Project Report: Editor Scalability
CS 427 Software Engineering, Fall 2012
Team G22 Bagyamn

Members:

1. Srikanth Srungarapu (srungar2)
2. Lionel Li (lli53)
3. Samuel Oshin (soshin2)
4. Sugandha (sugandh2)
5. Nisha Bagyam Nallasivam (bagyamn2)
6. Rucha Kanade (kanade2)

Table of Contents

1. Brief Description of Project

The current Photran editor behaves badly for large files. For very large files with several thousand lines of code, the editor is not able to keep up with typing. It freezes for several seconds before the user can see the characters that he typed. It also takes a long time to load a large file into the editor.

The main goal of our project is to improve the performance of the Photran editor for large files. In order to do this, we need tests to measure the performance of the original editor, and tests to show the improvement in performance after implementing our solutions.

This project is primarily about reverse engineering the system and comprehensive testing. Our implemented solutions are as follows:

1. Move the parser to the reconciler thread
2. Automated testing using SWTBot

2. Architecture & Design

Main Issue

We realized that the Photran Editor is very irresponsive when typing in a large Fortran file (>25 kLOC), and it gets worse with bigger files. We tried opening and typing on automatically generated Fortran source code that has about 75 kLOC, and the Editor gets unbearably slow as we type, to the point of freezing for more than 5 seconds. We investigated the issue and realized that parsing is the main bottleneck of the whole function call. Parsing a 75 kLOC file takes about 5 seconds, and it takes place every time a key press is captured. All this heavy work is taking place in the main UI thread.

Approach To Address Issue

We thought of doing the parsing in another thread instead of the main UI thread so that the Editor can still be responsive.

Failed Attempts

We wanted to do the parsing on a separate thread that gets scheduled after a certain time period (e.g. 3 seconds) have passed. What we did was to refactor the parse() method such that it becomes a TimerTask. Having a TimerTask is convenient as we can create a Timer that runs the TimerTask and specify a certain delay for the timer to execute the TimerTask. However the main drawback is that the Reconciler is running on a separate thread and it has a delay of 500ms. Parsing needs to be done before Reconciler kicks in as the Reconciler updates the OutlineView control. A naïve approach would be to make the Reconciler delay for a time period that is greater than the TimerTask's delay. This would ensure that by the time the Reconciler is running, the TimerTask would have completed, and the VPG and AST data structures would have been built.

However this is not a good practice and it is not always reliable. We couldn't find a way to notify the Reconciler thread to run once parsing is done, so we moved on.

We had the thought of creating a new CDTBasedTextEditor.java that extends the existing editor, but we weren't sure if we will break any functionality in the Editor. Moreover when we were stepping through the code, we noticed that the FortranModelBuilder.parse() method gets called from CDTBasedTextEditor.getAdapter(), which eventually calls several WorkingCopy methods. With these chains of events happening, we concluded that there are too many code changes involved and it would be challenging to make these changes safely. Moreover, the TranslationUnit.parse() method gets called and we have no way of stopping that because WorkingCopy.java and TranslationUnit.java are all library files. We cannot modify them.

Successful Attempt

Since we couldn't find a way to prevent FortranModelBuilder.parse() from being called by the main UI thread, we decided to make it quit immediately when the main UI thread is in the parse() method. We identify the main thread by checking the name of the thread against the default name of the main UI thread, which is "main".

With the knowledge that the parse method can be triggered by a sequence of method calls within WorkingCopy.java to TranslationUnit.java, we added code to retrieve the active TranslationUnit object in the editor, and tell it to reconcile (which will eventually build the AST model if there are changes) in the Reconciler's thread, just before it updates the Outline View.

