28	-0.07	**-0.33**	**-0.43**	**1.00**	-0.03	-0.31	-0.11	0.18	-0.13	-0.16	**-0.34**	-0.12	-0.06	-0.06	-0.11	-0.13
17	0.12	-0.00	0.00	0.09	-0.06	-0.15	-0.08	0.13	**0.37**	**0.57**	0.09	**0.41**	-0.03	0.20	0.32	-0.22	-0.03	**1.00**	**-0.40**	0.06	0.09	0.17	-0.21	-0.12	0.27	0.26	0.26	0.24	0.09
18	0.07	-0.08	0.02	-0.20	0.17	0.17	0.10	-0.19	-0.04	**-0.34**	0.16	-0.18	0.17	-0.05	-0.21	-0.10	-0.31	**-0.40**	**1.00**	-0.07	-0.22	-0.13	-0.05	-0.07	0.01	-0.03	-0.03	-0.03	-0.05
19	0.22	-0.04	-0.06	-0.14	0.07	0.05	-0.03	0.25	0.04	0.20	-0.25	0.15	-0.03	-0.26	-0.04	**-0.69**	-0.11	0.06	-0.07	**1.00**	0.22	**-0.41**	**-0.65**	-0.22	-0.05	-0.00	-0.00	-0.03	**-0.57**
20	0.19	0.23	0.14	0.13	0.01	0.17	0.23	0.15	-0.20	0.17	-0.18	0.03	0.09	-0.05	-0.08	-0.31	0.18	0.09	-0.22	0.22	**1.00**	0.14	-0.17	-0.19	0.02	0.02	0.02	0.00	-0.01
21	-0.10	0.21	0.26	0.31	-0.02	0.18	**0.52**	-0.30	-0.08	-0.18	0.06	-0.17	0.25	-0.15	-0.03	**0.34**	-0.13	0.17	-0.13	**-0.41**	0.14	**1.00**	0.25	0.10	0.16	0.16	0.16	0.17	**0.50**
22	0.14	-0.21	-0.03	-0.02	-0.07	-0.29	-0.08	-0.14	0.05	0.04	**0.34**	0.05	-0.01	0.10	0.13	**0.73**	-0.16	-0.21	-0.05	**-0.65**	-0.17	0.25	**1.00**	-0.14	-0.01	-0.18	-0.18	-0.05	**0.64**
23	**-1.00**	-0.24	-0.14	0.04	-0.19	0.05	0.21	-0.06	0.05	**-0.37**	0.05	-0.19	-0.01	0.17	0.05	**0.47**	**-0.34**	-0.12	-0.07	-0.22	-0.19	0.10	-0.14	**1.00**	-0.03	-0.06	-0.06	0.09	0.03
24	0.03	0.31	**0.55**	0.06	0.30	0.22	-0.15	-0.12	-0.02	-0.31	**-0.41**	**-0.35**	**0.39**	0.13	-0.22	-0.00	-0.12	0.27	0.01	-0.05	0.02	0.16	-0.01	-0.03	**1.00**	**1.00**	**1.00**	**0.99**	0.22
25	0.06	0.30	**0.55**	0.05	0.31	0.14	-0.17	-0.11	-0.02	-0.30	**-0.41**	**-0.37**	**0.39**	0.08	-0.18	-0.01	-0.06	0.26	-0.03	-0.00	0.02	0.16	-0.18	-0.06	**1.00**	**1.00**	**1.00**	**0.98**	0.21
26	0.06	0.30	**0.55**	0.05	0.31	0.14	-0.17	-0.11	-0.02	-0.30	**-0.41**	**-0.37**	**0.39**	0.08	-0.18	-0.01	-0.06	0.26	-0.03	-0.00	0.02	0.16	-0.18	-0.06	**1.00**	**1.00**	**1.00**	**0.98**	0.21
27	-0.09	0.27	**0.50**	0.06	0.27	0.22	-0.10	-0.11	-0.00	-0.31	**-0.39**	**-0.36**	**0.41**	0.17	-0.21	-0.01	-0.11	0.24	-0.03	-0.03	0.00	0.17	-0.05	0.09	**0.99**	**0.98**	**0.98**	**1.00**	0.20
