

Lighthouse Eclipse Plugin Evaluation

Chris Dimpfl Ankita Raturi Arthur Valadares Caitie Lustig

December 12, 2010

Abstract

The Lighthouse plugin for Eclipse is an innovative way to display modifications to collaborative software projects. However, it is still a prototype, and has not been rigorously evaluated for flaws in its interface. In order to evaluate Lighthouse, we first performed a heuristic evaluation of its setup and simple interactions, followed by a user evaluation by professional programmers, Lighthouse's intended audience. We found that Lighthouse has issues with its interface, and style of interaction, as well as several collaborative issues. We concluded that Lighthouse, while an interesting tool fails too frequently with too little notification to be relied on in a development environment. Furthermore, its numerous interaction flaws make it unwieldy to manipulate on the fly. However, with some changes, it could be a valuable addition to an Eclipse developer's toolkit.

1 Introduction

In this study, we evaluated Lighthouse, a plugin for the Eclipse IDE with an additional dependency on Subclipse, an Eclipse plugin for handling Subversion repositories. Lighthouse is intended to display real-time changes to projects, by displaying modified methods and variables in classes, in a format that loosely resembles UML diagrams.

Lighthouse had not previously been subjected to user testing, and is still a prototype; because of this, we initially wanted to run user evaluations, to find rough edges with visualization of changes as participants modify code, as well as manipulation of the display and filters. Furthermore, we were asked to evaluate the need for the addition of new features (such as the addition of an indicator that shows who on the team is the 'expert' on a class), as well as the utility of such features.

In evaluating the Lighthouse Eclipse plugin, our initial concerns were with navigability and discoverability of its EmergingDesignView, used for visualizing changes. Additionally, Lighthouse authors voiced concerns that with larger projects, Lighthouse might become too unwieldy, or did not provide a meaningful visualization. While interface discoverability and navigability are important, utility as a tool is vital; As such, we wanted to evaluate new users' interaction with Lighthouse in a collaborative environment.

2 Experiment Planning

In order to effectively evaluate interface issues in user evaluations, each member of our group first performed a heuristic evaluation individually, which was later merged when we discussed issues we'd encountered. The Lighthouse interface is primarily controlled through a group of buttons in the upper right hand corner of the EmergingDesignView. As seen below, these are the highlighter, layout manager, detail of display, authors filter, package filter, modifications filter, active class and dependencies filter, and zoom.



Setup was convoluted, and two members of our group required assistance before they could continue with their evaluation. Issues primarily arose in the Lighthouse preference pane, with Lighthouse either not remembering server information after pressing apply, or difficulty making Lighthouse use a specific user name for the EmergingDesignView.

Other common (and persistent) issues were with simply clicking the buttons, or clicking on the small down arrow next to them. In fact, in the case of the highlighter, author filter, package filter, and zoom buttons, clicking the button never does anything, and the corresponding arrow must instead be clicked for a menu to be displayed. Several of the issues we encountered (primarily bugs with the display of changes) could very easily prevent testing of other features, so we passed our bug reports and heuristic evaluations along to the designers, who made several fixes and minor changes before we began user tests. These changes included fixes for showing changes correctly, completely removing the 'link with editor' button, as well as changing several icons, including removing three layouts from the layout menu, and giving the detail filter menu items meaningful names (which had previously been Mode 1, Mode 2, etc.).

With this in mind, we created a two-phase experiment for user evaluations. We geared phase 1 toward teaching users how to use Lighthouse through manipulating the interface with a few simple tasks, such as finding modifications made to classes. This decision was made due to the technical inclinations of our target audience; learning through doing both gives us further insights into the discoverability of the Lighthouse interface, as well as engages the user so they aren't bored by our test. Phase 2 was initially designed to test Lighthouse's real-time collaborative features with two of us (in order to reduce possible divergence from the intended code changes) with a participant in another room who was led to believe that we were all recruited testers. However, this proved to be impractical because it was impossible to hide our involvement during the recruitment process from participants, so we were forced to change the scenario, and simply asked participants to pretend that we were participants, and collaborate with us over instant messenger if they felt it necessary for collaboration.

Because of issues with the pilot tests (discussed in detail later), we modified the experiment slightly for our user evaluations. Phase 1 gave us useful feedback, with no noticeable issues. However, phase 2 had several issues, in that tasks were confusing or ambiguous, , and

while we had written up code that we expected to write, we had not written explicit code for the participant, so our changes did not merge in the ways we expected. We overhauled the tasks, and rewrote all of the code that would be changed over the course of the experiment, which proved to work well in subsequent experiments, with a few minor issues.

3 Experiment Descriptions: Phase 1

Pilot 1 (P1)

While becoming familiar with the user interface (task 1), P1 struggled with a number of user interface related problems that we quickly found were consistent with problems faced by our other testers. P1 could not determine what purpose the highlight button had—the highlighting effects are subtle enough that a new user would not necessarily notice if the highlighting had been turned on or off, and described the highlight button as useless. P1 also tried to click all of the buttons—some of the buttons have no functionality if they are simply clicked instead of having their submenus accessed. In particular, the user found that clicking on the zoom button did nothing. The user found that he could not determine the meaning of the change, addition, and subtraction symbols in the class diagrams. However, the user found the rest of the user interface to be relatively easy to understand.

Filtering to show only the active class and its dependencies simply shows nothing if there is no active class in Eclipse. This is the expected behavior, but if the user does not understand what the button does because it is not clear to them, then this is very confusing. P1 stated that the relational arrows can be difficult to follow and suggested that there could be an alphabetical layout option in order to locate classes easier. The user also said that the filter modifications icon looked like it was for adding something because of the plus in the icon, which confused the user. As for the filters themselves, the user felt that the package and modified filters were useful, but could not find the user filter to be helpful.

Pilot 2 (P2)

P2 also tested the interface by only clicking on icons instead of exploring the submenu for each icon. As the user stated, a button shouldn't be shown if it's not going to do anything—it should just drop down. In particular, this pilot user also felt that the highlight button did not do anything. This user also noted that clicking on the zoom icon did not have any effect, and felt that a zoom function was useless anyway and that a search function would be much more effective. It was noted that the only show modifications option was unclear since the interface did not explain what baseline was being used. For all the user knew, the baseline could have been from when the user opened Eclipse, or it could have been from the last svn checkin, or any other number of possibilities. Regardless, the user did not feel like the only show modifications filter was particularly useful.

