COLMENA: Collaborative knowledge and user classification environment based on programming experience

Carlos Fernández-Medina¹, Juan Ramón Pérez-Pérez¹, Mª Puerto Paule-Ruiz¹, Victor Manuel Álvarez-García¹

¹PULSO Research Team. Department of Computer Science, University of Oviedo
33007 Oviedo, Spain
Carlosfernandezmedina@gmail.com, {jrpp, paule, victoralvarez}@uniovi.es

Abstract. Nowadays, software developers use their acquired formation, expertise and an integrated development environment (IDE) as the only resources for improving the development of applications. This paper proposes a system called COLMENA which aims to assist software developers during their learning process. COLMENA is based on information retrieval over errors and warnings made by developers (in our academic context, students) and compares this information with the rest of the group’s in order to obtain a user’s programming level. Moreover, the system provides students with feedback about/on their most common errors and warnings.

Keywords: user classification, integrated development environment (IDE), code analysis, e-learning.

1 Introduction

Traditionally, courses have been developed in specific learning platforms, where students can study theory and also solve exercises. However, in these platforms it is difficult to integrate the most practical parts of the course.

Traditional learning environments, based on theory and classroom exercises, do not provide good results in all situations [7]. This is the case of subjects which are fundamentally based on experimentation at different levels, such as the learning of programming languages.

In Computer Science, and more particularly in programming, it is accepted that the basic way to learn is making programs and testing them in real environments and not only studying the sentences and structure of the programming language [8]. For this purpose, students use tools in the development of software the same way professionals do. However their knowledge is still limited and many times the use of these tools represents an obstacle for learning.

It is usual that in their first lessons, students fail to master fluently the different concepts related to programming language usage (reserved words, program structure, visibility of variables, etc.). Besides, the tools might present a problem when they lack intuitiveness, making hard developing the first applications. This situation can lead to
frustration and demotivation, students can encounter obstacles to their learning progress and even consider quitting the course.

Other times, more experienced students, who are accustomed to the programming environment used in practical classes, make mistakes that showing they still have to acquire certain concepts.

In this type of situations, it would be useful to have a system that analyses what the developer’s mistakes are and, according to the level of the user, suggest different approaches to solve the problem.

COLMENA integrates in development environments features such as qualitative and quantitative data about which warnings or errors are most common, indicators of students’ progress over time and comments about the quality of code written. Thus, the system provides learning programming support for beginner and advanced students. Our proposal is based on combining students’ personal monitoring with learning by experimentation, which is made possible by the development environment itself.

This article is structured as follows: section 2 provides a state of the art, regarding user classification as well as current tools used to obtain information from the IDE. Our proposal is introduced and described in detail in section 3. Finally, our conclusions and future work lines are presented.

2 Related Work

The approach used in related work provides classification models based principally in user actions in specific systems, such as Control Version System (CVS). Through retrieval information techniques (e.g. Data Mining) these applications obtain a classification of users attending to expertise.

Some classification models and some systems that implement these models in order to obtain a ranking of experts are presented in section 2.1 and 2.2.

2.1 Classification of users based on expertise

Generally, obtaining a model of user classification is made in two different ways:

**Heuristics:** within this class there are systems like Expertise Recommender (ER) [11], which recommends experts based on two different heuristics:

- **Change History:** Based mainly in the “line 10 rule” [1]. When a developer has a problem with a part or section of the source code; this heuristic proposes checking the control version system (CVS) in order to select the last editors of the file as experts.
- **Tech Support:** Based on generating a collaborative knowledge base with problems and ways to solve them that developers have had previously.

**Empirical measures:** Mockus and Herselb [13] have developed a tool called Expertise Browser, which uses the concept “Experience Atoms” (EA). These EA are
the minimal expression of experience and they are assigned to the user depending on the task where the experience was generated.

Focused on integrated development environments, Anvik and Murphy [1] offer an empiric evaluation for determining “implementation expertise” from bug reports, obtained from the source code and errors repository. The result is a set of experts obtained from the tool that will be compared with results obtained by a real expert team. The approaches for obtaining this set of experts are the following:

- Examining source repository commit logs in modules related with the bug report.
- Use data from bug reports in the bug network.