Code Changes

In FortranModelBuilder.java:

```
private static final String MAIN_THREAD_NAME = "main"; //$NON-NLS-1$

private boolean isMainUIThread() {
  return Thread.currentThread().getName().equals(MAIN_THREAD_NAME);
}

public void parse(boolean quickParseMode) throws Exception {
  if (isMainUIThread()) {
    return;
  }
    // The rest of the code...
}
```

In FortranVPGReconcilingStrategy.java:

```
private void tryComputeAST() {
  try {
    ICElement translationUnit = getInput(editor);
    ((IWorkingCopy) translationUnit).reconcile(true, false, null);
  } catch (Exception e) {
    System.err.println("Attempt to compute AST failed"); //$NON-NLS-1$
  }
}
```

Packages Changed

We made code changes in the following packages:

1. Project name: **`org.eclipse.photran.cdtinterface.vpg`**
   Package name: **`org.eclipse.photran.internal.core.model`**
   File modified: **`FortranModelBuilder.java`**

The Model Builder contains the code to parse the source code and build the AST and VPG. It is a time consuming task that was previously run on the main thread. We improved the responsiveness of the editor by preventing the main thread from running this function, and made the Reconciler thread run it instead.

2. Project name: **`org.eclipse.photran.ui.vpg`**
   Package name: **`org.eclipse.photran.internal.ui.editor_vpg`**
   File modified: **`FortranVPGReconcilingStrategy.java`**

This is where the reconciler thread will call the reconcile function. It will result in the Outline View to be updated. In our solution, we call the working copy translation unit to reconcile and compute the AST before updating the Outline View.

3. Project name: **`org.eclipse.photran.ui`**

Automated test cases were added to this package, under the folder **swttests**

Flow Diagrams

When opening a Fortran source code in Photran:

Before: **After:**

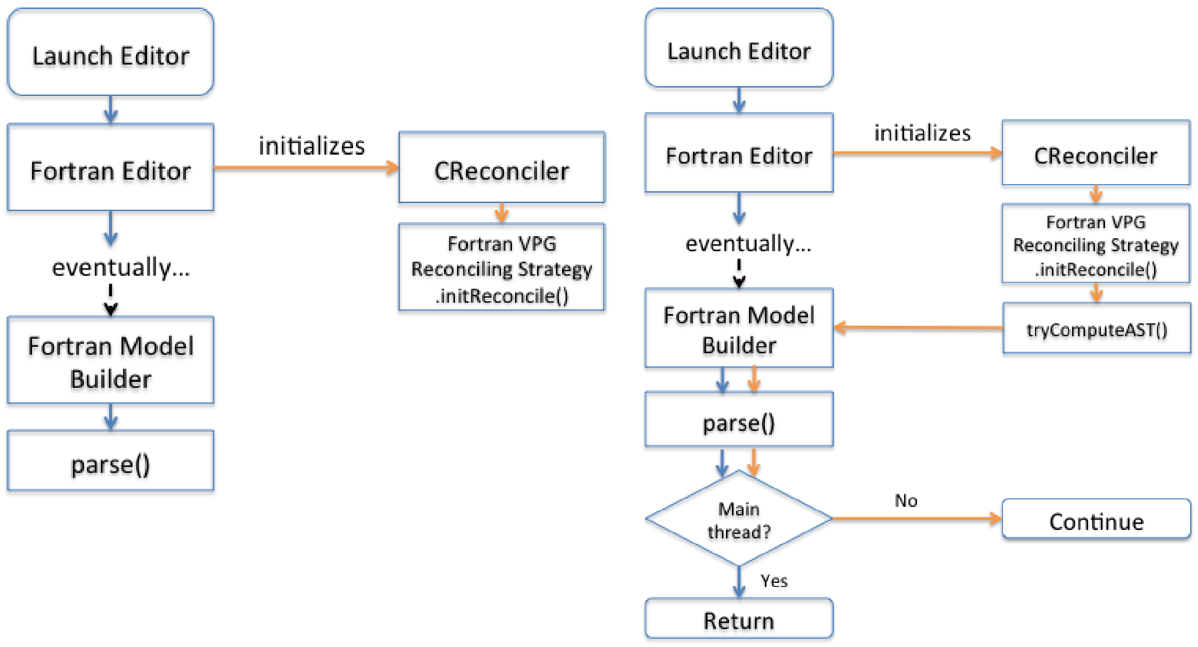
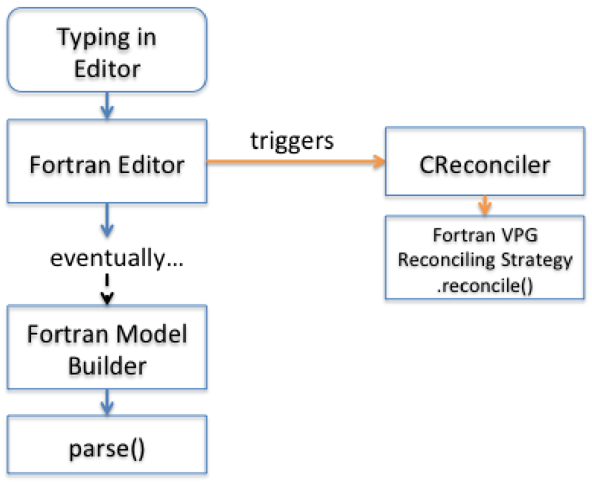