P2 noted that he was not sure what was the meaning of the position of the classes in the grid layout. Additionally, the user felt that the modification buttons symbol did not make

sense and might be better represented as a delta symbol. Because there were not many modifications to the project, the user noted that the first two class visualization modes (only class name and class/authors name) appeared to have the same functionality since there were no author names to display.

Subject 1 (S1)

Like other users, S1 felt that the highlighting option should not exist, but simply be always on. This user had some difficulty navigating the graph; he tried to move around the diagram by clicking on the map in the righthand corner and also by trying to use the mouse to zoom out (as is the standard way to zoom out in many applications). Like other subjects, he was confused by the meaning of the delta symbol on the modified classes.

This user was much more enthusiastic about the project than the two pilot users. He stated that he really liked the full view filter, the show only the active class filter, and the grid layout. He felt that the tooltip for the filter by users option was unclear. Like nearly all the subjects, he felt that the all fields and methods shown class visualization mode was by far the most useful visualization mode. S1 stated that the show active class only filter and the show modifications only filter were the most important filter. Some additional functionality suggested was that the grid layout should preserve some of the users arrangements to the layout. Which is to say, if the user moved a class around the diagram and then clicked grid layout, instead of resetting the layout, it should simply snap the classes to the grid. He also stated that it would be cool if I could click on what someone else was changing and see a diff.

Subject 2 (S2)

The user felt that the package filter was not meaningful, but he did find that show modifications only filter was particularly useful. Additionally, he felt that the zoom and diagram layout options were useful. He found the tooltips self explanatory. The user stated that all of the class visualizations modes would be useful for various situations. This user was particularly quiet and did not give a lot of verbal feedback, so unfortunately there is not much more that we can say about this particular users impressions.

Subject 3 (S3)

S3 had the same problem that our other users did with trying to click on the zoom icon as well as being confused about the highlight class option and the class/authors name mode. As with other users, it was unclear what the delta symbol meant or what modified really means in this context. The user pointed out that the filter by modifications icon, means java perspective in Eclipse itself, and so he expected it to have something to do with Java. He found that the icon was not self-explanatory and that he had to check the tooltip to understand it.

The user suggested that the icons should be ordered from left to right in order of importance—he felt that the show only modifications button was useful, but because it was to the far right, it was easily forgotten. The user also felt that it would be nice to have more view options besides just the grid layout or to have user customizable layouts. He also suggested that the icon for the class visualization mode should instead be Eclipses class symbol icon instead of the icon that the developers chose to use.

Subject 4 (S4)

This user was very quiet and did not offer a lot of verbal feedback. One thing that he did state was that the highlight icon was not that intuitive and that he could not determine the meaning of the show active class and zoom icons. He felt that if he clicked an icon twice, it should revert back to its default value. Another criticism that he made was that he felt that based on the icon, he expected the layout button would create a hierarchical layout. He felt that the grid layout should have conveyed some sort of hierarchical or organizational meaning. Yet again, this user struggled with the meaning of the plus, minus, and delta symbols in the class diagram.

Some additional functionality that the user suggested was that there should be a distinction in the class diagram to denote which items were abstract class or interfaces. Overall, he felt that the whole class diagram was confusing—in particular the meaning of the plus, minus, and delta symbols, as well as the meaning of the strikethrough for deleted variables or methods.

Subject 5 (S5)

S5 was also confused about what the plus and minus symbols meant, and guessed that perhaps they had something to do with UML conventions. The user stated that he could not see why anyone would not want to use highlighting. Additionally, he stated that it would be nice to be able to highlight related classes (Currently, Lighthouse can highlight the relationship lines between classes but does not actually highlight the related classes themselves).

He also felt that having the grid layout as the only option did not make a lot of sense and that the organization of the grid layout seemed to be random. Instead, it would be nice to have a hierarchical layout. The user remarked that the zoom icon looked like it was the icon for a search function and stated that he believed that there should be a search function. S5 stated that instead of using the submenu to select a zoom level, it would be nice to be able to use a slider to select the zoom level, which would give more control of the granularity of the zoom. He was confused about the meaning of the delta icon.

4 Experiment Descriptions: Phase 2

Pilot 1 (P1)

The first pilot used in the experiment was a graduate student with some professional development experience. His work in Java was primarily Web applications, and he did not have much experience with collaborating on software projects.

Initially, the subject was very confused about the tasks. He did not understand the purpose of Lighthouse, and challenged the specifications of the task that was given to him. At the point where the conflict of code is introduced, he noticed that C1 had already created the Banks class, but decided it was taking too long, and rather than communicating with C1, he proceeded to create his own Banks, while C1 deliberately stalled, waiting for a request from P1.

Part of the reason why P1 was confused was that the tasks were confusing as initially designed. This was compounded by the fact that C1 had not finished his own coding tasks before the experiment started, causing him to be confused about both his and the participants tasks, and when they were supposed to conflict. This led to confusion not only from P1, but also from C1. Furthermore, the subject was unfamiliar with common banking concepts, such as overdrafts, daily withdrawal limits and withdrawal fees. He was instructed about these concepts through chatting with C1.

Throughout the experiment, P1 was frustrated with bugs and confused by the interface. He did not understand why he couldn't see other collaborators code. He went through the whole experiment without any collaboration with C1 and C2, except for questions when trying to understand the tasks. Finally, he expressed that he would not use Lighthouse again if given the choice.

Pilot 2 (P2)

The second pilot subject was also a graduate student with little experience in the industry, but wider experience in academia. He had little collaborative work experience and his experience in Java was limited to academic applications.

This subject seemed more comfortable with the tasks at hand than P1. While he did ask C1 for guidance and confirmation of his work several times, he did not question the tasks themselves. When his own work intersected C1's, Lighthouse proved useful in showing him that C1 was working with the class he needed and asked C1 to commit it to continue his work. Later, during task 4, he successfully noticed that he needed the code from C2, and requested it.

While C1 felt that he understood the problems of his own coding in the first pilot test, new problems arose during this experiment. Once again, he confused the collaborator and subject. P2 also demonstrated a great deal of frustration with bugs and idiosyncrasies during the experiment, and was further annoyed that he could not see the code that was showing as modified by other users in Lighthouse.

