SDAT a Visual Source Code Comparison Tool

Prepared by Rolf Biehn (rolfbiehn@yahoo.ca)

Introduction

Software developers are often faced with the task of addressing regression in code and maintenance of source code in multiple branches within source control. For these tasks, and many others, a tool capable of visually comparing source files is highly desirable.

My project, SDAT (Software Difference Analyzation Tool), intends to aid developers comparing similar source files.

Related Work

Numerous programs for source file comparison exist such as Beyond Compare, KDiff, and WinMerge. Several programs for XML comparison exist such as HTMLDiff, Araxis Merge and Guiffy. The problem of source code comparison was briefly addressed by Chevalier et al. (1). Muzner (2), Bauman (3) and Holten (4) explored issues regarding tree comparison visualizations. Several programs are available for visualizing source code such as Relief(which uses a radial gliff system) and aiCall (which uses flow-charts). Both Voinea (5) and Jones (6) used pixel maps to indicate areas of interest in source code. Appert (7) introduced the idea of orthogonal zooming.

Dataset and Processing

Source files must be pre-processed in order to detect differences and missing nodes. The results of this pre-processing will be used as input for SDAT. The main focus of this project is not the difference detection algorithm but the visualization of detected difference. Therefore, a minimal amount of work will be invested into the difference detection algorithm.

Multiple source code file revisions will be obtained from an open source JAVA project. Source files will be pre-processed with a JAVA to XML utility called JAVAML (8). Comparison will be done using an open source xml difference comparison library and some heuristics. This will result will be an XML structure augmented with meta-data detailing the differences and missing nodes. Some manual tweaking of this dataset is expected. If the tweaking becomes problematic, alternative pre-processing methods will be considered such as purely-textual comparison or manually preparing the difference data set based on the source files.

My Background

I have been programming computers for over 20 years, professionally for approximately 6 years. My current MSc. thesis is to design a comparison tool for program structure (across multiple languages).

Proposed Solution

Generalize Hierarchy

A program can be thought of as a very strict hierarchy of nodes, each of which contains properties on every hierarchical level. For example the class-level hierarchy contains properties regarding member variables, class name, visibility, etc… I will simplify this hierarchy for visualization purposes because I do not think that most developers care about the level of detail after statement. The very high level plan is to create multiple views of this tree which specialize in viewing specific levels of this hierarchy.

Figure 1 A simplified program hierarchy.

Navigation

Users can navigate this structure by navigating familiar tree widget controls and clicking tool bar items for next and previous differences. They can choose to create new views using the specialized viewers at any time.

Shared Components

Some common components will be used in many of the views. Each view will be split into two panes to visualize differences. The left pane will contain one version of the source code while the right pane will contain the other. Differences will be highlighted in red while missing nodes will be highlighted in blue. Each view could support filtering such as name matching and showing only the differences. Each view will contain a mini-map-scroller (described below).

