Evaluation of Jupiter and Hackystat Review Analysis: A framework for code review and automatic collection and analysis of review process and data

Master Thesis Proposal

Takuya Yamashita
Collaborative Software Development Laboratory
University of Hawaii at Manoa
takuyay@hawaii.edu

Abstract

Over many years, there is general agreement that software inspection reduces development costs and improves product quality by finding defects in early software development, and these software tools help the code review process efficiently and effectively. In spite of the agreement, developers have not used one review tool for a long time. Or they even have given up the tool any more and gone back to use extremely lightweight text editor tool for code review.

However going back to the text editor review does not solve the underlining problems for the efficiency of review. By using Jupiter (code review tool), the review would be conducted more profitably and practically because of its usefulness and usability. As a result, it helps developers to continuously use the lightweight functional review tool rather than text editor based review tool. By using Hackysatat Review Analysis System the better understanding of review process is provided. By providing enough information about review process reviewers and their team can be aware of the usefulness and usability of the review analysis system.

To investigate my research questions, I will set text based and tool based review experiment in software engineering class to give Java based assignments for around 20 students. In first several round, students are encouraged to learn both text editor based and Jupiter based review. Qualitative evaluation will be conducted after the round to see the usefulness and usability of the review tool. In the next several round, Hackystat review Analysis system will be introduced to gather review metrics and analyze the review data. The second qualitative evaluation will be conducted after the introduction and several uses in order to see the usefulness and usability of the review analysis system..

The expected results would be that the review analysis framework, including the review tool, provides some useful and usable aspects for tool and review process. This result would be one milestone for empirical software inspection community.

Finally, this thesis is going to be done by not later than July, 2005 with first milestone for the implementation of analysis tool by February 2005, second milestone for the evaluation of the review tool by March 2005, and final milestone for the evaluation of the automated review system by May 2005.

1. Introduction

For the past 30 years, software researchers have been establishing theories and best practices to improve software quality [1]. However, no matter how hard the industry tries, software is still released with many bugs. This is not because developers have neglected removing bugs. Most developers have strived removing them by means of exercising unit testing and coverage before release. Existence of bugs in a system in spite of the effort is because developing or developed software is so complicated that they can not recognize the bugs until software is released.

Software review, or code inspection, is one of approach to remove bugs before the software release. Software review is effective in that it can identify most faults during design and code inspection. Here are some specific examples. 93 percent of all faults in a 6000-line business application were found by inspections [2]. Seven thousand inspection meetings included information about 11,557 faults [2]. In addition, there are many other studies sharing that software review identifies and removes faults. Thurs, no one doubts that the software review can identify and reduce faults somehow.

However, a review is not necessary to be conducted efficiently all the time. In a review process, a team experiences common problems that can undermine the effectiveness of software review. For example, too much material may be scheduled for a single review because participants are not aware of realistic review rates [3]. On the contrary, too less preparation time may be scheduled for the review because reviewers are not prepared before the team meeting. Even though reviewers are well prepared, the team may not inspect all raised issues because it runs out of time so that the rest of the untouched issues are marked as validated without careful examination.

Many software review tools have been developed to support the review processes and help reviewer to conduct review efficiently.

CSRS (Collaborative Software Review System) has been used for review in Unix systems with Emacs front-end. This system has full matured functions to support review process in many aspects such as FTArm technique for asynchronous reviews when participants cannot easily meet physically and is under GNU license.

ASSIST (Asynchronous/Synchronous Software Inspection Support Tool) has been used to support any inspection process with client/server architecture. It provides both individual and group-based phases of inspection. The group based phase can be performed synchronously or asynchronously [7].

1.1 Current Issues for the Tools

Even though there are a lot of full functioned review tool out there, it seems that there is no review tool to dominate the review area for a decade. There are many reasons for development teams to give up review even though these review tools are fully functioned. Time for review can not be spared frequently because of the tight software development process even though it helps finding defects. If these tools are commercial product, budget for the review tool would rather cut than update of IDE. Web-based tool can not directly add review comment from a source code so that the usability of the review tool might not defeat that of IDE integrated review tool.