C1's initial approach to the problem was more complex than what was suggested by the

task descriptions, but throughout the experiment P2 understood the purpose of experiment. He did question some design decisions, which were promptly explained over IM that the tasks were meant to be simple as Lighthouse was being tested, rather than coding ability. Finally, P2 had some privacy concerns related to his changes being displayed to other developers, and was worried about coding well so he wouldn't look bad to other developers on the team.

Subject 1 (S1)

This subject worked as a programmer during school, and recently graduated with his Masters. He now works as a software developer, frequently in a collaborative environment. While Java is not his language of choice, he was comfortable enough to work with it, but his experience was limited.

S1 quickly understood the tasks that were given to him, being the fastest of all the subjects and pilots to go through all the tasks. His only troubles were with Java itself. He asked C2 about not having Banks (did not check Lighthouse to see who was doing that job, as hinted by the tasks), but eventually asked C1 for his code. When the second conflict came up, he collaborated with C2, using Lighthouse effectively. Lighthouse briefly broke during one of the tasks, but C2 told him how to resynchronize, which worked and work continued.

The user enjoyed using Lighthouse, and while recognizing it was still buggy, he felt it would be useful for him and for others. He felt that while sharing the changes between developers was not too much of an issue, auditing from managers and superiors on seeing this information could be a very real issue. This experiment was largely bug-free, but despite changes to the code base and experiment, the subject commented on several design flaws of the new code, and the simplistic way it was coded.

This subject had suggested Lighthouse include a way to reset modifications displayed to other participants, so that dirty work that was done while the developer is figuring out how to write a solution is shown in a clean way to other developers. This was requested because of concern for showing messy code to coworkers, and because of concern for showing other developers who may be working on the same code confusing changes. He was also concerned that Lighthouse may not be able to provide a meaningful visualization with very large projects, even with filters. As was common with other study participants, he also wanted the ability to see code changed by other users, so that not only the UML of the modifications are shown, but also the code itself.

Subject 2 (S2)

Subject 2 was a graduate student with previous industry experience. He worked with Java at his previous job, and has had several years of collaborative development experience as well.

During the experiment, he did not have too much difficulty understanding the tasks. He did request confirmation for some of his work throughout the experiment.

During task 1, Lighthouse stopped working for a short time, until one of the observers resynchronized it with the server. However, the user was not using Lighthouse during this

time, and it was only discovered because the observer was actively chatting with C1. During task 3, he successfully noticed and asked C1 to commit the Banks class, as expected. However, Lighthouse became unsynchronized again at this point. We were not able to get it to correctly display changes again, and did not finish task 4.

Subject 3 (S3)

Subject 3 was a graduate student with many years of industry experience, during many of which were with Java. In his work, collaborative development was also very common.

During the experiment, this subject questioned a design choice, when he expected some code to be there, to which C1 replied that it would be future work. The subject later contacted C1 asking where the Banks class was (instead of realizing by using Lighthouse that C1 was supposed to have it). During task 4, the subject successfully contacted C2, and S3 and C2 were able to successfully prevent merge conflicts.

Subject 4 (S4)

Subject 4 was a graduate student with a few years of industry experience. He works with Java at his job, as well as in his academic tasks. He has some collaborative experience from his previous job.

When S4 finished his first task, he contacted C1 to tell him he finished before he proceeded to task 2 (the task mentioned someone would need his code eventually). He did not realize that C1 would know about his changes and ask for them later.

This subject questioned C1 about the need to create Banks class. When C1 asked him to double check his task, the subject mentioned he could see a Banks class being created by C1, but he did not have access to the code. C1 explained that the code was not yet committed, and proceeded to commit it. He never contacted C2, and task 4 conflicted badly.

Subject 5 (S5)

Subject 5 was a recent graduate and employed as a developer. His work is highly collaborative and deals mainly with Java. He primarily uses NetBeans as his IDE, rather than Eclipse, and had little experience with Eclipse.

S5 was critical of our design from the start, but was constructive in his opinions. When he presented a possible flaw or misunderstanding in the code, he presented his understanding of the system and his solution to the problem. S5 caught details of implementation that were not correct, but yet no other subjects had noticed. For instance, the UML for one of the tasks would not match perfectly if merged with the Banks class C1 was using, so he proposed a way to merge it.

When the task 3 conflict was presented, he quickly noticed C1 was working on it, and requested the Banks class be committed. After the work was committed, he realized that C2's work would also be needed, and tried to initiate a group chat to improve communication.

However, due to an IM bug, group chat failed and communication remained one to one. During task 4, the subject asked C2 for his work, and finished his task without conflicts.

5 Analysis

5.1 Heuristics Analysis

The heuristics analysis performed by the team yielded many suggestions and criticisms of the first version of Lighthouse we used (which was later updated to improve upon some of these criticisms). Using Nielsen's usability heuristics, here are the usability violations discovered:

- **Icons were not good metaphors for the functionality they are supposed to cover.** Most of the icons that were used were borrowed from Eclipse's icons, and had no relation with the new functionality they were representing. This was improved upon in the second version of Lighthouse.
- **Tooltips display slowly over icons in the upper right of the Lighthouse screen.** Because icons aren't good metaphors, it was tedious for the user to get help in understanding the functionalities the buttons provide.
- **Difficult to view and click on links between classes.** The lines are very thin, hard to see, and very difficult to be selected. It is unclear that selecting them at all is useful for the visualization, or that they should be selectable at all.
- **There are no shortcuts available for experienced users.** The input to Lighthouse's `EmergingDesignView` is mouse-only, which is frustrating and slow for experienced users.
- **The interface fails to explain the purpose of certain features.** The highlighter button doesn't seem to be useful. Highlighting relationships (links) between classes does not seem to have any purpose and can be done independently if the highlight relationship is on or off. Disabling highlighting of links or classes doesn't seem to have a purpose.
- **Highlighting a group of classes requires knowledge from user, with no help from interface.** To highlight several classes and group them, the user must hold the control button. There is no information on screen to help the user understand this functionality and how to use it. Furthermore, shift clicking should also be supported to help discoverability and navigation.
- **When navigating between classes, layouts aren't remembered.** Clicking on a class reverts the view to the spring layout.