FG	-0.03	0.08	0.19	0.16	0.01	0.01	0.06	-0.09	-0.25	-0.30	-0.03	-0.21	**0.32**	-0.19	-0.04	**0.57**	-0.13	0.09	-0.05	**-0.57**	-0.01	**0.50**	**0.64**	0.03	0.22	0.21	0.21	0.20	**1.00**

Table A-3.: Correlations among source code metrics in 2016-I

Metric	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	FG
0	1.00	0.67	0.60	0.62	0.71	0.65	0.43	-0.53	-0.30	-0.37	-0.22	-0.52	0.16	-0.27	-0.55	-0.06	0.40	0.52	0.03	-0.12	0.09	0.54	0.75	-1.00	-0.48	-0.25	-0.25	-0.57	0.50
1	0.67	1.00	0.89	0.89	0.81	0.92	0.56	-0.79	-0.56	-0.57	-0.47	-0.76	-0.06	-0.57	-0.87	0.18	0.09	0.44	0.05	-0.14	0.08	0.50	0.70	-0.67	-0.04	0.17	0.17	-0.12	0.41
2	0.60	0.89	1.00	0.69	0.80	0.82	0.47	-0.86	-0.58	-0.68	-0.53	-0.85	-0.14	-0.59	-0.90	0.28	0.09	0.31	-0.00	-0.15	0.13	0.40	0.71	-0.60	0.04	0.24	0.24	-0.04	0.43
3	0.62	0.89	0.69	1.00	0.67	0.86	0.57	-0.63	-0.45	-0.40	-0.39	-0.61	0.02	-0.48	-0.71	0.14	0.11	0.41	0.00	-0.10	0.11	0.54	0.60	-0.62	-0.13	0.05	0.05	-0.19	0.40
4	0.71	0.81	0.80	0.67	1.00	0.75	0.39	-0.70	-0.47	-0.53	-0.37	-0.70	0.22	-0.48	-0.72	0.15	0.16	0.43	0.06	-0.19	0.22	0.42	0.68	-0.71	-0.18	0.01	0.01	-0.26	0.50
5	0.65	0.92	0.82	0.86	0.75	1.00	0.52	-0.83	-0.61	-0.69	-0.51	-0.85	-0.04	-0.54	-0.83	0.15	0.10	0.45	0.09	-0.16	0.08	0.53	0.67	-0.65	-0.08	0.12	0.12	-0.15	0.44
6	0.43	0.56	0.47	0.57	0.39	0.52	1.00	-0.39	-0.46	-0.40	-0.30	-0.46	-0.01	-0.33	-0.47	0.06	0.13	0.26	-0.04	-0.17	0.06	0.83	0.38	-0.43	-0.07	0.04	0.04	-0.13	0.23
7	-0.53	-0.79	-0.86	-0.63	-0.70	-0.83	-0.39	1.00	0.55	0.75	0.45	0.90	0.16	0.50	0.82	-0.15	-0.08	-0.34	-0.09	0.15	-0.08	-0.34	-0.59	0.53	-0.07	-0.23	-0.23	0.00	-0.39
8	-0.30	-0.56	-0.58	-0.45	-0.47	-0.61	-0.46	0.55	1.00	0.68	0.61	0.71	0.27	0.41	0.51	-0.09	-0.23	-0.05	0.13	0.27	-0.05	-0.36	-0.36	0.30	-0.19	-0.27	-0.27	-0.13	-0.11
9	-0.37	-0.57	-0.68	-0.40	-0.53	-0.69	-0.40	0.75	0.68	1.00	0.54	0.91	0.22	0.47	0.66	-0.01	-0.14	-0.17	-0.01	0.24	-0.04	-0.41	-0.39	0.37	-0.18	-0.31	-0.31	-0.10	-0.14
