# Pockets: A Tool to Support Exploratory Programming for Novices and Educators

Erina Makihara
Nara Institute of Science and Technology, Japan
makihara.erina.lx0@is.naist.jp

## ABSTRACT

Exploratory programming is one of the programming techniques, and it is considered to be an effective way to improve programming skills for novices. However, there is no existing system or programming environment educating exploratory programming for novices. Therefore, we have developed a tool, named as Pockets, to support novice's exploratory programming. Through Pockets, educators are able to identify where and when novices experience difficulties during exploratory programming. In addition, it is possible to assist educators' mentoring by referring collected logs through the proposed system. We have also conducted a case study and evaluated the usefulness of the tool. As a result, Pockets makes novices' exploratory programming more efficient, and also allows more accurate advice by educators.

## Categories and Subject Descriptors

K.3.2 [**Computer and Education**]: Computer and Information Science Education—*Computer science education*

## Keywords

programming education, exploratory programming, coding process visualization

## 1. INTRODUCTION

When expert software developers deal with unfamiliar programming languages and APIs, they usually perform the technique of exploratory programming for solving their problems[7]. Most existing research are focusing on supporting expert software developer's exploratory programming, aiming at improving their working efficiency[6, 5]. On the other hand, some other research have revealed that repetition of altering a portion of source code and checking the results after changes, which is being conducted in the manner of exploratory programming, is considered as a good practice of novices to improve programming skills[1, 3, 9].



*ESEC/FSE'15*, August 30 – September 4, 2015, Bergamo, Italy
ACM. 978-1-4503-3675-8/15/08
http://dx.doi.org/10.1145/2786805.2807564

In this research, since we regard exploratory programming as an effective way for improving novice programmer's skills, we propose Pockets, a web-based tool to support efficient exploratory programming for novices. In Pockets, we specifically focus on the feature of 'backtracking', regarded as one of the characteristics of exploratory programming. Backtracking means as "going back at least partially to an earier state either by removing inserted code or by restoreing removed code"[9]. Our system visualizes the programming process as a sequence of thumbnails, which shows a timestamp and an execution result when a user performs actions such as save, compile and run. Users could easily revert to earlier revision by clicking a corresponding thumbnail.

We evaluated our tool by conducting a case study, in which we asked novices to work on some provided programming exercises. As a result, novices who used our system perform exploratory programming more often than their counterparts who did not use the tool. In addition, we found that novices often conduct exploratory programming not only when they met compile or runtime error, but also when they got different results from correct outputs. The novices were very likely to spend much time for exploratory programming regardless of whether they caused errors. This indicates that educators should understand and support exploratory programming in introductory programming course.

## 2. BACKGROUND AND RELATED WORK

In software development, when developers need to deal with unfamiliar programming languages, APIs or new algorithms, they often develop by trial-and-error, in which they repeat many cycles of implementations and then evaluations. Rosson et al. investigated the main behaviors of software developers during their developments[4]. They reveal that developers follow the continuous progress of programming, executing and testing the program, then evaluating whether the program works as expectation or not. Brandt et al. shows that developers often revert their source code to earlier state, aiming at trying many more possible solutions based on results from their experiments[2]. This programming style that progresses through various attempts at running multiple implementations of source code, together with their respective evaluations is called the 'Exploratory Programming[7]'.

Sandberg et al. and Myers et al. described that it is important to evaluate frequent changes of code pattern repeatedly for keeping a high-quality software design[1, 6]. In addition, it desires for novices to conduct exploratory programming when they learn new knowledge. They also insist

that exploratory programming is highly recommended for novices who are learning techniques and knowledge for programming.

## 3. APPROACH AND UNIQUENESS

We developed a web-based tool, named as Pockets, to support exploratory programming for novices. Yoon et al. developed selective undo that is a major feature for IDE supporting "backtracking" while a user performs exploratory programming. However, we consider that even though their proposed system, named as AZURITE, is effective to improve smoothness of software development, but it is not designed to educate users to perform exploratory programming. Because AZURITE has too much functions, it will lead novices into confusing[8].

Pockets consists of three main areas, as shown in Fig. 1. The details of these areas are as follows:

**Programming Area** It allows users to write, save, compile or run his/her source code and check the results from the bottom dialog box.

**List of Revision Area** Whenever a user press the button of save, compile or run, a thumbnail is appended to the right to the List of Revisions Area. A thumbnail represents a revision of source code. The border color of thumbnail corresponds to the mentioned actions such as save(blue), compile(yellow) or run(red). Also thumbnail contains, (1) the time when the action is performed, (2) the corresponding result indicating whether compile or runtime errors occurred or not, and (3) ID.

**Differences View Area** This area displays the differences of source code and output between the newest revision and previous revisions. It allows users to check conveniently what characters, variables or sentences being added/deleted, and also the what results or outputs being produced in the corresponding revision.

Usage scenarios of Pockets are as follows:

**Step 1** User programs her/his assignments in programming area.

**Step 2** When the user presses the save, compile or run button, a thumbnail representing the newest revision is appended to the right to the List of Revisions Area.