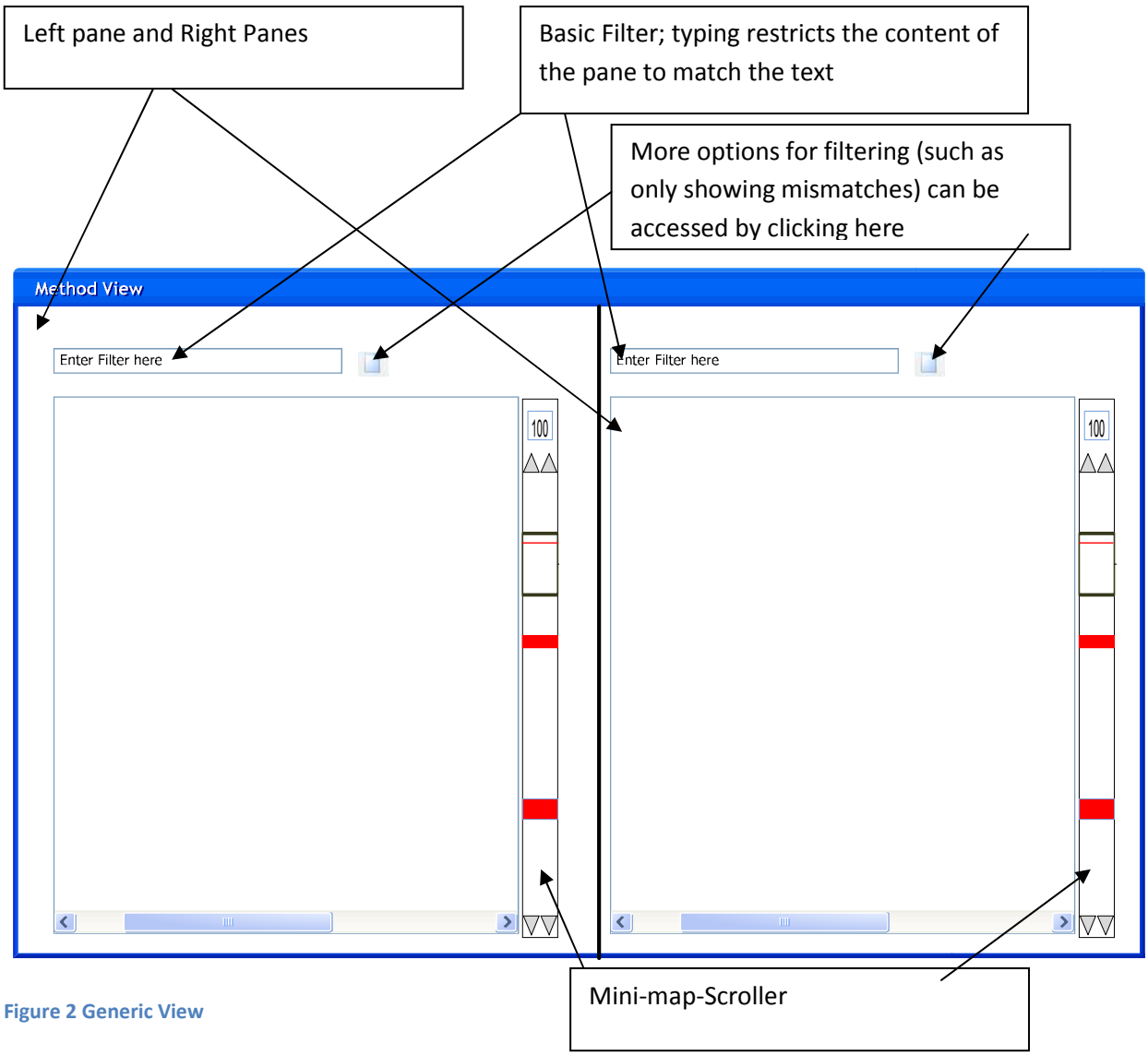

Figure 2 Generic View

Mini-map-scroller

A pixel map scroller will be used to give the user an overview of the differences in the pane as well as assist in navigation.

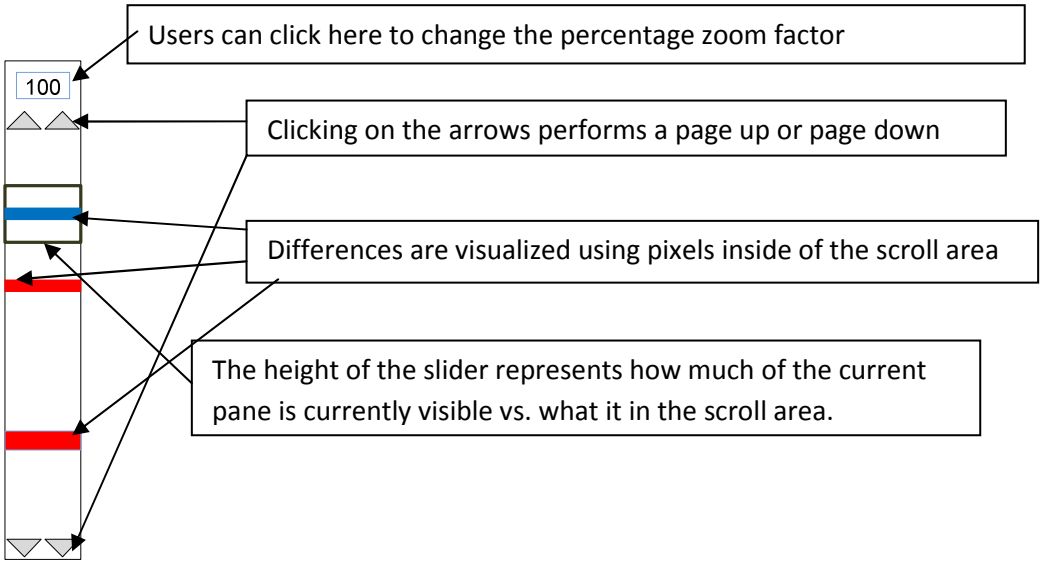

Figure 3 the mini-map-scroller.

