A tool for Automating Repetitive Tasks

Qi He
1002509831
University of Toronto
heqi4@cs.toronto.edu

Meng Zhang
1004063292
University of Toronto
mzhan63@cs.toronto.edu

Wei Zheng
999986261
University of Toronto
zhengw14@cs.toronto.edu

ABSTRACT
Performing repetitive tasks on the computer are both boring and error-prone. We designed an application to identify and automate the repetitive tasks to help computer users completing these tasks. This report shows our design justifications, introduces functionality of the application and provides a sample walk-through from users' perspective to demonstrate how this application will be used.

AUTHOR KEYWORDS
Repetitive computer tasks; automation; interactive operation; interaction elements.

ACM CLASSIFICATION KEYWORDS
H.5.m. Information interfaces and presentation (e.g., HCI): Miscellaneous.

INTRODUCTION
The graphical user interface (GUI) allows users to interact with electronic devices through graphical icons and visual indicators. It makes many users who do not have specific knowledge in the computer able to complete their tasks using a computer [1]. However, for the repeated actions, it cannot be performed effectively or efficiently [7]. Very often we must do some simple repetitive actions on the computer frequently, such as mouse movements, keystrokes, and text input. This work is monotonous and extremely boring. Besides, human factor causes mistakes even when we are doing repetitive work. Thus, optimization and automation of computer work are very important and helpful.

In our work, we designed a system to support the detection and automation of the repetitive operations. Our system was designed based on programming by demonstration(PbD), which is a method let user demonstrate what the task to automate should do, and formulate a program from observing this demonstration to perform the tasks automatically. The system has a design to avoid influencing user's behavioral mode. It tries to record and learn user's habit silently before it detects the iteration loop. Our application will monitor and record user actions in four aspects, including mouse click, keyboard input, file selection and windows changes[6]. By analyzing these actions after user repeatedly operates twice, our system could intelligently identify the loop of the repetitive operations. Afterward, when the user tends to perform the loop for the third time, a notification informs the user that repeated actions are detected and ask whether the user wants those actions to be conducted automatically. If the user confirms the request, a file or multiple files can be chosen to be processed automatically following the repetitive actions learned from the demonstration. Finally, the system would notify user tasks have completed successfully.

The goal of our work is to realize repetitive actions be dynamically identified and automatically performed. We are committed to improving the user experience in handling repetitive tasks effectively. Apart from completing the function successfully, we also focus on enhancing the interaction between human and computer[8] in dealing with repetitive tasks and reducing the learning cost and burden on use for the user.

DESIGN
In this phase, we did a series of research on the paper about automation of repetitive tasks on the computer and existing tools on the market for recording the repeated actions, aiming to create a system which could provide a better interactive experience for users. We have some changes compared to our low fidelity. In our low fidelity design, we planned to provide a user interface. Which let users tell the application that they may start a series of repeated actions by clicking the button "Start Recording." Then, after the user selects the files, if he/she wants the computer to perform these tasks automatically, the user needs to notify the application to apply these actions by pressing the button "Apply Actions." Although this design could let users understand the work flow of the system, it seems superfluous in the real scenario.

We figured that the previous design involved some drawbacks in user experience, which could damage the user's experience with our application. First, turning on the application and performing a specific action before the repeated actions is a user behavior that needs to be cultivated [3]. In an ideal user experience of an application, we should not change user's behavioral mode or increase learning cost [3]. Second, most repetitive tasks are relatively simple. The user does not want to spend time on learning how to use a tool and interacting more with a computer to tell it what should do. Therefore, we improve our design in the aspects of interaction, consistency, and ease of use.

In the current design, our design goal is to intrude minimally on the user's activities [4]. Therefore, the application avoids asking the user for information. The only information comes from recording the user's ongoing actions[9]. The user does not need to signal the start of the repeated actions. We use an implicit mode to monitor and track user's actions. The application does not have any redundant interaction with users before it detected the loop of iteration[2]. When the system identifies the repetitive tasks the user performed, there will be

a prompt window (Figure 1) to ask users whether they want those repetitive tasks to be performed automatically. The user only needs to click "Yes" or "No" to finish the interaction. If the user chooses "Yes", the application would complete the task without user's actions. The application will ignore the same repetitive tasks if the user chooses "No".