10	-0.22	-0.47	-0.53	-0.39	-0.37	-0.51	-0.30	0.45	0.61	0.54	1.00	0.66	0.17	0.46	0.52	-0.23	-0.03	-0.04	-0.01	0.24	-0.01	-0.25	-0.29	0.22	-0.19	-0.29	-0.29	-0.14	-0.04
11	-0.52	-0.76	-0.85	-0.61	-0.70	-0.85	-0.46	0.90	0.71	0.91	0.66	1.00	0.14	0.56	0.82	-0.13	-0.13	-0.26	-0.16	0.19	-0.09	-0.43	-0.56	0.52	-0.08	-0.25	-0.25	0.00	-0.28
12	0.16	-0.06	-0.14	0.02	0.22	-0.04	0.16	-0.01	0.27	0.22	0.17	0.14	1.00	0.14	0.04	-0.12	0.03	0.03	-0.02	0.25	0.15	0.06	0.14	-0.16	-0.42	-0.40	-0.40	-0.41	0.25
13	-0.27	-0.57	-0.59	-0.48	-0.48	-0.54	-0.33	0.50	0.41	0.47	0.46	0.56	0.14	1.00	0.71	-0.06	-0.02	0.02	0.09	-0.03	-0.10	-0.18	-0.29	0.27	-0.07	-0.15	-0.15	-0.05	-0.10
14	-0.55	-0.87	-0.90	-0.71	-0.72	-0.83	-0.47	0.82	0.51	0.66	0.52	0.82	0.12	0.71	1.00	-0.24	-0.01	-0.21	-0.08	0.08	-0.07	-0.39	-0.64	0.55	-0.05	-0.24	-0.24	0.02	-0.38
15	-0.06	0.18	0.28	0.14	0.15	0.15	0.06	-0.15	-0.09	-0.01	-0.23	-0.13	0.04	-0.06	-0.24	1.00	-0.27	-0.00	-0.21	-0.35	0.03	-0.05	0.49	0.06	0.17	0.20	0.20	0.17	0.38
16	0.40	0.09	0.09	0.11	0.16	0.10	0.13	-0.08	-0.23	-0.14	-0.03	-0.13	-0.12	-0.02	-0.01	-0.27	1.00	0.04	-0.31	-0.34	0.15	0.12	0.19	-0.40	-0.17	-0.13	-0.13	-0.23	0.03
17	0.52	0.44	0.31	0.41	0.43	0.45	0.26	-0.34	-0.05	-0.17	-0.04	-0.26	0.03	0.02	-0.21	-0.00	0.04	1.00	0.03	-0.16	-0.02	0.43	0.40	-0.52	-0.23	-0.10	-0.10	-0.28	0.46
18	0.03	0.05	-0.00	0.00	0.06	0.09	-0.04	-0.09	0.13	-0.25	-0.01	-0.16	-0.02	0.09	-0.08	-0.21	-0.31	0.03	1.00	-0.07	-0.21	0.05	-0.05	-0.03	-0.04	0.00	0.00	-0.07	-0.10
19	-0.12	-0.14	-0.15	-0.10	-0.19	-0.16	-0.17	0.15	0.27	0.20	0.24	0.19	0.25	-0.03	0.08	-0.35	-0.34	-0.16	-0.07	1.00	-0.04	-0.08	-0.34	0.12	-0.11	-0.12	-0.12	-0.07	-0.20
20	0.09	0.08	0.13	0.11	0.22	0.08	0.06	-0.08	-0.05	-0.04	-0.01	-0.09	0.15	-0.10	-0.07	0.03	0.15	-0.02	-0.21	-0.04	1.00	-0.00	0.06	-0.09	-0.13	-0.12	-0.12	-0.15	0.05
21	0.54	0.50	0.40	0.54	0.42	0.53	0.83	-0.34	-0.36	-0.41	-0.25	-0.43	0.06	-0.18	-0.39	-0.05	0.12	0.43	0.05	-0.08	-0.00	1.00	0.37	-0.54	-0.20	-0.06	-0.06	-0.26	0.29
22	0.75	0.70	0.71	0.60	0.68	0.67	0.38	-0.59	-0.36	-0.39	-0.29	-0.56	0.14	-0.29	-0.64	0.49	0.19	0.40	-0.05	-0.34	0.06	0.37	1.00	-0.75	-0.26	-0.06	-0.06	-0.34	0.65