Figure 1: Control Flow when launching the Photran editor

When typing in Photran

Before:



After:

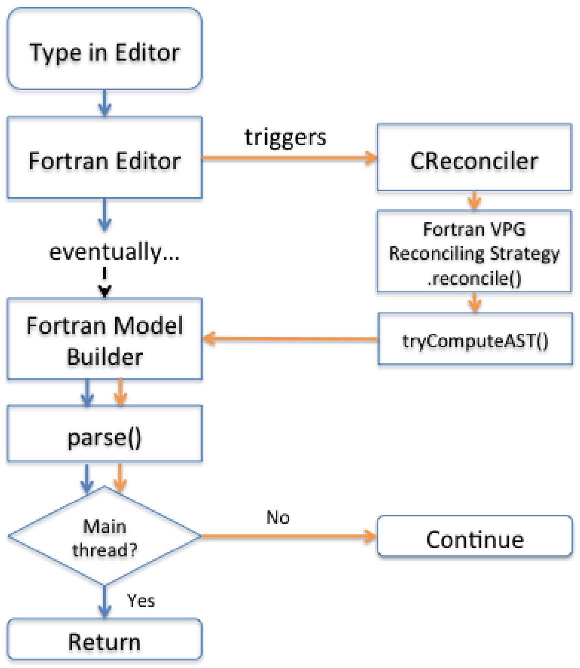


Figure 2: Control flow when typing in Photran editor

Note: The blue arrows represent the path flow of the main UI thread. The orange arrows represent the path flow of the Reconciler thread.

3. Future Plans

We plan to upload our code to github and continue working on it through the winter vacation. There were some tasks on our agenda that could not be completed due to time constraints. We plan to work on these and submit our code for inclusion into the plugin.
We will be specifically focusing on the following:

1. Include a GUI option to implement the scalability option.
2. Try to load a large file seamlessly.
3. Test more thoroughly, because there is always a requirement of more testing in any system.
    a. We used SWT bot for testing and noticed some inconsistencies in its behavior. It gives the testing time readings in a range of +/- 20 for the same file. We plan to get to the root of these deviations observed in its behavior.
    b. We also want to explore alternative testing strategies.
4. We found out that spell-checking is slow when there are a lot of constant strings. We want to explore this option to see if it gives any improvement over our solution.
5. We will try to find out why the AST and VPG tasks are not running.

Personal reflection by each member:

Lionel:
The Editor Scalability project attempts to address the issue with the Photran editor where the editor does not scale well with the size of the file. The process of reverse engineering the Photran Editor is very challenging. It was difficult to know where to start as the whole project it huge and consists of many functions. Some components run on different threads, and the Photran Development Guide didn't really help us too much in our project, as it didn't have much documentation on the classes and components that pertain to our development. The XP process is particularly useful. Working in pair programming fosters us to think together and get everyone involved in a project as large as the Photran Editor. I find the testing process critical to our development process as it helps redefine our solution.

Srikanth:
This project helped me in gaining insights about how to approach the problems related to scalability in heavy weight applications like Eclipse. It also taught me the importance of how debugging can be helpful in reverse engineering. The existing documentation is not really up-to-date and didn't really help us in reverse engineering. There aren't really many GUI testing frameworks for editors. And it is not really difficult for us to narrow down on using SWTBot. However, writing tests is not really straight forward as the tests involve asserting on appearance of widgets and refreshing editors. It also taught me about how to co-ordinate and finish the tasks while working in larger groups.