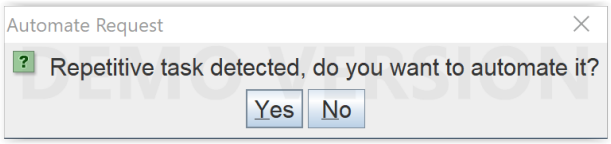


Figure 1. Automate Request Prompt Window

Another important design decision of our application is that we would like to represent actions in the same modality that the user employs for performing those actions [5]. Because of the concern about losing control of the computer, users prefer to know how the actions are being performed. Some of the users not only want to know what functions can the application achieved but also want to see the process. On the other hand, if the user does not want the application demonstrates the processing flow, the system would have the window (Figure 2) to ask users whether they want to skip details of performing the repetitive tasks when there is a file (or files) chosen. If the user chooses to skip details, those tasks would be completed implicitly. Users will only be presented the results after auto-completion. This option could be very useful when the user selects a massive number of files to apply the automation process on.

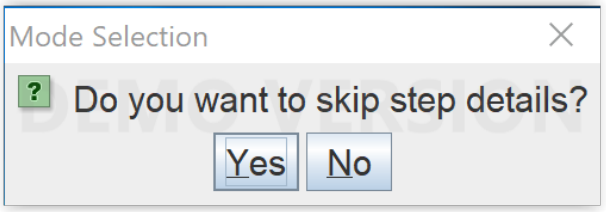


Figure 2. Mode Selection Prompt Window

After the automatic tasks are completed, the application would provide a prompt widow (Figure 3) to notify the user that the files he/she chose have completed successful and ask the user whether continue to process other files. If the user chooses "Yes", he/she could select the files to perform the repetitive tasks automatically. If the user selects "No", the record of the repetitive task will be removed from the action recorder and the events monitor will be restated.

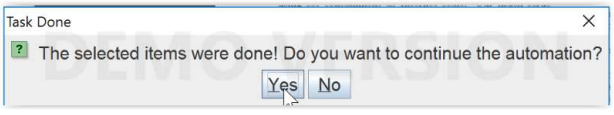


Figure 3. Continue Confirmation Prompt Window

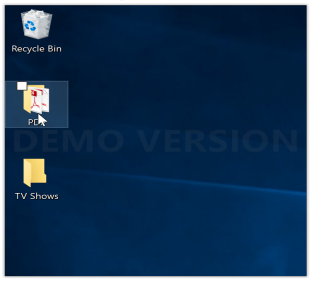
The action detector supports to identify the repetitive tasks in any operated order. It could accurately detect the iteration loop, even the actions are performed in different order between the twice demonstration given by the user. Our application will perform the actions automatically based on the sequence of the first loop the user performed. We defined the different events on mouse and keyboard with different value. After the application captures these events, it analyzes the actions by eliminating the effect of their sequence based on the values. By doing this, it can detect any loop of iteration even in the situation with noise.

We also designed the fallback feature. If the automation results are not desirable, users can utilize the hotkey "Ctrl + Z" to undo these changes. By doing that, the application will revert to the state before the automatic actions. User also can utilize the hotkey "Ctrl + Y" to redo an application. These functions can guarantee user get the desirable results via this automation of repetitive tasks.

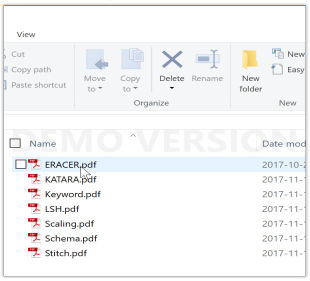
Storyboard

One typical scenario is for extracting titles from PDFs. This is particularly useful to scholars and students who need to make citations or literature lists from the massive amount of academic papers. A narrative walk-through of how the automated program will work in this scenario is shown in Figure 4.