- **All check-box filter menu items start unchecked, yet their functionality acts as if they were all checked.** For instance, the package button and the participants button start with no option checked, but all participants' changes and all packages are shown. This violates the user's expectations that only checked items will be shown.
- **The "authors" button fails to provide feedback of non-existing authors.** When no authors exist, the button exists, is clickable, but nothing happens on the interface. Furthermore, clicking the authors button never gives any feedback, and only brings up a menu when the small menu button is clicked.
- **Setting up Lighthouse is a complicated and fragile process.** During the experiment, a lengthy undocumented process had to be followed in order to assure that Lighthouse would not fail during experiments. If steps were skipped and Eclipse work areas were reused, old changes would pop on the users screen when they attempted to commit or update the svn.
- **When the Lighthouse screen size is adjusted, the content does not follow it.** It requires the user to request a new grid layout in order to reset the classes in their optimum screen size positions.
- **Highlighting a class is barely noticeable.** There is not enough difference between the highlighted class and the ones that aren't to draw user's attention. A few suggestions are changing the entire color of the box, drawing a box around selection and changing the background, or changing brightness and saturation for the class box.
- **Zoom icon is positioned too close to the minimize screen button.** It is very easy to miss the zoom icon and unintentionally click the minimize button.
- **Zoom icon has a confusing metaphor graphic.** The magnifier glass has two common meanings, search or zoom. Applications usually put a plus sign on the icon to express it is for zooming, rather than a search button.
- **"Only Class Name" layout is default and does not show any changes to classes.** There is no visual indication that the interface is on mode 1, yet in this mode no changes are shown. User can mistake it for lighthouse not working.
- **Interface fails to provide information about its current state.** By looking at the lighthouse interface, the user does not know what package filters, author filters, grid layout or mode lighthouse is operating in.
- **Layout button: Unattached classes are placed randomly.** There is no consistency where the classes with no relationships are placed through the many different types of layouts.
- **Lack of consistency on clicking buttons.** Some buttons will react to clicking by cycling options; others will simply do nothing, yet are still clickable (they should open the corresponding menu, as in the case of the Author and Package button).

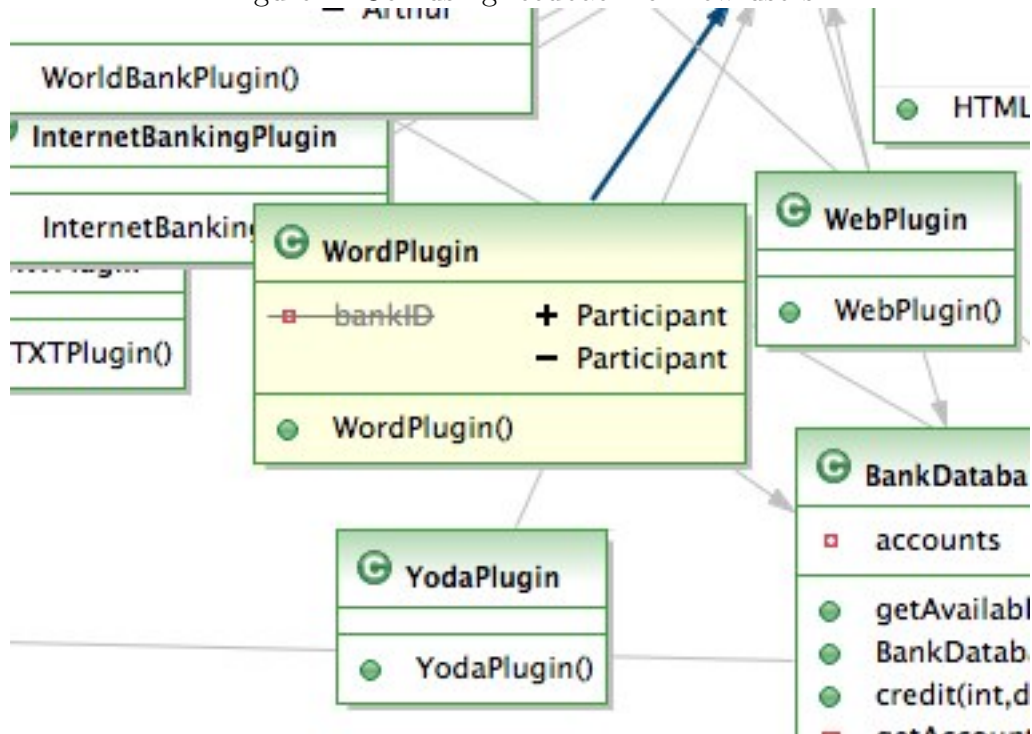
- **Server indicator icon at the bottom left has no apparent connection to Lighthouse at all.** The button is isolated on Eclipse's window, when it should be grouped with the Lighthouse window, in order for the user to understand they are part of the same application.
- **Icons are randomly placed in the graphical menu.** They should be grouped, using either whitespace or lines. A suggestion would be by functionality:
 - Filters: Modes, layout, packages, "show only modifications" and authors.
 - View: Zoom, highlighting, "show only active class and dependencies"
- **"Import project" is confusing in its context.** When using client software, importing can reasonably be understood as bringing data from the server to the client. In Lighthouse, it means taking the project to the Lighthouse server. Suggestion to change it to "Upload project to server".
- **No undo/redo offered in the visualization screen of Lighthouse.** If a user clicks on a button or changes a filter by mistakes, his original view of the classes is lost forever.
- **The word "Username" is used for two different meanings.** In the Lighthouse options, username is used to refer to the MySQL database username, and to the username that will be the identifier for changes in the visualization of classes. A suggestion would be changing the visualization name to "Nickname" or "Display Name".
- **Errors are silent and provide no useful feedback.** For instance, if the user attempts to synchronize to Lighthouse without uploading the project first, no feedback is given. Every time Lighthouse failed and stopped synchronizing, it did so silently, with no error messages or status change. Lighthouse also takes a very long time to time out from the server when attempting to import project for the first time.
- **Filters should have select all/select none.** If there are too many authors or packages, an option to select all and deselect all should be available so the user won't have to check or uncheck one by one.
- **Provide a search function.** This way users can search for a specific class without having to go through many filters to see it.
- **Layout options should be saved and loaded.** Provide user with the opportunity to save the filters and layout options, so he can apply it to easily again to any project.
- **No way to hide changes that were done and undone.** Developers face the problems through many different angles until they arrive at a good solution, and they may not wish to show their draft, as it will be messy and potentially embarrassing. Allow developers to reset to only the final modifications (compared to the original).