Nisha: This project changed my impression about testing and made me realize how significant testing is. I learned to write automated GUI testing using SWTBot and it felt good to see the green bar and UI running automatically. The most time consuming and challenging part was the reverse engineering where we had absolutely no clue initially where to start. The project served as a good means to boost my confidence to work on big size projects next time.

Sugandha: This project taught me how to ramp up on a new area quickly and understand the basic workings without reading every line of code. It helped me develop the skill to read somebody else's code intuitively. The continuous emphasis on testing helped us ensure a good quality final product. We often underestimate the importance of testing as a coder, but this project helped to make continuous testing a habit.

Rucha: Working on this project gave me a flavor of working on a large scale project. I realized that software engineering is not always about writing new code; understanding code written by someone else is an equally important skill. The project helped me improve my reverse engineering skills. I learned how to quickly navigate to different parts of the project, in order to understand the flow of logic between the different components of the system. GUI testing using SWTBot was an area that was new to me, and the rigorous testing that we needed to perform for our project helped me get to know the different issues associated with automated GUI testing.

Sam: Trying to identify issues with editor scalability was a task, which at first, didn't seem like much of a project. This was the complete opposite. Getting an understanding of what the project demanded from us took a long time. Our initial solutions were shut down due to the lack of understanding. After revising our plan and solution, a more substantial solution was proposed and implemented. A prominent issue with this project is the lack of guidance with the framework of our solutions. I feel a little more explanation as to what and how we should go about solving the issues, especially what not to consider solutions. For example, the possible solution listed in the project description is a solution that was turned down by the TA. Finding a good and usable GUI testing framework is another issue. Trying to look for a GUI tester, and then implement it with our project was another difficult but surmountable task. There is a lack of GUI testers, and some of them are actually proprietary. This increased the time it took to create decent automated tests because of the steep learning curve and lack of documentation for the few open source testers. Another difficult task was reverse engineering. This was especially hard because the documentation on the editor and its various components is severely lacking. Took copious amounts of time to identify the various aspects of code we needed and the various features it affected. Overall the project was hard, but beneficial in acclimating me to reverse engineering.

4. Testing

In order to automate the UI testing for our project, we used the SWTBot framework. SWTBot is an open-source Java based functional testing tool for testing SWT and Eclipse based applications. More information about SWTBot can be found at the following links:

http://wiki.eclipse.org/SWTBot/UsersGuide
http://www.vogella.com/articles/SWTBot/article.html

Test Cases

For each of the test case, we measure the time taken. The start variable is initialized before typing starts. The end variable takes the time after typing ends. The difference in time is stored. These test cases were run first without the solution and the values were taken and populated in the test cases so that they can be compared with time take for the test cases with the solution. The assert statements are true if the new time taken is less than old time. There is a delay set after every character insertion to simulate typing.

- **testOutlineView** - Inserts a module in the large file and checks if the module is updated in the outline
- **testTypingOnReconcile** - Creates a delay such that reconcile is hit and then types again.
- **testSimulateHumanTyping** - Generates random line numbers, to insert text in the large file.
- **testrandomwaitTyping** - Generates random line numbers and random delays after every character insertion.

Test Results:

Size of file	Test Name	Time taken (Before)	Time taken (After)
75000	testSimulateHumanTyping	160318, 191213, 194480, 193375, 184498	92395,97329,91067, 95354, 93025
	testrandomwaitTyping	99219, 127005, 122227, 117234, 130022	32934, 30724,31034, 31543, 32008
	testOutlineView	40586, 40909, 41286, 41006, 40155	40185,39393, 39088, 38997, 41200
	testTypingOnReconcile	103495, 111806, 127954, 118965, 109876	37181, 35099, 35307, 36008, 36759
50000	testSimulateHumanTyping	95916, 99259, 104253, 98765, 100045	56627, 53989, 55247, 54254, 54890
	testrandomwaitTyping	69752, 63458, 66545, 67007, 65543	28060, 29635, 34828, 30766, 32871
	testOutlineView	39152, 39103, 39606, 39876, 40056	37986, 38269, 38164, 37951, 38657
	testTypingOnReconcile	71023, 65203, 65192, 66007, 65789	34670, 34491, 34648, 34354, 35002
20000	testSimulateHumanTyping	42842, 42485, 40474, 42089, 42897	26090, 26032, 26029, 25938, 25893
	testrandomwaitTyping	37559, 38392, 39515, 38867, 40087	21832, 27753, 30246, 27602, 24542
	testOutlineView	37551, 37749, 37968 38087, 36987	36755, 36896, 36686, 36625, 36664
	testTypingOnReconcile	39333, 39393, 39580, 39765, 40089	32922, 33021, 32940, 32562, 32487
10000	testSimulateHumanTyping	29319, 27647, 28382, 28237, 28477	21293,20893, 20912, 21670, 20543
	testrandomwaitTyping	31342,30560,31003, 29238, 28650	28849, 26400, 30016, 27890, 26785
	testOutlineView	37060, 36921,37011, 37134, 37011	36517, 36253, 36199, 35998, 36700
	testTypingOnReconcile	32201, 32443,32404, 32425, 33007	31960, 32037, 32009 32789, 31897
5000	testSimulateHumanTyping	22490, 23982, 23503, 23590, 22982	20738,20510, 20732 20987, 21008
	testrandomwaitTyping	26613, 31132,29273, 30437, 25725	27655, 30849, 24366 27520, 26789

	testOutlineView	36217, 36579,36745, 36698, 36552	35875, 35250, 35849 36789, 34908
	testTypingOnReconcile	31825, 31968,31879, 31844,31858	31818, 31782, 31861 31902, 30987