Minto and Murphy [12] propose Emergent Expertise Locator (EEL), a recommender system for teams of experts about a set of files, promoting collaboration between the current developer and this team of experts. For this purpose, EEL uses cooperative tools like chat or e-mail.

Schuler and Zimmerman [16] propose a new heuristic based on analysis of CVS. They present a new concept called “usage expertise”. This recommendations system focuses on the calls and changes in methods, assuming that the developer who made more changes and made more calls to a method will know more about this, because otherwise they would not call it and even less modify it.

2.2 Integrated tools

In addition to expertise selection models, it is interesting to highlight the essential characteristics that programming environments must have for being reusable, extensible and automatable. Following the aforementioned approach, Zeller [17] proposes three characteristics to be considered:

1. **Integration**: the integrated development environments must be each day more modular, extensible and interoperable. Eclipse [5] is a good example of this tendency, being a framework composed by different extensions, also called “plug-ins”.
2. **Synergy**: Zeller explains that in software production, we must not analyse only the source code generated by developers, but also analyze the activities of developers through change history, logs, email files or bug databases.
3. **Assistance**: In this last point, the advantages of assistance to some programming activities were remarked. An example of this assistance is code visualization, through techniques like data mining.

Examples of tools that have the abovementioned characteristics are DDCHANGE [3], EROSE [18] and SUADE [15]. These tools are usually developed as a plug-in for Eclipse, although they could be Java applets or another type of application.

DDCHANGE provides information about the different causes of a bug or error in a program. In order to achieve this function, the tool is based on several modules such
as change history, automatic testing and the reconstruction and static analysis of code. EROSE is based in techniques of data mining, offering suggestions of methods, classes or documentation to the developer. These suggestions are based on other developers’ previous experience.

In a similar way, SUADE plug-in recommends potential elements for a given context like labels and methods. For this task, the tool analyses structural dependences of the program between internal elements.

3 Our Proposal: COLMENA

The objective of COLMENA is to orient the user in the development of software. To do this, it firstly classifies users according to their programming level, assigning a custom profile that allows adaptation to a development environment. The system consists of the following parts (Fig. 1):

- **Information retrieval**: The result of analysing the user’s code is collected.
- **User Classification**: It contrasts information about one developer against information about the whole group is contrasted in order to assign a certain level to the user.

**Adaptation of interface**: According to a developer’s level, different parts of the interface will be enabled or disabled. Moreover, it will show customised and individual information about the developer (Fig. 3).

![Fig. 1. Classification and adaptation model.](image)

3.1 Data capture

Nowadays development environments inform developers about errors in the source code and during execution, but they do not capture nor manage this information.
Developers use this information in real time to fix errors and information is discarded afterwards.

In our proposal, we suggest storing and using the aforementioned information for later analyzing and providing feedback on it. The gathering will take place during what we call "programming session" of the user. This period includes the time since the user is validated in the system until the session is concluded. These sessions will last, as a rule, a lab class or a day's work.

The main data that will be captured are related to the software development process of the user:

- **Compilation errors**: This type of errors indicates that the source code of the program is wrong and it is not possible to execute it.
- **Execution errors**: The program is executable but there are errors in its operations.
- **Warnings**: Characteristics of the code structure that could be symptoms of low quality. Although development environments already provide this information, other external tools such as PMD (http://pmd.sourceforge.net/), FINDBUG (http://findbugs.sourceforge.net/index.html) or JPLAG [9] can be used for completing and improving this process.

All this information is stored in a database for later access. Besides errors and warnings, other parameters are taken into account, such as the date and time when errors happen, the code line of the error or the file that contains them, so COLMENA can perform different types of exhaustive analysis.

### 3.2 Classification of user inside a group

In a similar way as in other systems [14], individually collected data are combined in a common database to perform an analysis of all developers in order to obtain global information about the group.