The Collaborative Software Development Laboratory (CSDL) team in the Information and Computer Sciences of the University of Hawaii at Manoa, where CSRS has been developed, has gave up using the CSRS inspection tool in their development process for many years, and its answer for the code review is to be back to using a normal text editor with their own code review guide line [3].

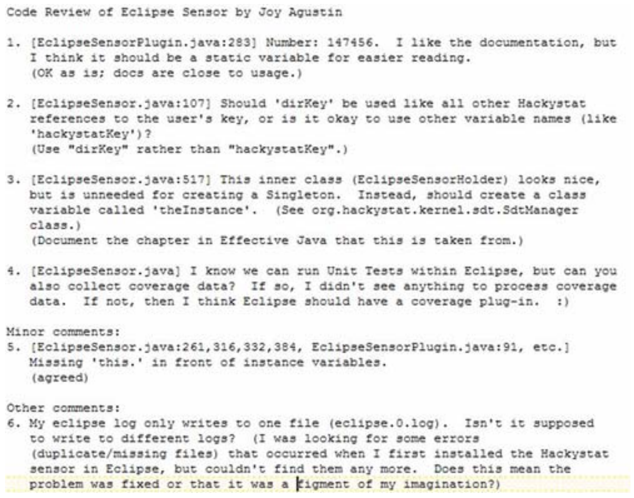
The next section describes the text editor review process for the CSDL.

1.2 Text Editor Based Review

The CSDL review process has five phases: (1) Announcement, (2) Preparation, (3) Review, (4) Revision, and (5) Verification.

In the "Announcement" phase, "the author of a code sends an email to (a mailing list) with description of the software to be reviewed." The amount of review is limited to less than half dozen classes or one package because it wants to keep review "lightweight" and prevent excessive workload for reviewers. The author also provides a list of "burning questions" regarding the code.

In the "Preparation" phase, "all of the participants in the review should read the code and prepare for the meeting", save review comments in a .txt file, and commit it to a CVS repository. An example of the comments for the preparation phase is following:



```
Code Review of Eclipse Sensor by Joy Agustin

1. [EclipseSensorPlugin.java:283] Number: 147456.  I like the documentation, but
   I think it should be a static variable for easier reading.
   (OK as is; docs are close to usage.)

2. [EclipseSensor.java:107] Should 'dirKey' be used like all other Hackystat
   references to the user's key, or is it okay to use other variable names (like
   'hackystatKey')?
   (Use "dirKey" rather than "hackystatKey".)

3. [EclipseSensor.java:517] This inner class (EclipseSensorHolder) looks nice,
   but is unneeded for creating a Singleton.  Instead, should create a class
   variable called 'theInstance'.  (See org.hackystat.kernel.sdt.SdtManager
   class.)
   (Document the chapter in Effective Java that this is taken from.)

4. [EclipseSensor.java] I know we can run Unit Tests within Eclipse, but can you
   also collect coverage data?  If so, I didn't see anything to process coverage
   data.  If not, then I think Eclipse should have a coverage plug-in.  :)

Minor comments:
5. [EclipseSensor.java:261,316,332,384, EclipseSensorPlugin.java:91, etc.]
   Missing 'this.' in front of instance variables.
   (agreed)

Other comments:
6. My eclipse log only writes to one file (eclipse.0.log).  Isn't it supposed
   to write to different logs?  (I was looking for some errors
   (duplicate/missing files) that occurred when I first installed the Hackystat
   sensor in Eclipse, but couldn't find them any more.  Does this mean the
   problem was fixed or that it was a figment of my imagination?)
```

In the "Review" phase, they "go through each comment in each file as a group" during a lunch meeting. A moderator (or the author of the code) opens the line of the code in the class which is referred in a review entry in the text file if necessary. After discussion, the moderator adds a short annotation with a parenthesis at the end of each comment to indicate if the comment is valid or not. You can see the annotation with a parenthesis in the above example.

In the "Revision" phase, they "assign one or more people (usually the authors) to perform revision on the code following the comments brought up in the review and recorded in the text files." After fixing problems, the assigned developers are supposed to comment the status of the resolution in the text file, using a "***" prefix.