- **When a class is deleted, it disappears from the interface.** If a class is deleted, there is no record of it having ever existed. There should be a history that indicates that the class was deleted and by whom.
- **Filter by relationships only allows one degree of separation.** It would be useful functionality if user could specify how many degrees of separation he wishes to see from a particular class.

5.2 User Evaluations: Phase 1

In phase 1, we simply had the users try to become familiar with the interface and learned what new users with only a very basic understanding of the ideas behind Lighthouse would experience. As it stands, new Lighthouse users will probably find themselves confused about the plus, minus, and delta symbols in the diagram and will feel uncertain how to interpret the modifications that their colleagues make. This will result in users having to contact their colleagues for information or simply guessing what the symbols mean. In our phase 1, the users were not in touch with their collaborators and so they simply had to make educated guesses. Their conclusions in general were correct, but the degree of uncertainty that they had made completing tasks difficult. This was particularly true in task 3, where users were asked to determine what C1 had modified. This was difficult for users, because when a method had been both added and then deleted, both the plus and the minus appeared.

Figure 1: Confusing feedback for new users



Additionally, users were frustrated that Lighthouse did not obey conventions that they were familiar with. For example, the first pilot, and third and fifth user all had issues with Lighthouse icons that they were already familiar with them, but had a different meaning in Lighthouse. Many icons did not behave the way they expected them too—users expected a certain functionality when they clicked on icons and some icons did not have any functionality when clicked. Furthermore, users were unfamiliar with the filtering mechanisms presented and expected there to be a search functionality.

Some of the icons seemed to give the users options to turn on or off functionality that should simply always be "on". For example, all users felt that they would never want to turn off the highlighting functionality. Almost all users seemed to prefer the diagram view with the highest level of detail. All users liked the grid functionality although they felt frustrated that the grid layout did not seem to convey any hierarchical meaning or organizational structure as well as not conveying any changes that they had made to the layout themselves.

5.3 User Evaluations: Phase 2

From analyzing the data obtained in the pilot and user tests, many problems with Lighthouse were confirmed from our heuristic analysis, and new issues were discovered.

The persistent issue present in every experiment was that users were confused and frustrated at not being able to see code showing as changed in the Lighthouse interface. Although each user was explicitly informed that the interface only shows changes, and not the code itself, every user expected to be able to see the modified code. This is perhaps the case because users can click on method and variable names to see his own modified code. As such, the user expects that double clicking a method/variable or new class created by a collaborator should also be available for viewing. HCI has guidelines that items that are grouped together makes the user believe that they have similar functionality as well, but perhaps coloring available changes and unavailable changes differently would solve this issue, and make it easier to see at a glance what other participants have changed.

In analyzing the usage of Lighthouse itself, it is apparent that most users felt that it was not relevant to the completion of the tasks themselves. Indeed, Lighthouse does not provide a tool that is essential to the development process, but rather helps raise attention to changes that might affect your work. We believe that, just as users have trouble with a new functionality on a new version of a well known application, users need time to adapt to new functionalities that are not essential, yet will tend to be useful in the feature. The user tests showed how users resist using new technologies and changing the way they work. Discarding the frustration on bugs, most users said that even if the software works correctly, they would not be interested in using it. It remains to be proved if, in the long run, users will accept the new tool and use it out of their own interest.

The active use of Lighthouse seemed to be directly related to the subject's collaborative work experience. The two subjects (S1 and S5) who are currently working collaboratively in companies were not only excited about the functionalities, but also felt it was very interesting and it had lot of potential as a collaborative tool. As for P1, when he was questioned about not collaborating, and simply recreating C1's work locally (without talking to C1 at all), he

stated that he didn't understand why what he did was wrong. It is our belief that there are a number of social considerations involved in disturbing other colleagues to ask them to collaborate, from simply not wanting to disturb other coworkers, to not wanting to look bad in front of colleagues. So while Lighthouse helps developers to be more in touch with other collaborators on a project, it's possible it that it creates a great deal of tension while doing so.

One concern common to many subjects was privacy. Just as a developer is nervous about committing code that might contain bugs, or not compile at all, developers using Lighthouse may be hesitant to share what they're working on because of fears that others will interpret them as incompetent. While this may partly be a side effect of users of our study being part of a study with an unfamiliar code base, it cannot be completely discounted. Because of these social factors, a more in-depth evaluation on the social impacts of Lighthouse, and how to mitigate harmful influences may be useful. Our first subject also voiced concern over management using Lighthouse to evaluate developers (similar to requiring developers fix some number of bugs every day), which, although not Lighthouse' primary use, may be worth considering.

While the functionality that Lighthouse provides is impressive, it is tedious and time consuming to modify the visualization for different tasks. Furthermore, users were somewhat overwhelmed with this new tool that they had little to say about the usability of buttons and layouts. Compound this with Lighthouse's occasional glitches during experiments (which brought the second user test to a halt twice). While these short, one time user tests were useful and provided good insights into some of the issues surrounding Lighthouse, as well as some of the interface problems that exist, most tools such as this tend to be used for a longer period of time. As such, a better usability test for Lighthouse should be numerous samplings over weeks or months of usage by a team.

In general, users and experimenters felt that bugs were extremely harmful. When a bug occurred, it isn't clear that an error has occurred, or to what extent it has occurred. This results in a great deal of frustration and confusion on the part of all collaborators, and because there is no simple way for Lighthouse to recover, it may be necessary to have to go through the lengthy process of setting Lighthouse up from scratch. While we did not resort to setting up Lighthouse from scratch during an experiment, several synchronization errors occurred over the course of our experiments, very negatively affecting feedback as to the overall usability of Lighthouse.

6 Recommendations

During the course of this evaluation, certain recurring issues have been identified over the course of the heuristic evaluations, as well as user studies. In this section, we propose a number of recommendations based on these results.