23	-1.00	-0.67	-0.60	-0.62	-0.71	-0.65	-0.43	0.53	0.30	0.37	0.22	0.52	-0.16	0.27	0.55	0.06	-0.40	-0.52	-0.03	0.12	-0.09	-0.54	-0.75	1.00	0.48	0.25	0.25	0.57	-0.50
24	-0.48	-0.04	0.04	-0.13	-0.18	-0.08	-0.07	-0.07	-0.19	-0.18	-0.19	-0.08	-0.42	-0.07	-0.05	0.17	-0.17	-0.23	-0.04	-0.11	-0.13	-0.20	-0.26	0.48	1.00	0.96	0.96	0.98	-0.31
25	-0.25	0.17	0.24	0.05	0.01	0.12	0.04	-0.23	-0.27	-0.31	-0.29	-0.25	-0.40	-0.15	-0.24	0.20	-0.13	-0.10	0.00	-0.12	-0.12	-0.06	-0.06	0.25	0.96	1.00	1.00	0.91	-0.20
26	-0.25	0.17	0.24	0.05	0.01	0.12	0.04	-0.23	-0.27	-0.31	-0.29	-0.25	-0.40	-0.15	-0.24	0.20	-0.13	-0.10	0.00	-0.12	-0.12	-0.06	-0.06	0.25	0.96	1.00	1.00	0.91	-0.20
27	-0.57	-0.12	-0.04	-0.19	-0.26	-0.15	-0.13	0.00	-0.13	-0.10	-0.14	0.00	-0.41	-0.05	0.02	0.17	-0.23	-0.28	-0.07	-0.07	-0.15	-0.26	-0.34	0.57	0.98	0.91	0.91	1.00	-0.35
FG	0.50	0.41	0.43	0.40	0.50	0.44	0.23	-0.39	-0.11	-0.14	-0.04	-0.28	0.25	-0.10	-0.38	0.38	0.03	0.46	-0.10	-0.20	0.05	0.29	0.65	-0.50	-0.31	-0.20	-0.20	-0.35	1.00

Table A-4.: Correlations between technical features and MSLQ features on 2015-I

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
0	0.29	-0.13	0.04	0.02	0.08	**0.44**	0.24	0.21	0.22	0.04	0.09	0.02	0.07	-0.23	0.04	0.17	-0.10	0.15
1	0.15	0.08	0.05	**0.37**	0.26	**0.35**	0.08	0.30	-0.09	0.00	0.27	**0.37**	0.04	-0.20	0.26	**0.52**	0.25	0.23
2	0.10	0.09	0.10	0.18	0.27	0.19	-0.05	0.08	-0.29	-0.06	0.09	0.24	-0.01	-0.23	0.12	0.33	0.25	-0.01
3	0.21	-0.02	-0.13	0.32	0.20	0.33	0.21	0.32	0.16	0.11	**0.35**	0.25	-0.02	-0.29	0.16	**0.52**	0.22	0.26
4	0.34	-0.17	-0.10	0.03	0.12	**0.46**	0.24	0.10	-0.11	-0.06	0.24	0.12	0.02	-0.19	0.05	0.33	0.07	0.03
5	0.28	0.08	0.04	**0.41**	0.22	**0.36**	0.27	**0.36**	0.02	0.10	**0.46**	**0.37**	0.11	-0.23	0.06	**0.46**	0.19	**0.41**
6	0.24	0.05	-0.08	**0.37**	0.14	0.28	0.28	**0.36**	-0.12	0.17	**0.45**	**0.39**	0.01	-0.07	0.13	**0.39**	0.32	0.31
7	-0.27	-0.13	0.07	-0.26	-0.12	-0.33	-0.23	**-0.42**	-0.01	-0.24	**-0.37**	**-0.40**	-0.19	0.09	-0.22	**-0.46**	**-0.38**	-0.32
8	-0.09	-0.05	-0.10	-0.08	-0.31	-0.15	0.01	-0.13	0.05	0.04	-0.07	0.01	0.23	-0.01	-0.06	-0.04	0.02	0.11