In the "Verification" phase, "the group goes (quickly) through the text file one more time, this time focus on what the authors did in response to the comments," by checking the "***" prefix revision comments.

The CSDL review process could minimize the time they spent on it and maximize the benefits. However, using a text editor raises problems of review editing and proper classes finding.

The next section describes the problems we encountered in the CSDL review process with a text editor.

Text editors are universal tool, so review comments written in the text can be easily seen by any developers as well as any reviewers. Due to the universal tool, it can not provide special functionality to support a review process nothing more or less than the functionality for the review process.

In the “Preparation” phase, reviewers are required to launch an IDE to open a system module. Whenever defects in a source code are found, they are required to write its source package and class as a reference of the code (in team, and revision phase). For example, if they are reviewing csdl.jupiter.ui.Foo class and find a defect in the 123 line of the bar() method, they have to describe “[csdl.jupiter.ui.Foo.java : bar() : 123]” as well as a review number. Thus, they are forced to type the extra characters such as “1. [csdl.jupiter.ui.Foo.java : bar() : 123]” in the beginning of the comments for the defect. In addition, they are required to copy and paste the target snippet of a code to the text file. Furthermore, if you write the long comment for the review issue, you might forget the review number. As the result, you need to scroll up to the point in which you can see the previous review number.

In the “Review” phase, the modulator in the team review is required to open the line of the code specified in the review entry with a text editor. Due to the text editor functionality, it takes a lot of time to open the line. In addition, there is no sort function by review items in a text editor; the text editor can not filter the most important reviews that the review team should take a look at. For example, what if there are 5 reviewers and 10 review issues for each reviewer and the most critical defects they found were listed in the end of the review text file? They might not reach the most critical review comments in an hour, which is specified in the time limit for the review phase.

In the “Revision” phase, each reviewer is supposed to open the annotated review text file and open the target source code to be fixed. As we explained in the Review phase, it is easy to imagine that it takes a while to track the source code. In addition, it is hard for each reviewer to find the most critical “needs-fixing” issues first because text editors do not provide sort or filter function for review issues.

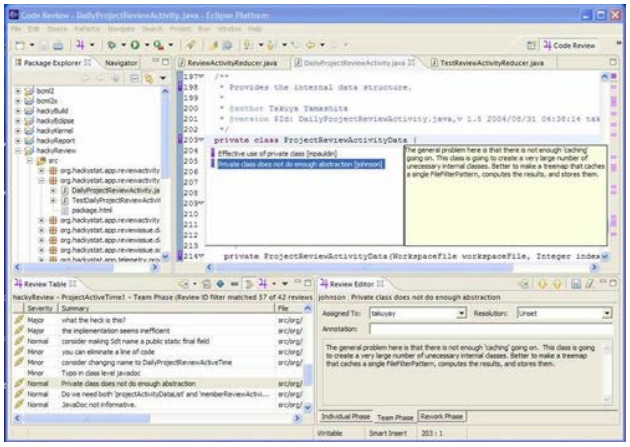
Finally, in the “Validation” phase, a team leader is required to check for any unresolved issues, and notify the assigned developer to fix them. However, since text editors do not provide sort functionality for review entries, the leader has to open all review text files and check review issues one by one.

As we have seen, text editors have a limitation to be used as a review tool. An integrated review tool with an IDE can provide efficient functionalities such as jumping to the target source code which is refereed by a review entry and a sort/filter function by some categories. As a result, the CSDL review process could benefit from an efficient software review tool for the Java environment.

Next section describes a new technology review tool, called Jupiter, which has been developed since 2003 and provides lightweight review tool on the Eclipse platform. This tool would overcome the previous bottleneck for the review tool.

1.3 Jupiter Review Tool

Jupiter has been developed by CSDL since October 2003 as an Eclipse plug-in tool. Its features include: open source, free of charge, cross platform, xml data storage, sorting/filtering, and file integration. It is also designed to support each phase of the CSDL code review process.