Table 1: Time improvements after implementing the solution

1. Average speedup

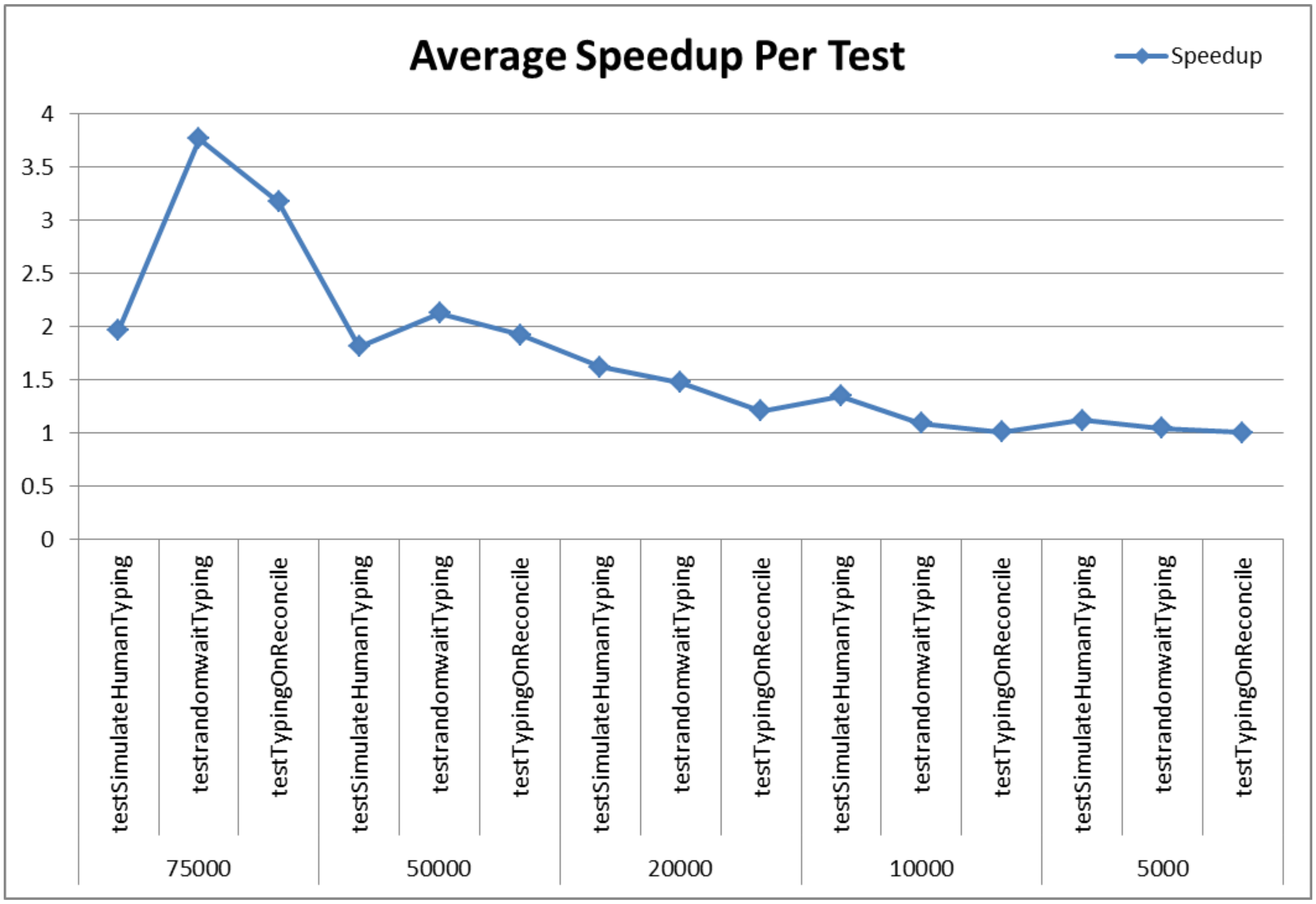


Figure 3: Average Speedup vs. Number of lines in editor

2. Average Time

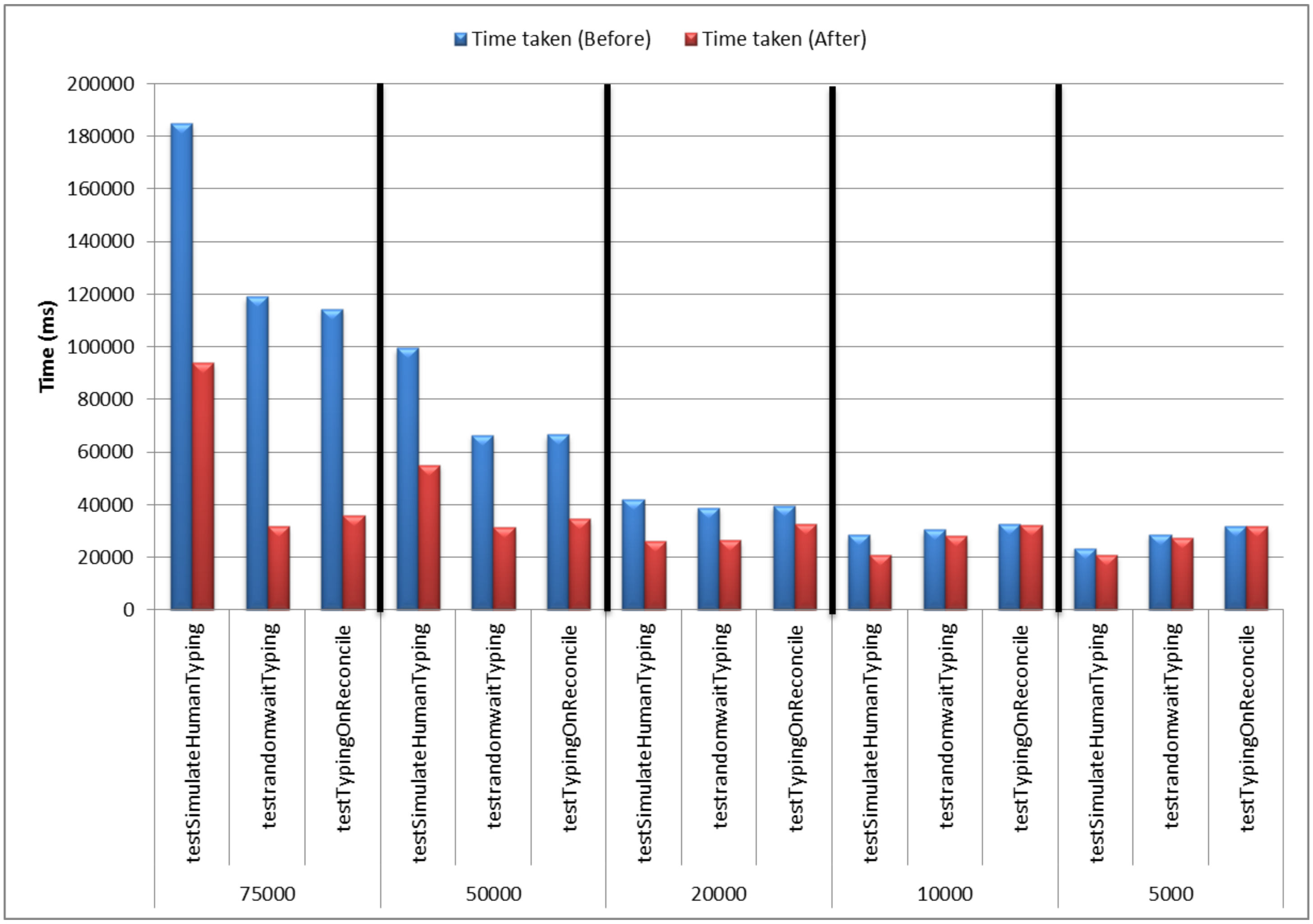


Figure 4: Average Time (before and after implementing the solution) vs. Number of lines in the editor

5. Appendix