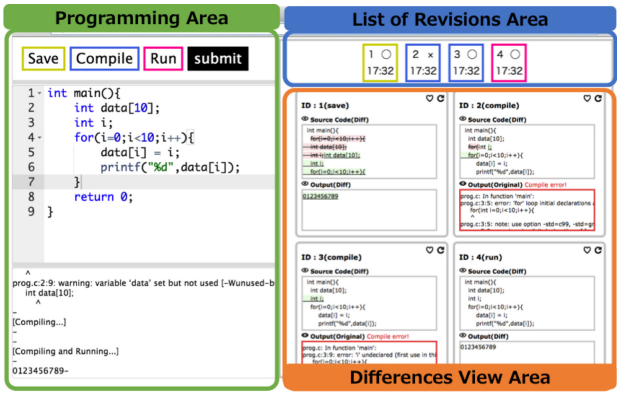


**Figure 1: The UI of Pockets**

**Table 1: Number of backtracking(The number indicates the times of performed backtracking by using our implemented function while the number inside the parenthesis refers to backtracking without using the function. )**

| Provided Features | Subject | |
|---|---|---|
| | sub1 | sub2 |
| All | 2(2) | 4(1) |
| Only programming Area | 0 | 1 |

At the same time, a corresponding box which shows the difference information is created in the Difference View Area.

**Step 3** Whenever a user wants to revert her/his source code to previous revision, she/he can use the backtracking function. By simply clicking the thumbnail in the List of Revisions Area or the arrow icon in the Difference View Area, the current source code is reverted to the corresponding previous revision.

## 4. RESULT AND CONTRIBUTION

We performed a small-scale case study for investigating how Pockets promotes exploratory programming for novices. In the case study, we separated ten novices into two groups, Group A and Group B, as each group consists of five novices. We prepared two program assignments(sub1 and sub2), and both of them only required basic programming skills such as array, loop function and conditional statement for solving. Both groups tackled sub1 simultaneously, as students of Group A works in Pockets with full functionalities, while Pockets being used by Group B only has the Programming area, indicating no convenient backtracking function. By the time when both groups solved sub2, they swapped their programming environment.

Table 1. shows the result of case study. Compared to the environment with only Programming area, novice performs more exploratory programming while using Pockets. Furthermore, according to the detailed logs gathered during backtracking, novices reverted to 1 or 2 revisions without using the function of Pockets. On the other hand, novices reverted to more previous revision with using Pockets, which is difficult to revert by using normal undo[1] function. This result indicates that Pockets helps novices to perform their exploratory programming.

Based on the results, we summarized two main contributions of Pockets for novices and educators as follows:

**Novices** can perform exploratory programming more conveniently than regular programming environment.

**Educators** can collect data about the timing when novices use exploratory programming and which revision they revert to. Such data are helpful hints for educators to provide more corresponding supports to novices.

As a future work, we are planning to perform a larger scale experiment to collect more concrete data, and a comparative experiment with undo command to understand how novices usually backtrack and what method is most desired for novice's backtracking.

[1]It erases the last change and reverts to one older state.

## 5. REFERENCES


[1] Variations to support exploratory programming. `http://www.exploratoryprogramming.org/`.

[2] J. Brandt, P. J. Guo, J. Lewenstein, M. Dontcheva, and S. R. Klemmer. Two studies of opportunistic programming: Interleaving web foraging, learning, and writing code. In *Proc. SIGCHI Conference on Human Factors in Computing Systems*, pages 1589–1598, 2009.

[3] B. A. Myers, S. Oney, Y. Yoon, and J. Brandt. Creativity support in authoring and backtracking. In *Proc. Workshop on Evaluation Methods for Creativity Support Environments at CHI*, pages 40–43, 2013.

[4] M. B. Rosson and J. M. Carroll. The reuse of uses in smalltalk programming. *ACM Transactions on Computer-Human Interaction*, 3(3):219–253, 1996.

[5] J. Sametinger, A. Stritzinger, and J. Kepler. Exploratory software development with class libraries. In *Proc. 7th Joint Conference of the Austrian Computer Society and the John von Neumann Society for Computing Sciences*, pages 24–31, 1992.

[6] D. W. Sandberg. Smalltalk and exploratory programming. *ACM SIGPLAN Notices*, 23:85–92, October 1988.

[7] B. Sheil. Environments for exploratory programming. *Datamation*, 29(7):131–144, July 1983.

[8] H. Tamada, A. Ogino, and H. Ueda. A framework for programming process measurement and compiling error interpretation for novice programmers. In *Proc. Joint Conference of the 21st International Workshop on Software Measurement and 6th International Conference on Software Process and Product Measurement*, pages 233–238, 2011.

[9] Y. Yoon and B. A. Myers. Supporting selective undo in a code editor. In *Proc. 37th International Conference on Software Engineering*, pages 223–233, 2015.