6.1 Interface Elements

In phase 1 of the user evaluations, a detailed exploration of all the interface elements was conducted. While the intuitiveness of the icons and other interface elements was relatively high, it was also dependant on how familiar with Eclipse users were. The more the user knew about Eclipse, the more knowledge they had of the style of icons and elements used, making the task of discovering the icon's use simpler, with several exceptions.

6.1.1 Icons (Current Name → Suggested Name)


-  Highlight elements → Highlight

While the icon itself is meaningful and intuitive, it was not clear that clicking on the icon itself did nothing, and the menu next to it must instead be clicked.

In addition, the option to highlight in different colors according to user specified options may be convenient. The colors currently used to highlight classes are also not highly visible and should be changed to more pronounced colours.

-  Diagram layout

Several users suggested that the grid layout automatically snap into place if selected to do so. For example, when the grid layout is selected, but then enables icon 7 (active class and dependencies), the layout reverts to a radial layout with the active class in the center; this behavior is fine. However, when this filter is turned off again, the radial layout remains, with the rest of the class diagram simply reappearing on top. Therefore, it is suggested that when other filters are turned off, the diagram should snap back to the grid layout for the sake of clarity. There is also no apparent rule as to how the grid is structured - participants have suggested organizing the grid alphabetically, or minimizing overlapping lines, and should be investigated as potential alternative behaviors.

-  Class visualization modes → Diagram Detail





Suggested icon: Same icon, possibly with a 'C' eclipse class symbol superimposed in the top left corner.

Menu options:

- Only Class name
- Class/Author's name → Class or Participant names
- Modified attributes and methods
- All attributes and methods

User response for this icon has been excellent overall. There have been some issues with the lack of visible difference between "Only Class name" and "Class/Author's name".

The terminology Author has also proven to be ambiguous (see section Terminology) and the lack of feedback from the system when changing modes, especially in cases where there is no visible change is confusing (see section Behaviours; Disabling and Feedback)

-  Filter by users → Participant filter
-  Filter by package → Package filter
-  Filter by modifications → Modified classes filter
-  Show the active class and its dependencies → Active class & dependencies filter

There has been some confusion as to actual functionality, and what users initially believe it to be. There is frequently a delay before users realize this filter focuses the view on the class open in the editor, and most expect it to focus the view on the class currently selected in the diagram itself. Therefore, it may be useful to split this icon into two: 1. Show selected class & dependencies 2. Show editor class & dependencies

-  Zoom

Users would like the clicking of this button to actually alter the zoom level. It has also been suggested that the diagram have a +.....- bar, possibly located near the overview map in the bottom right corner, in the style of Google Maps. This is advisable as it would eliminate this potentially confusing icon.

6.1.2 Symbols: plus, minus, and delta

While the first two are used to identify whether people are adding or removing elements, users had difficulty associating the delta symbol with modifications. It is therefore recommended, that color coding be used in addition to the symbols, using green for added, red for removed, and orange for modified. Once explained, the users understood appreciated the logic behind the delta symbol, so it is suggested that the symbol be retained.

6.1.3 Terminology

"Authors"; In Java projects, there is "author metadata" available in the classes themselves via the @author field in the comments of a class. This may lead users familiar with Java to assume that when the term authors is used in Lighthouse, that a reference is being made to this metadata. It is therefore suggested that all instances of the term "Author" be replaced with "Participant" to make a clear distinction between participants within Lighthouse, and the Java author metadata.

6.2 General Behavior

6.2.1 Disabling Features

One major point throughout the study was the lack of feedback when changing filters, particularly in the case of the class detail button. For example, when participants have not made any changes, changing the filter to "Class/Author's name" from the "Only Class names" view appears to do nothing. One suggested change therefore is that if a particular option is not going to result in visual changes then it should be disabled so that it is not selectable. Alternatively, text inside the Lighthouse window with mode information could provide useful.

6.2.2 Snap to default

As mentioned with the grid layout, when disabling a feature previously enabled users generally expect the view to return to the original state. This should be reflected when enabling and disabling various filters and layouts within the Lighthouse interface.

6.2.3 Clicking

Currently, in order to open a class in the editor, one must click on the class name itself. According to Fitt's law, such targets should not be so small. In addition, throughout the user studies, users would double click anywhere in the class box itself expecting it to open, and were disappointed to find that they had to click on the text. As such, we propose that the interaction with the class diagram be as follows: 1. Single click highlights that class and its relationships. 2. Double click anywhere withing a class, opens the class. If the class is already open, it makes it the active class in the editor. 3. If a specific property or method of the class is double clicked, open the class and jump to that property; if it is already open, simply jump to that property.

6.2.4 Feedback

The Lighthouse interface is currently very conservative in the feedback it provides users. Several users had difficulty simply because they were not aware of the current system state, or changes being made by collaborators. The "filter by modifications" and "show active class" features were very popular, as they simply and easily allowed the user to know what was actually being modified. The following alterations may improve global feedback:

- Change of State by Participant

Lighthouse must inform the user as to currently selected filters. One solution may be to include state information at the top of the emerging design view pane, with information such as "Grid View; Showing Modifications Only" and so on. This would give the user constant feedback as to what filters and options are currently in action.

- Change of class diagram by others

One of the main purposes of Lighthouse was to create an awareness amongst the users as to what is being modified in existing code. When another participant alters the code, currently, the class diagram quietly updates. It is therefore suggested that perhaps there be a "Verbose Mode" toggle. When enabled, the class diagram may highlight new changes in a relevant color (red/green/orange), or even integration into software such as Growl, Snarl, or NotifyOSD. It is possible that levels of verbosity be useful, rather than "on/off" as well.

6.2.5 Constant Synchronization, and Histories

Although Lighthouse is meant to be real time, it sometimes requires the user to manually synchronize with the database. No user ever found the synchronize button on their own, and when Lighthouse issues forced us to use it, users found it cumbersome to have to manually do so. Therefore, it is suggested that perhaps the system should constantly synchronize with the database at a specified regular interval. Furthermore, maintaining synchronization after a user reconnects after a network failure is crucial.

In addition, there should also be a history, or log of changes to the real time diagrams. This is important because it allows the users to see recent changes if they stopped paying attention, or recently returned to work, and keep themselves up to speed with what has recently changed.