Resulting from our work as teachers in programming courses we can conclude that within the group of students, most of them have a number of errors and warnings near to the average, a minority with a much above than average and fewer significantly lower than average. This distribution of students in the group suggests, statistically, that they form a normal distribution.

To classify the users into subgroups based on the mistakes they make, we divide the distribution taking into account the global standard deviation (σ). This way, we create thresholds that differentiate each level. The chosen values for splitting the normal distribution are the value of standard deviation (± σ) and twice the same (± 2σ). In this division, six segments appear: each segment is a group, apart from the central interval comprising two segments, which is considered the same group because it is the closest to the average. This initial theoretical approach allows us to model results obtained from programs submitted by students of computer science at the University of Oviedo (Fig. 2).
3.3 Adaptation of interface and indicators

From the obtained data and classification, we consider several important parameters when designing the interface to facilitate the use of the software development environment, and thus the learning of programming:

1. **Indicators.** The user interface has some indicators, which inform user about their level respect the group, statistic of errors, his learning progress and qualitative information about his source code (Fig. 3).
   - **Relative level of user:** Current programming levels are shown from users’ classification, which considers compilation and execution errors as well as warnings.
   - **Error statistics:** It displays a pie chart on the absolute data of different types of errors made by the programmer.
   - **Progress chart:** The evolution over time of the number of user errors is shown as a line graph, changing colour dynamically according to the trend of the errors [6].
   - **Qualitative information on the code:** Textual information is displayed about typical errors associated with the level of the programmer.

2. **List of frequent errors.** This list shows the error messages and warnings which are most common to the users so that they can work on them to avoid these errors in the future.

3. **Adaptation of options in development environment.** Using the methods and techniques of adaptive hypermedia [2] it is possible to adapt the user interface. Depending on the level of the user and using the technique of hiding links, it is possible to enable or disable menu items in the IDE. Thus, for example, a novice user only sees menu options that they will use, hence simplifying the handling of the IDE.
Conclusion and future work

Traditionally, the programming learning presents the problem that the theoretical knowledge is not sufficient, making necessary to use an IDE to obtain certain skills. However, the lack of feedback and assistance in programming makes students feel confused with the programming language and the available number of tools to learn it.

COLMENA suggests solving this problem by storing individual errors and combining them with errors made by other developers. Then the system classifies developers into five different levels depending on the number of mistakes they make. In addition to this classification, there are indicators that provide the user with information about their most frequent errors and warnings. In this way, the system is capable of providing feedback on students’ most significant difficulties.

Technologies themselves do not directly cause learning to occur, but can afford certain tasks that themselves may result in learning or give rise to certain learning benefits [4]. In this way, COLMENA attempts to integrate into development environments the necessary characteristics to provide students with effective learning.

As future work, we propose a dynamic system of values that gives more weight to errors considering their impact and their complexity. In this way it could evaluate the
influence of various errors and warnings at the time the classification of developers is being performed.

It is also interesting to add to the tool a new teacher-oriented approach, allowing analysing the distribution of the group. Through this analysis in the different stages of learning, teachers could get to know relevant facts such as whether a group is improving in certain aspects, or if, conversely, learning is not proving very fruitful.

Other minor extensions for the system could include making assistance available for students at different stages and levels of software development, as it is proposed in systems such as COALA [7], or integrating the list of common mistakes into a knowledge base which contains information about their solution.

At the time of writing this paper, an experimental version of COLMENA is already available for use. The current version stores data locally and do not provide a graphical user interface. The COLMENA prototype is downloadable from URL (http://www.di.uniovi.es/~juanrp/colmena/).

References

15. Robillard, M., with Frédéric Weigand-Warr: SUADE Plugin for Eclipse, 
16. Schuler, D., Zimmermann, T. Mining usage expertise from version archives. Proceedings of 
   the 2008 international workshop on Mining software repositories - MSR ’08. 121 (2008).
   316-325 (2007).
18. Zimmermann, T. et al. Mining version histories to guide software changes. IEEE 