9	-0.21	0.06	0.18	0.03	0.01	-0.34	-0.17	0.00	0.18	-0.01	-0.17	-0.21	0.03	0.07	-0.01	-0.10	-0.08	-0.06
10	-0.15	0.15	-0.02	0.17	-0.09	-0.25	-0.11	-0.05	0.26	0.02	-0.01	-0.01	0.21	-0.05	0.15	0.01	-0.07	0.19
11	-0.33	-0.06	0.03	-0.11	-0.12	**-0.43**	-0.24	-0.20	0.22	-0.04	-0.30	-0.33	0.00	0.00	-0.03	-0.26	-0.24	-0.17
12	-0.05	-0.13	-0.13	-0.07	-0.04	0.04	0.10	0.17	0.23	0.16	0.01	0.25	0.16	0.09	-0.11	0.20	0.11	0.24
13	0.06	0.09	-0.01	-0.03	0.18	0.14	0.21	-0.04	0.21	0.08	0.02	-0.13	**0.39**	0.14	0.25	0.07	0.21	0.09
14	-0.26	0.09	-0.05	-0.25	-0.09	-0.26	-0.19	-0.23	0.29	0.06	**-0.42**	-0.30	0.07	0.28	0.04	-0.34	-0.03	-0.19
15	0.00	0.12	0.09	0.32	0.18	0.05	0.19	**0.41**	-0.08	0.26	0.27	**0.38**	0.16	0.13	0.14	0.25	0.30	0.24
16	-0.07	-0.26	-0.03	-0.00	0.24	0.14	0.05	0.20	0.16	0.14	-0.15	0.08	0.13	-0.07	0.12	0.23	-0.11	-0.06
17	0.13	-0.11	0.00	-0.24	0.27	0.18	-0.05	0.16	0.13	-0.04	0.07	0.18	0.13	0.03	0.22	0.32	-0.05	-0.13
18	-0.05	-0.20	-0.11	0.11	-0.10	-0.12	-0.14	-0.25	0.04	0.12	0.07	-0.17	-0.21	-0.12	-0.11	-0.11	0.07	-0.00
19	0.02	0.20	-0.15	**-0.39**	-0.09	0.04	-0.10	-0.33	0.02	-0.26	-0.16	**-0.40**	-0.08	-0.00	-0.14	-0.10	**-0.36**	-0.03
20	-0.08	0.22	0.03	-0.17	0.03	-0.11	-0.10	0.13	0.13	0.05	0.01	-0.17	0.00	-0.17	-0.17	**-0.38**	0.05	-0.09
21	-0.06	-0.06	0.03	0.24	0.07	0.02	0.08	0.23	-0.11	0.11	0.32	0.08	-0.05	-0.19	0.01	**0.43**	0.09	**0.37**
22	0.22	-0.04	0.18	0.23	0.14	**0.40**	0.31	**0.36**	-0.01	0.11	0.31	0.21	0.09	-0.14	0.12	0.27	0.08	0.32
23	-0.29	0.11	-0.04	-0.02	0.00	**-0.44**	-0.24	-0.12	-0.19	0.02	-0.09	-0.00	-0.07	0.23	0.01	-0.08	0.04	-0.07
24	0.06	-0.13	0.10	-0.15	0.15	0.02	-0.01	0.19	**-0.42**	-0.02	0.07	0.21	0.05	-0.15	-0.13	-0.15	0.20	-0.25
25	0.12	-0.12	0.14	-0.09	0.17	0.05	0.01	0.23	**-0.41**	0.00	0.09	0.23	0.06	-0.17	-0.11	-0.13	0.22	-0.25
26	0.12	-0.12	0.14	-0.09	0.17	0.05	0.01	0.23	**-0.41**	0.00	0.09	0.23	0.06	-0.17	-0.11	-0.13	0.22	-0.25
27	0.01	-0.13	0.09	-0.04	0.17	0.00	-0.01	0.26	**-0.41**	0.01	0.06	0.29	0.06	-0.18	-0.10	-0.21	0.25	-0.30

Table A-5.: Correlations between technical features and MSLQ features on 2015-II