6.2.6 Repository state information

Currently, there is a major disconnect between what the users see in the real time Lighthouse diagrams and what has been checked in to the repositories. All users mentioned a strong desire to be able to see code that had not yet been checked in. A more noticeable indicator of checked-in code may help alleviate this issue, although simply relying on users discovering the distinction through use of Lighthouse. It may also be useful for the emerging design view pane to label the diagram according to the repository revision it is displaying, and the currently available revision on the SVN

6.3 Desired Features

6.3.1 Potential Layout Options

- **Save Visual Layout** In addition to returning to defaults, users have also indicated a desire to be able to save specific layouts. It is recommended then that the layout view should allow users to save a specific layout as the default layout for that project, as well as being able to save user customized layout views.
- **Business Logic**

While this feature was only mentioned during a single user study, we feel that the rationale is such that it would make an incredibly powerful addition to the Light-

house layout repertoire. We therefore recommend the following mechanism to enable a "Business Logic" layout.

- User defined either rows or columns that correspond to options such as a 3 or 4 tier architecture.
 - These sections would then feed off the class metadata to establish what class belongs to what tier, and perhaps even identify the number of tiers automatically via this metadata.
 - Allow users to move classes between tiers and potentially alter the metadata through this view.
 - However, as this option could potentially be messy, it is recommended that a thorough investigation of the issues involved be performed first.
- Radial, Tree Layout The developers had shown us a radial layout in a previous version of Lighthouse, and throughout the user studies, it seemed as though users were interested in viewing the class diagram in these layouts when appropriate. It may be useful to retain these as layout options.
 - User Customizable As previously mentioned, Users have displayed an interest in being able to save specific layouts. They would also like to have more control over the layouts by being able to simply move classes around into a view that suits them and save this as a user defined layout.

6.3.2 Search

Often, the Zoom icon was mistaken for a search icon. In addition, when asked to locate a particular class, all users tend to look through the Eclipse package manager and then use the "active class" filter to locate it in the diagram. A simple solution would be for the Lighthouse interface to provide a search box to allow user to simply filter the diagram by class name, locate and focus the diagram onto a specific class. This could also be extended to include searching by method and variable names.

6.3.3 Hover Information

Another feature brought up by several users was the potential to display detailed information by mousing over or clicking classes in less detailed modes (such as "Only class names"). This could potentially be used to display detailed information for relevant classes, and less detailed information for peripheral classes without losing the level of detail required for important classes.

6.3.4 Customization

The lack of user control over the diagram views and general Lighthouse functionality was something that also came up during these studies. The developers should keep in mind

that the ability to define custom filters, layouts, save views, and remember views should be provided to ensure that the user always has control over the interaction.

6.3.5 Help and Documentation

There is a complete lack of help and documentation throughout the Lighthouse interface. There should be a wiki, or other online documentation containing information regarding all the functionality available, as well as a detailed installation and setup guide. In addition, within the interface, there is potential for a simple "?" help button to allow users to discover out what different icons are, where functions are located, or to interact with some help documentation.

7 Conclusions

The Lighthouse collaboration and awareness tool has proven its potential value over the course of the evaluations carried out by our team. Even ignoring current functionality bugs, the overall impression from users was that the tool was that these issues hindered use, but that it had potential for improvement. In addition to fixing the usability issues described in this report, in order for Lighthouse to be adopted, it would require a bit more than just polish.

All of the users felt the second monitor for the tool was appropriate, although users were concerned with how Lighthouse will scale for larger projects. To address this issue, a more customizable mix of layouts, and filters could help Lighthouse scale better, as well as improve clarity and usability. This would hopefully remedy issues that exist because of the variety of interaction styles that different developers display.

There were some privacy concerns regarding the use of Lighthouse in a real development environment. There is the ever present issue of managers using such a tool to monitor developers as well as concerns that new developers may be intimidated with having their work displayed with such high visibility. However, as developers themselves, prospective users of Lighthouse appreciate being able to improve awareness of what others are doing, as well as making their work visible to other users.

The lack of feedback and notification from Lighthouse detracts greatly from its usefulness. As an awareness tool, Lighthouse is only partially successful in this regard. However, we feel strongly that the recommendations provided should be implemented in order to augment the visibility of participants and allow for Lighthouse to become a more efficient and effective awareness tool.

All of the users in these studies had from two to ten years of Java experience, and all but two had used Eclipse as their primary Java IDE. Furthermore, they all had at least some experience with collaborative development. With such a population in both the pilot and user studies, we feel that the results and recommendations reflected in this report are both appropriate and useful. However, we would also recommend longer term usability testing for Lighthouse in a real development environment once the functionality bugs have been ironed

out, to get an idea of how exactly users would interact with such a tool in the real world. In order for Lighthouse to meet the needs of a development environment, it must be stable, as well as have greater customizability to suit the needs of developers. We hope that these conclusions can aid the developers in improving the Lighthouse interface, making it a more effective and usable collaboration and awareness tool.

8 Appendix

Figure 2: Participant statistics

	User	Gender	Age	Java Exp	Eclipse Exp	Collab Dev Exp
Pilot Studies	1	male	18-25	5-10	2-5	2-5
	2	male	26-40	5-10	>5	2-5
User Studies	3	male	18-25	2-5	2-5	>5
	4	male	18-25	2-5	0-1	2-5
	5	male	26-40	5-10	>5	>5
	6	male	18-25	2-5	not primary	0-1
	7	male	18-25	2-5	not primary	0-1