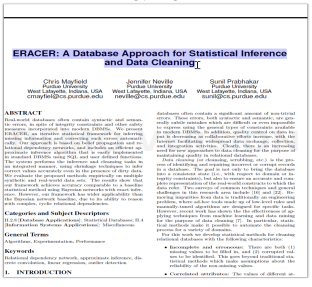
User opens PDF folder



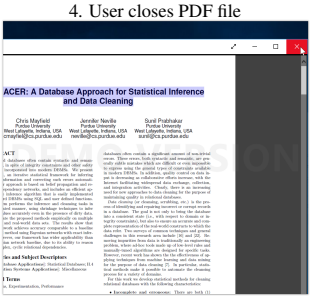
2. User opens the first PDF file



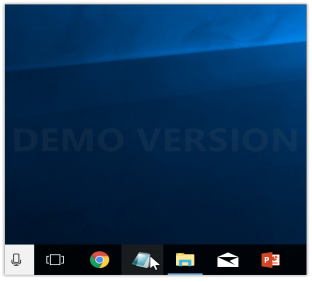
3. User selects title of the PDF and hits Ctrl+C



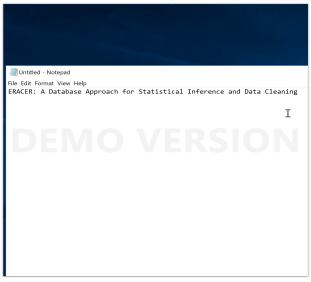
4. User closes PDF file



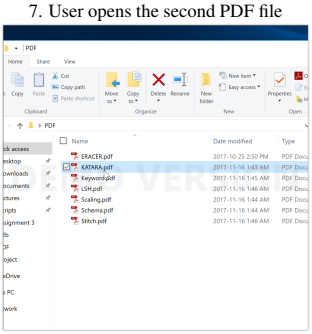
5. User launches Notepad



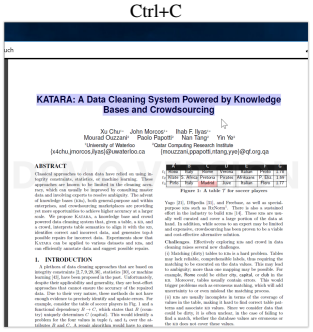
6. User hits Ctrl+V to paste the copied title to Notepad



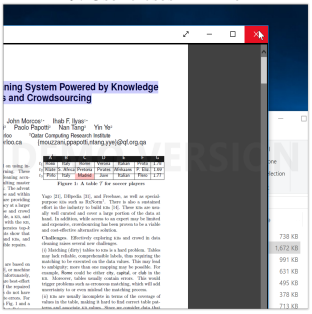
7. User opens the second PDF file



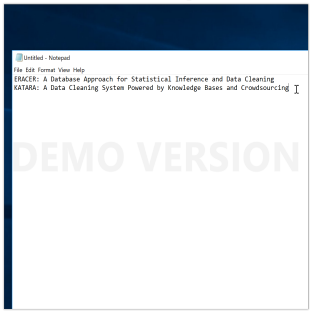
8. User selects title of the PDF and hits Ctrl+C



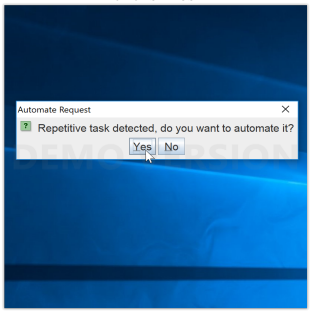
9. User closes PDF file



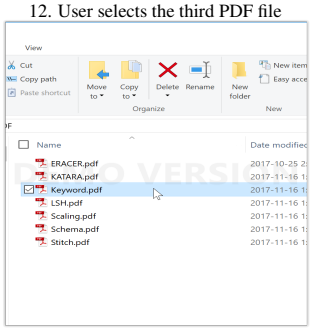
10. User hits Ctrl+V to paste the copied title to Notepad



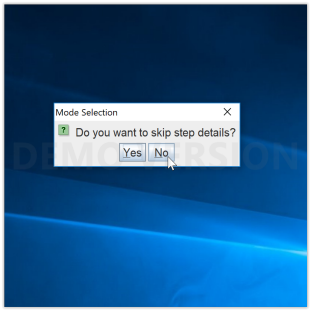
11. User sees a prompt window asks if he/she wants to automate the task. User clicks 'Yes'