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
0	0.25	0.24	0.25	0.22	0.06	**0.38**	**0.47**	0.06	0.05	0.02	-0.12	0.16	0.19	**0.33**	0.06	0.12	0.05	0.03
1	-0.10	0.04	-0.01	-0.19	0.04	0.16	0.13	0.28	0.14	-0.04	-0.17	0.08	0.00	0.13	0.13	0.16	0.09	0.16
2	-0.03	0.13	0.20	0.09	0.13	0.18	0.16	0.14	0.07	0.07	-0.09	0.09	0.05	0.10	0.15	0.10	0.05	0.31
3	-0.09	**-0.36**	-0.31	**-0.41**	-0.16	-0.09	-0.03	0.22	0.07	-0.11	-0.29	-0.22	-0.19	0.03	-0.03	0.05	-0.22	-0.14
4	-0.18	0.31	0.11	0.02	0.10	0.07	0.09	0.02	-0.06	0.07	-0.13	0.17	-0.14	0.08	-0.10	-0.09	-0.04	0.14
5	-0.25	-0.16	**-0.40**	**-0.39**	-0.19	-0.17	-0.27	0.03	-0.14	-0.18	0.01	-0.18	-0.27	-0.11	-0.12	0.03	0.03	-0.01
6	-0.30	-0.32	-0.17	-0.27	0.05	-0.24	-0.20	-0.00	-0.12	-0.23	-0.07	-0.17	-0.24	0.06	-0.04	-0.05	-0.15	-0.12
7	0.10	-0.20	-0.06	-0.17	-0.01	-0.11	0.05	-0.21	0.21	0.05	-0.12	0.01	0.06	0.06	-0.01	0.10	0.03	-0.14
8	-0.00	**0.35**	0.11	0.22	-0.00	0.15	-0.00	0.03	0.07	-0.01	0.14	0.16	0.01	0.11	0.12	-0.21	0.11	0.18
9	0.29	0.25	0.15	0.27	0.24	0.22	0.29	0.12	0.30	0.08	0.10	0.24	0.19	0.17	0.15	-0.08	0.19	-0.07
10	0.05	0.00	-0.19	0.06	-0.14	0.00	-0.07	-0.17	-0.24	-0.05	-0.08	-0.13	-0.04	-0.10	-0.12	**-0.32**	-0.14	-0.19
11	0.18	0.10	0.07	0.10	0.02	0.06	0.14	-0.11	0.14	-0.01	0.00	0.07	0.14	0.09	0.05	-0.11	0.11	-0.13
12	-0.06	-0.25	0.01	-0.22	-0.02	-0.06	-0.00	-0.06	0.04	-0.00	**-0.34**	-0.12	-0.10	0.17	-0.03	-0.05	-0.21	0.15
13	-0.05	**0.36**	0.15	0.27	-0.18	0.16	0.01	0.08	0.13	0.11	-0.04	-0.00	-0.02	-0.00	0.01	-0.14	-0.04	0.16
14	0.18	0.08	0.22	0.14	-0.01	0.07	0.18	-0.04	0.23	0.01	-0.10	0.06	0.22	0.10	0.06	-0.05	0.08	-0.04
15	0.02	-0.23	-0.01	0.02	-0.04	-0.04	-0.08	-0.00	-0.18	-0.03	-0.04	-0.16	0.01	-0.03	0.08	-0.08	-0.21	-0.01
16	-0.08	0.02	**-0.33**	-0.25	-0.26	-0.01	-0.05	0.02	-0.11	-0.13	**-0.35**	-0.14	-0.11	-0.13	-0.13	-0.01	-0.24	-0.29
17	**0.34**	0.09	0.09	0.26	0.17	0.30	0.31	**0.33**	**0.56**	0.30	0.10	0.31	0.24	**0.34**	0.24	-0.05	0.26	0.21
18	-0.25	0.24	0.04	0.08	-0.11	0.10	-0.05	-0.26	-0.31	-0.08	0.19	-0.03	-0.16	0.02	-0.26	-0.14	0.07	0.09