The following properties will hold for this widget:

- Content of the pane is assumed to be in line format (such as an equally sized tree)
- Scale can be input via a text box (called the zoom factor)
- Each pixel line of the scroller represents a region of content in the pane. Mapping of the height is done by considering the scroller height, the size of the content and the current zoom level
- Lines are used to indicate areas of interest, red means different blue means missing.
- If the number of lines exceeds the pixel height available to the scroller, then the width of the scroller will also be used. i.e. if the width of the scroller is 10pixels, then 3 dots of red followed by 2 dots of blue would indicate that 30% of this region is different and 20% of the region is missing. (the percentage will be rounded up, so there is no data loss)
- The slider changes height to represent what is currently being shown respective to the slider
- Because of this the slider can become very thin (one pixel high). Clicking (and holding) the left mouse button anywhere on the scroll area will lock the mouse on the slider region (therefore there is no need to try to click a very small slider). While in this mode, the zoom factor can be changed by moving the mouse orthogonally left and right while clicking and holding the left mouse button
- Moving the scroll wheel moves up/down to the next difference.
- Arrows are provided for page up and page down navigation.

Outline View

An outline view provides an over-all picture of the software. It supports navigation down the entire hierarchy.

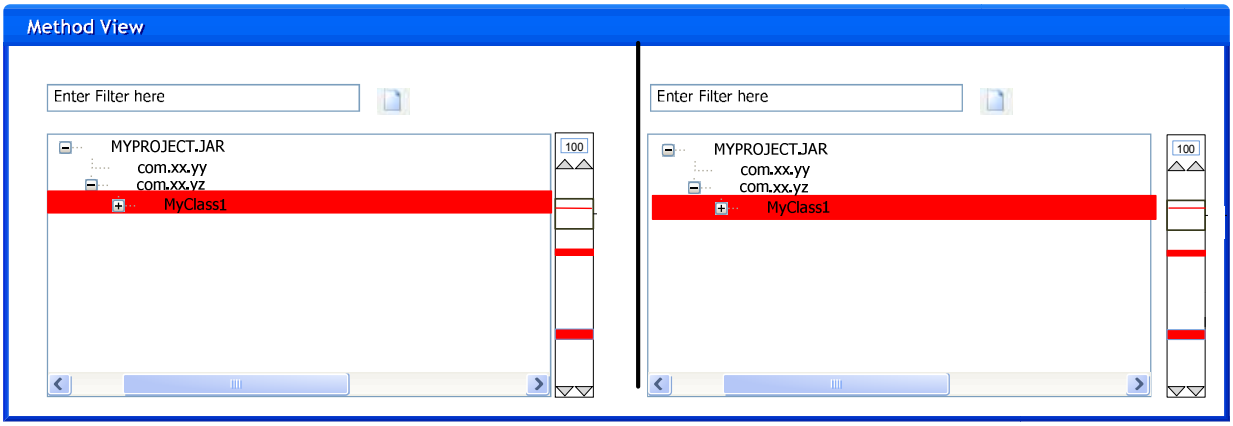

Figure 4 The Outline View. MyClass1 is highlighted because it has different member variables.

Method View

A more specific view tailored to showing the differences between two methods.