Installation instructions:

Download the latest code from repository.
Repository location:
https://subversion.ews.illinois.edu/svn/fa12-cs427/_projects/G22/

Packages to be downloaded:

```
1. org.eclipse.photran.ui (for automated tests)
2. org.eclipse.photran.cdtinterface.vpg
3. org. eclipse.photran.ui.vpg
```

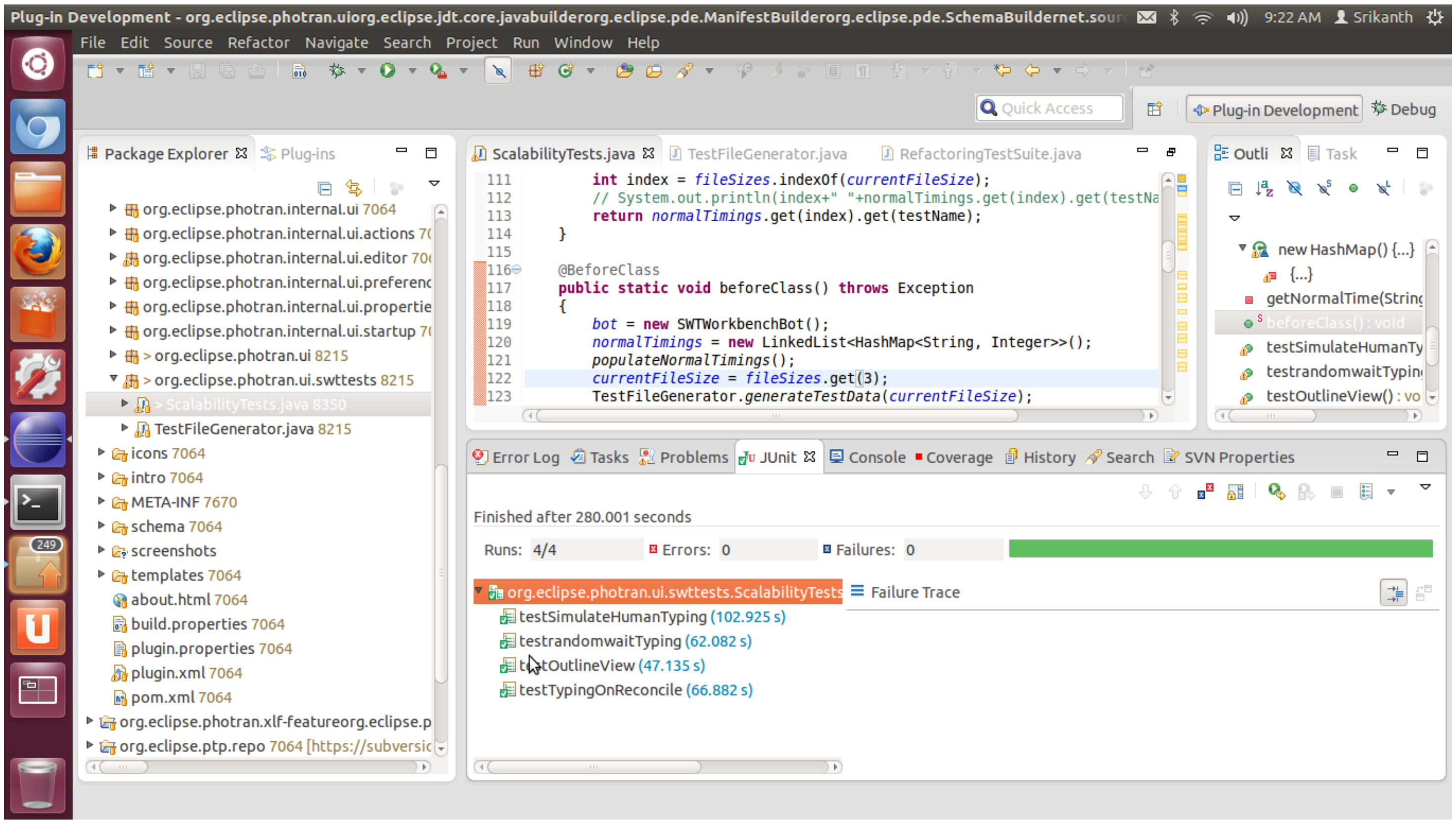
Steps to run the program:

No other changes need to be made. Editor can be used as before.

Steps to run the tests:

1. Navigate to the following location:
   org.eclipse.photran.ui/trunk/src/org/eclipse/photran/ui/swttests/

2. Right click on the test files individually - > Run As -> SWTBot test


Plug-in Development - org.eclipse.photran.uiorg.eclipse.jdt.core.javabuilderorg.eclipse.pde.ManifestBuilderorg.eclipse.pde.SchemaBuildernet.sour
9:22 AM
Srikanth
File Edit Source Refactor Navigate Search Project Run Window Help
Quick Access
Plug-in Development
Debug
Package Explorer
Plug-ins
org.eclipse.photran.internal.ui 7064
org.eclipse.photran.internal.ui.actions 70
org.eclipse.photran.internal.ui.editor 70
org.eclipse.photran.internal.ui.preferenc
org.eclipse.photran.internal.ui.propertie
org.eclipse.photran.internal.ui.startup 70
> org.eclipse.photran.ui 8215
> org.eclipse.photran.ui.swttests 8215
> ScalabilityTests.java 8350
TestFileGenerator.java 8215
icons 7064
intro 7064
META-INF 7670
schema 7064
screenshots
templates 7064
about.html 7064
build.properties 7064
plugin.properties 7064
plugin.xml 7064
pom.xml 7064
org.eclipse.photran.xlf-featureorg.eclipse.p
org.eclipse.ptp.repo 7064 [https://subversic
ScalabilityTests.java
TestFileGenerator.java
RefactoringTestSuite.java
Outli
Task
new HashMap() {...}
{...}
getNormalTime(Strin
beforeClass() : void
testSimulateHumanTy
testrandomwaitTypin
testOutlineView() : vo
Error Log
Tasks
Problems
JUnit
Console
Coverage
History
Search
SVN Properties
Failure Trace

```
        int index = fileSizes.indexOf(currentFileSize);
        // System.out.println(index+" "+normalTimings.get(index).get(testNa
        return normalTimings.get(index).get(testName);
    }

    @BeforeClass
    public static void beforeClass() throws Exception
    {
        bot = new SWTWorkbenchBot();
        normalTimings = new LinkedList<HashMap<String, Integer>>();
        populateNormalTimings();
        currentFileSize = fileSizes.get(3);
        TestFileGenerator.generateTestData(currentFileSize);
```

```
Finished after 280.001 seconds

Runs: 4/4     Errors: 0     Failures: 0

org.eclipse.photran.ui.swttests.ScalabilityTests
    testSimulateHumanTyping (102.925 s)
    testrandomwaitTyping (62.082 s)
    testOutlineView (47.135 s)
    testTypingOnReconcile (66.882 s)
```

Figure 5: Screenshot of JUnit test runs