19	0.10	0.09	0.21	0.06	0.26	-0.09	0.02	-0.05	0.08	0.03	0.25	0.22	0.04	-0.03	0.02	0.16	0.30	0.11
20	-0.13	-0.06	-0.20	-0.20	0.00	-0.07	-0.06	-0.07	-0.02	-0.05	0.03	0.01	-0.12	-0.05	0.07	0.04	0.03	-0.17
21	-0.14	-0.30	-0.07	-0.23	0.04	-0.01	0.03	0.15	-0.10	-0.18	-0.24	-0.10	0.13	0.25	0.30	0.08	-0.05	0.07
22	0.16	-0.03	0.13	0.18	0.03	0.23	0.19	0.04	-0.20	-0.07	-0.12	-0.07	0.12	0.21	0.15	-0.04	-0.12	-0.04
23	-0.25	-0.24	-0.25	-0.22	-0.06	**-0.38**	**-0.47**	-0.06	-0.05	-0.02	0.12	-0.16	-0.19	**-0.33**	-0.06	-0.12	-0.05	-0.03
24	0.15	-0.05	0.10	0.15	-0.15	0.17	0.19	0.19	0.27	0.30	-0.07	0.16	0.11	0.19	0.16	0.19	0.06	**0.40**
25	0.13	-0.12	0.13	0.09	-0.15	0.12	0.11	0.18	0.26	0.23	-0.09	0.10	0.15	0.15	0.17	0.15	-0.01	**0.38**
26	0.13	-0.12	0.13	0.09	-0.15	0.12	0.11	0.18	0.26	0.23	-0.09	0.10	0.15	0.15	0.17	0.15	-0.01	**0.38**
27	0.13	-0.06	0.10	0.14	-0.14	0.15	0.15	0.19	0.27	0.29	-0.08	0.14	0.08	0.16	0.15	0.18	0.03	**0.39**

Table A-6.: Correlations between technical features and MSLQ features on 2016-I

	0	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**13**	**14**	**15**	**16**	**17**
0	-0.01	-0.15	**-0.28**	0.09	0.09	0.13	0.03	0.17	0.03	0.18	0.12	**0.26**	0.11	0.18	**0.36**	0.14	0.17	0.12
1	-0.08	0.02	**-0.31**	0.04	0.01	-0.08	-0.11	0.13	-0.02	**0.22**	0.07	0.16	0.09	0.12	**0.29**	0.01	0.02	0.16
2	-0.09	0.07	-0.17	0.04	0.02	-0.11	-0.13	0.17	-0.00	**0.25**	0.01	0.17	0.09	0.11	**0.25**	0.08	0.02	0.14
3	-0.12	0.06	**-0.33**	-0.05	-0.01	-0.10	-0.13	0.07	-0.03	0.11	0.04	0.05	0.07	0.15	0.19	-0.07	-0.02	0.13
4	-0.02	-0.01	-0.15	0.07	0.07	-0.02	-0.01	0.13	-0.05	0.17	0.02	**0.23**	0.05	0.10	**0.24**	0.03	-0.03	0.16
5	-0.06	0.05	**-0.25**	0.01	0.05	-0.04	-0.07	0.09	-0.01	0.21	0.10	0.14	0.09	0.14	**0.33**	-0.02	0.05	0.14
6	0.06	0.01	-0.12	0.18	0.15	0.11	0.03	0.03	0.06	0.03	0.01	0.08	0.15	0.18	**0.27**	-0.08	-0.02	0.05
7	-0.04	0.01	0.07	-0.06	-0.13	-0.05	0.02	-0.17	0.04	**-0.29**	-0.11	**-0.22**	-0.18	-0.13	**-0.34**	-0.04	-0.12	-0.12
8	0.16	0.04	0.10	-0.00	0.00	0.02	0.05	-0.03	-0.10	-0.13	0.04	-0.11	-0.19	-0.07	**-0.23**	0.14	0.10	-0.06