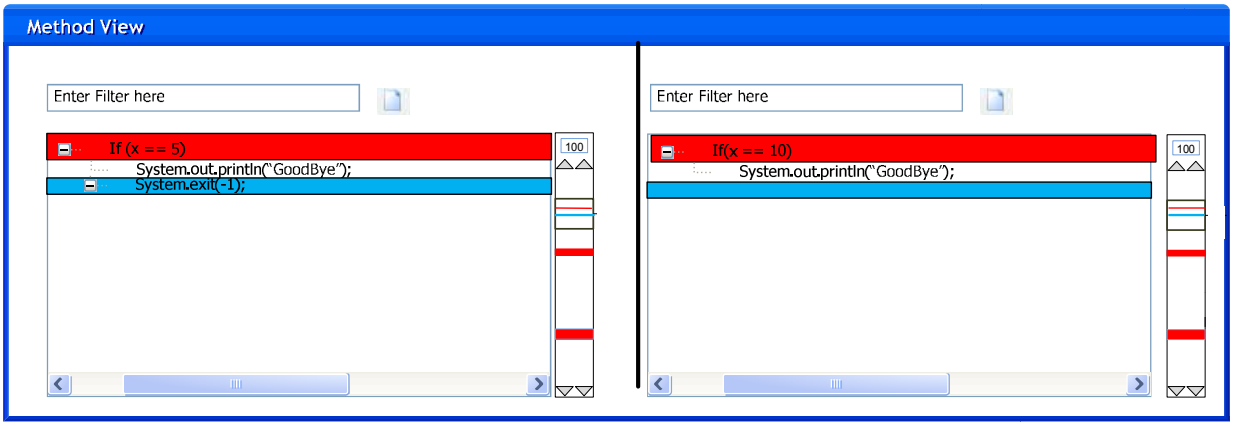

Figure 5 the Method View

Detailed Difference View

This view is linked to the currently active view selection. When the user selects an element in another view, this view updates with content from that selection.

Figure 6 Detailed Difference View

Scenarios of Use

#1) A regression has been discovered and assigned to a developer. The current source code is compared to a previous version where the regression does not exist. The developer is familiar with program so uses the outline view to target suspect areas of changes. The method view is used for more details on differences in the methods and the problematic check-in is discovered.

#2) A user wishes to review his changes before checking them into source control, therefore they wish to see all of the differences prior to checking in the code. The user navigates every difference by using the next difference navigation.

Implementation

I plan to implement this project on Windows using the Eclipse RCP framework and SWT. I have chosen this architecture because I familiar with it. I will investigate using a combination of JAVAML (8) and DiffX (9) [A java library for XML comparison].

Milestones

Unfortunately time constraints will not allow all of these features to be implemented, therefore features will be implemented in a priority basis and, if time allows, additional features will be added. The following milestones will used:

Nov. 7th:

- Method of comparison determined and implemented. Output is generated.

Nov. 15th:

- Output from comparison can be read into some kind of hierarchical object model.
- Work begins on the Mini-map-scroller.

Nov. 21st:

- Mini-map-scroller completed (including a tree widget capable of navigating thousands of nodes).

Nov. 28th:

- Method View – able to visualize a single side of a program.
- Details View – able to see differences in a highlighted manner.

Dec . 5th:

- Method View – complete.
- Next difference navigation fully supported.

Dec . 12th:

- Complete project write-up.
- Add filtering && outline view (if time permits).

References

Works Cited1. *Structural Analysis and Visualization of C++ Code Evolution using Syntax Trees.* **Chevalier, Fanny.** 2007.

2. *TreeJuxtaposer: Scalable Tree Comparison using Focus+Context with Guaranteed Visibility.* **Tamara Munzner, et al.** s.l. : SIGGRAPH 2003, published as ACM Transactions on Graphics 22(3), pages 453--462, 2003.

3. *Treefoil: Visualization and Pair Wise Comparison of File System Trees.* **Shannon Bauman, James Clawson, Josh Cothran, Jeanie Miskelly, Zach Pousman.**

4. *Visual Comparison of Hierarchically Organized Data.* **Wijk2, Danny Holten and Jarke J. van.**

5. *Evolution, CVSscan: Visualization of Code .* **Lucian Voinea, Alex Telea and Jarke J. van Wijk.**

6. *Visualization for Fault Localization.* **James A. Jones, Mary Jean Harrold, and John T. Stasko.**

7. *OrthoZoom Scroller: 1D Multi-Scale Navigation. .* **Fekete., Catherine Appert and Jean-Daniel.** 2006 : SIGCHI 06, pp 21-30.

8. **Badros, Greg J.** JAVA ML : An XML-based Source Code Representation for Java Programs. [Online] http://www.badros.com/greg/JavaML/.

9. DiffX. *A java library for comparing source files.* [Online] http://www.topologi.com/diffx/.