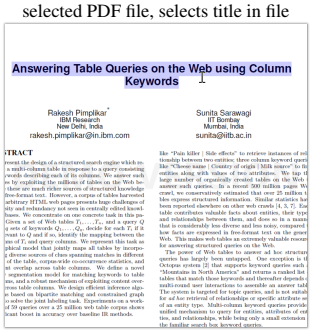
12. User selects the third PDF file



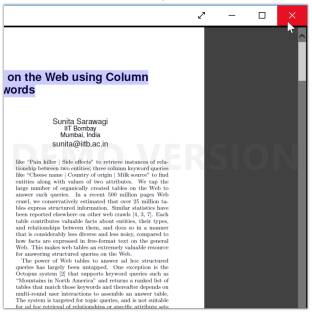
13. User sees a prompt window asks if he/she wants to skip the details of repeated steps on the select file. User clicks 'No'



14. User sees the program opens the selected PDF file, selects title in file

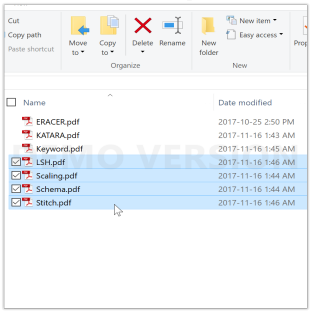


15. User sees the program closes the PDF file

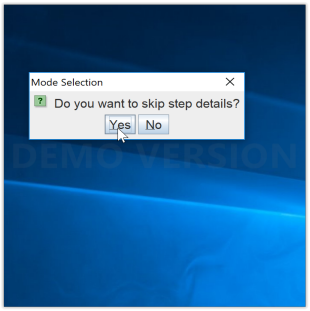


16. User sees the program pastes the third PDF's title to Notepad

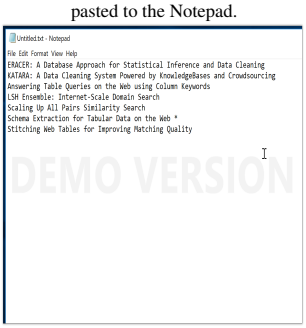
17. User selects multiple PDF files



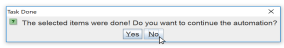
18. User sees a prompt window asks if he/she wants to skip the details of repeated steps on the select file. User clicks 'Yes'



19. User sees all selected PDFs' titles are pasted to the Notepad.



20. The program notify the user "the selected items were done" and ask whether he/she want to continue the automation. The user clicks "No".



21. The user hit ctrl + Z to undo the last automate application.

22. The user hit ctrl + Y to redo the last operation, recover the automation results and finish

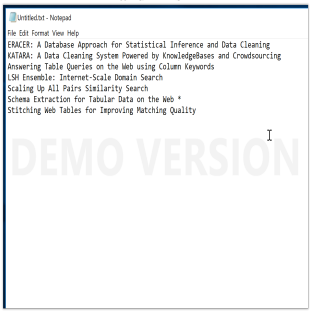


Figure 4. Storyboard

IMPLEMENTATION

Task Breakdown and Workload Split

We divided the workload into several implementation tasks based on the design:

1. Record user actions
   a. Record the mouse events such as click, wheel, drag and their pointer positions.
   b. Record keyboard events such as typing and hotkey combinations.
2. Analyze the repetitive actions
   a. Intelligently identify repetitive routines from recorded actions and components from Task 1.
   b. Generate repetitive action queue according to the routine.

3. Build user interface to interact with users.
    a. Implement the Automate Request, Mode Selection and Continue Confirmation prompt windows.
    b. Verify the wording and options for each prompt window.
4. Implement repeat Robot.
    a. Implement functions to repeat user's actions with detailed steps if the user chooses 'Yes' in Automate Request (use automate function) and 'No in Mode Selection (do not skip step details).
    b. Implement functions to repeat user actions by only showing the result when done if the user chooses 'Yes' in Automate Request (use automate function) and 'Yes' in Mode Selection (skip step details).
    c. Implement functions to repeat user actions when the user chooses "Yes" in Continue Conformation and selects other files.
5. Implement neglecting function to ignore repetitive actions if user chooses 'No' in Automate Request (do not use the automate function)

We provide the source code of the application on Github. Please refer to the appendix for the link.