9	0.01	-0.04	0.08	-0.12	-0.09	-0.02	0.00	-0.09	-0.12	**-0.24**	-0.15	**-0.22**	-0.18	-0.19	**-0.38**	-0.01	-0.15	-0.18
10	0.21	-0.11	0.06	0.08	0.08	0.10	0.03	-0.03	-0.12	-0.11	0.04	-0.02	-0.00	-0.14	-0.07	0.05	-0.07	-0.21
11	0.07	-0.05	0.12	-0.04	-0.08	0.00	0.03	-0.12	-0.09	**-0.25**	-0.13	-0.20	-0.12	-0.16	**-0.34**	-0.02	-0.11	-0.21
12	0.10	0.09	0.18	0.01	0.10	0.14	0.13	0.12	-0.01	-0.07	0.02	0.01	-0.13	0.08	-0.07	**0.22**	0.02	-0.03
13	-0.01	-0.12	0.19	-0.09	0.04	0.15	0.09	-0.10	-0.07	-0.13	-0.06	-0.11	-0.19	-0.14	-0.13	-0.02	0.03	**-0.23**
14	0.02	-0.06	**0.24**	-0.08	-0.01	0.13	0.08	-0.15	-0.03	**-0.28**	-0.08	-0.16	-0.09	-0.11	**-0.27**	-0.03	-0.05	**-0.23**
15	-0.15	0.17	0.03	-0.10	-0.19	**-0.38**	**-0.29**	-0.07	**0.24**	0.00	-0.19	-0.16	-0.20	-0.05	-0.17	-0.02	-0.16	-0.15
16	-0.07	-0.21	-0.04	-0.03	0.10	**0.22**	0.03	0.17	0.09	0.11	0.06	**0.26**	**0.33**	0.21	**0.33**	-0.01	0.10	**0.22**
17	0.03	-0.01	-0.06	0.07	0.11	0.18	0.00	-0.10	-0.10	0.00	0.06	0.11	-0.02	0.03	**0.24**	-0.06	0.02	-0.12
18	0.06	0.07	-0.16	0.19	0.08	-0.00	0.11	-0.07	0.06	0.04	0.19	0.10	-0.06	-0.10	0.12	0.01	0.16	0.04
19	0.10	0.05	-0.03	-0.13	-0.01	0.04	0.16	0.08	**-0.26**	-0.02	0.09	-0.04	-0.08	0.00	-0.21	0.19	-0.04	-0.07
20	-0.07	0.01	0.14	-0.20	0.12	0.03	-0.07	-0.03	-0.04	-0.14	-0.02	-0.07	-0.12	0.04	0.03	-0.15	-0.07	0.12
21	0.05	0.04	-0.14	0.16	0.09	0.15	0.09	0.00	-0.02	0.04	0.13	0.12	0.13	**0.25**	**0.28**	-0.05	0.14	0.05
22	-0.17	-0.02	-0.19	-0.02	-0.03	-0.09	-0.14	0.16	0.20	0.15	-0.09	0.18	-0.00	0.13	**0.22**	0.05	-0.06	0.06
23	0.01	0.15	**0.28**	-0.09	-0.09	-0.13	-0.03	-0.17	-0.03	-0.18	-0.12	**-0.26**	-0.11	-0.18	**-0.36**	-0.14	-0.17	-0.12
24	-0.04	0.08	0.06	0.08	-0.19	-0.13	-0.07	-0.02	-0.02	0.12	-0.06	0.00	0.14	-0.04	-0.20	-0.07	-0.05	0.05
25	-0.05	0.08	-0.03	0.11	-0.19	-0.13	-0.07	0.03	0.01	0.17	-0.02	0.07	0.16	0.02	-0.11	-0.05	-0.02	0.08
26	-0.05	0.08	-0.03	0.11	-0.19	-0.13	-0.07	0.03	0.01	0.17	-0.02	0.07	0.16	0.02	-0.11	-0.05	-0.02	0.08
27	-0.01	0.07	0.10	0.08	-0.19	-0.12	-0.07	-0.02	-0.03	0.10	-0.08	-0.02	0.14	-0.06	**-0.23**	-0.07	-0.06	0.03