Figure 3: A generic Lighthouse interface with no filters

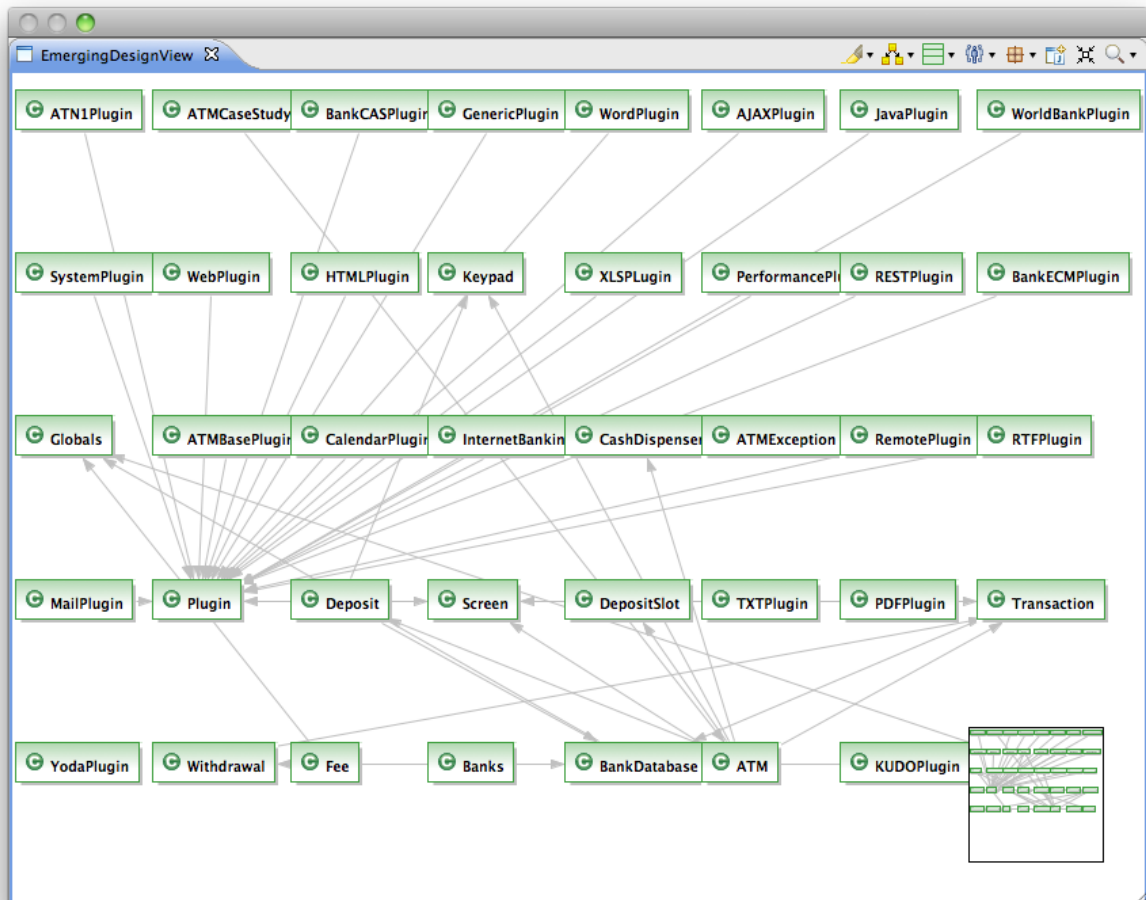


Figure 4: A generic Lighthouse interface with several filters

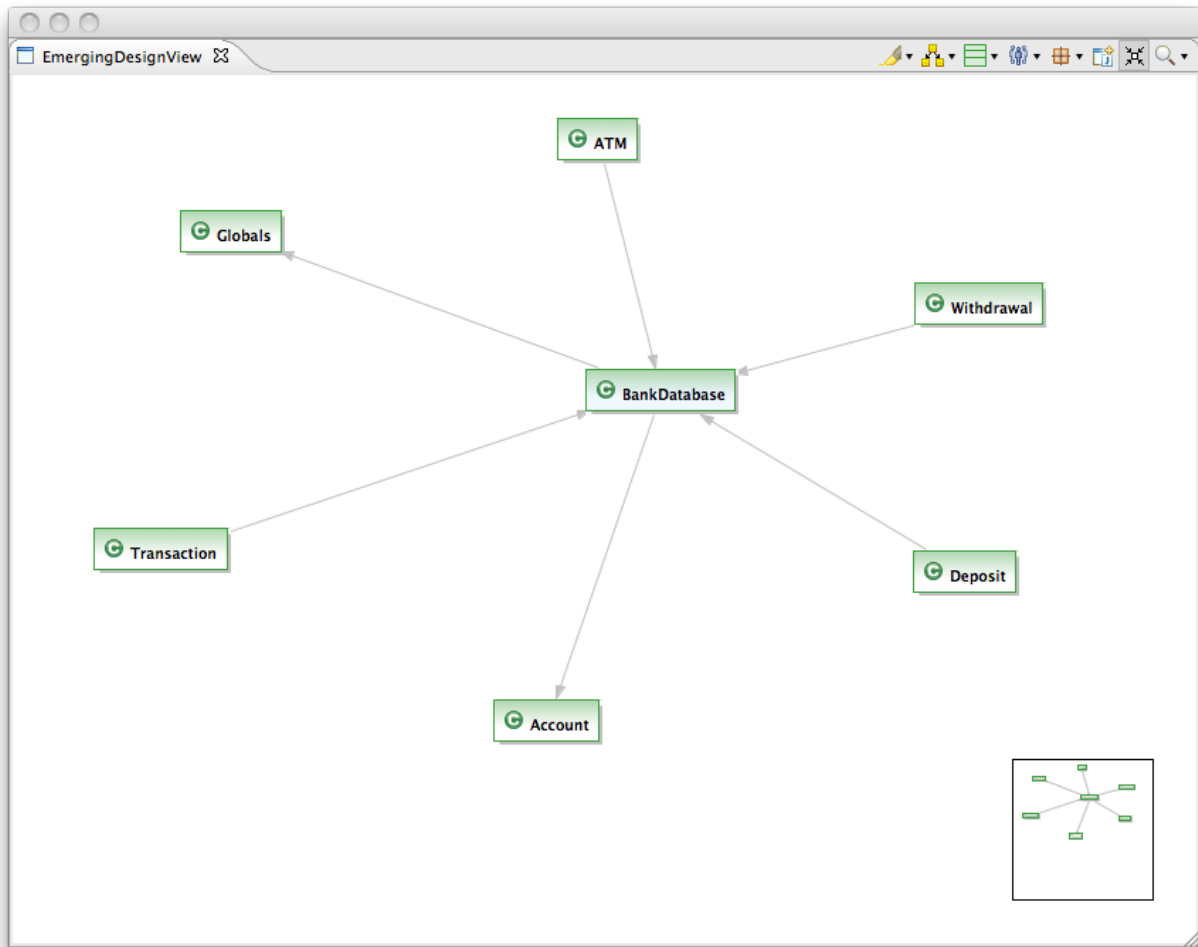


Figure 5: A generic Lighthouse interface with several filters in the 'All methods' mode