Developer Platform

We decided to use Java to implement this project with the leverage of Java's Robot and AWT packages. Compared with other programming Language, Java has its natural advantages. It supports cross-platform. Java has the features, which can be compiled in various operating systems and do not need to use virtual machine. Besides, Java provide a series of APIs to complete window and events tracking and monitoring to enable us implement the repetitive actions detection and automation.

Task Split

Due to the dependencies of tasks, it was difficult to split tasks among all three group members. To ensure the efficiency and code consistency, we had two group members working on the coding; the other one person was responsible for providing related documentation, solution suggestions through research. This person also wrote use cases according to the tasks above and the Storyboard section.

Use Cases

The following use cases were tested. The results revealed bugs during each iteration of testing. After fixing bugs for all iterations, our code passed all testing scenarios.

1. As a user, I should be able to complete my actions in different orders, and the application can still identify repetitive tasks.
2. As a user, I should be able to complete my actions with interruptions (e.g., check emails, dismiss notifications.) and the application can still identify repetitive tasks.
3. As a user, I should be able to choose if I want to automate the repetitive tasks.
4. As a user, I should be able to choose if I want to see the detailed steps.
5. As a user, I should be able to select a file to apply the automation
6. As a user, I should be able to select multiple files to apply the automation.
7. As a user, I should be able to terminate the automation.
8. As a user, I should be able to continue the automation.
9. As a user, I should be able to undo the last application of the automation
10. As a user, I should be able to redo the last operation of the automation.
11. As a user, I should be able to finish the automation and remove the record of iteration loop.

Duration of Implementation

It cost us 4 hours to break down tasks and assign workload. The two developers spent 13 hours in total over two weeks to implement the code. The research took 6 hours. Testing and bug fixing took all three group members 15 hours spreading over four days.

REFERENCES

1. Peter Achten, Marko Van Eekelen, and Rinus Plasmeijer. 2003. Generic graphical user interfaces. In *Symposium on Implementation and Application of Functional Languages*. Springer, 152–167.
2. B Douglas Blansit. 2008. Automating repetitive tasks: getting started with hotkeys. *Journal of Electronic Resources in Medical Libraries* 5, 2 (2008), 187–198.
3. Margaret Burnett. 2003. HCI research regarding end-user requirement specification: a tutorial. *Knowledge-Based Systems* 16, 7 (2003), 341–349.
4. Allen Cypher. 1991. EAGER: Programming Repetitive Tasks by Example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*. ACM, New York, NY, USA, 33–39. DOI: http://dx.doi.org/10.1145/108844.108850
5. Sumit Gulwani. 2015. Automating Repetitive Tasks for the Masses. In *Proceedings of the 42Nd Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL '15)*. ACM, New York, NY, USA, 1–2. DOI: http://dx.doi.org/10.1145/2676726.2682621
6. Urmila Kukreja, William E Stevenson, and Frank E Ritter. 2006. RUI: Recording user input from interfaces under Windows and Mac OS X. *Behavior Research Methods* 38, 4 (2006), 656–659.

7. David Kurlander and Steven Feiner. 1992. A History-based Macro by Example System. In *Proceedings of the 5th Annual ACM Symposium on User Interface Software and Technology (UIST '92)*. ACM, New York, NY, USA, 99–106. DOI: http://dx.doi.org/10.1145/142621.142633

8. Jean Scholtz, Jeff Young, Jill L Drury, and Holly A Yanco. Evaluation of human-robot interaction awareness in search and rescue. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, Vol. 3. IEEE, 2327–2332.

9. M Tomizuka, G Anwar, and Bin Fang. 1988. Control of robot manipulators under repetitive tasks-segmented repetitive control approach. In *Intelligent Robots, 1988., IEEE International Workshop on*. IEEE, 157–163.

APPENDIX

Source code Link:
https://github.com/zhan4806/CSC2514Group9

Note: This link contains the demo version of the application. We developed the application in Surface pro and have not tested on other machines. We are not sure whether there will be a resolution issue as the surface pro has a unique resolution for the display. Please contact us if you run into any problem.