Code Review - DailyProjectReviewActivity.java - Eclipse Platform
File Edit Source Refactor Navigate Search Project Run Window Help
Code Review
Package Explorer
Navigator
ReviewActivityReducer.java
DailyProjectReviewActivity.java
TestReviewActivityReducer.java
bowl2
bowl2x
hackyBuild
hackyEclipse
hackyKernel
hackyReport
hackyReview
src
org.hackystat.app.reviewactivity
org.hackystat.app.reviewactivity
DailyProjectReviewActivity.ja
TestDailyProjectReviewActivi
package.html
org.hackystat.app.reviewactivity
org.hackystat.app.reviewissue.d
org.hackystat.app.reviewissue.d
org.hackystat.app.reviewissue.s
Review Table
Review Editor
Individual Phase
Team Phase
Rework Phase
Writable
Smart Insert
203 : 1

```
/**
 * Provides the internal data structure.
 *
 * @author Takuya Yamashita
 * @version $Id: DailyProjectReviewActivity.java,v 1.5 2004/05/31 04:38:14 tak
 */
private class ProjectReviewActivityData {
    Effective use of private class [mpauldin]
    Private class does not do enough abstraction [johnson]

  private ProjectReviewActivityData(WorkspaceFile workspaceFile, Integer index
```

The general problem here is that there is not enough 'caching' going on. This class is going to create a very large number of unecessary internal classes. Better to make a treemap that caches a single FileFilterPattern, computes the results, and stores them.

hackyReview - ProjectActiveTime1 - Team Phase (Review ID filter matched 37 of 42 reviews

Severity	Summary	File
Major	what the heck is this?	src/org/
Major	the implementation seems inefficient	src/org/
Normal	consider making Sdt name a public static final field	src/org/
Minor	you can eliminate a line of code	src/org/
Minor	consider changing name to DailyProjectReviewActiveTime	src/org/
Minor	Typo in class level javadoc	
Normal	Private class does not do enough abstraction	src/org/
Normal	Do we need both 'projectActivityDataList' and 'memberReviewActivi...	src/org/
Normal	JavaDoc not informative.	src/org/

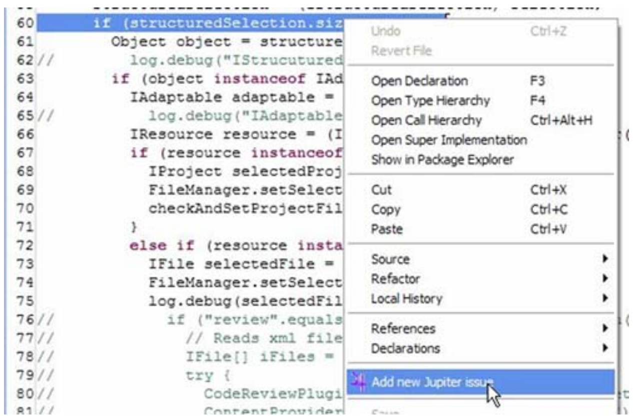
johnson : Private class does not do enough abstraction

Assigned To: takuyay
Resolution: Unset
Annotation:

The general problem here is that there is not enough 'caching' going on. This class is going to create a very large number of unecessary internal classes. Better to make a treemap that caches a single FileFilterPattern, computes the results, and stores them.

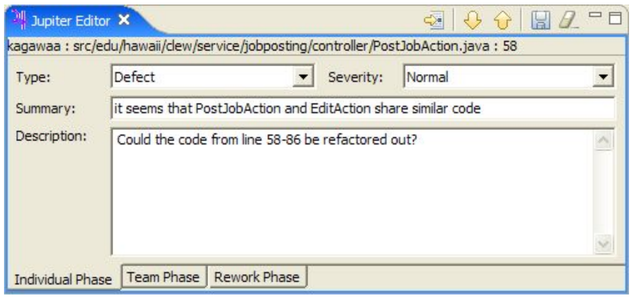
The configuration phase fits the “Announcement” phase in the CSDL code review process. This phase provides review ID configuration to determine a unique review session. The main purpose of the review ID is to manage review issue entries, and avoid file confliction in a CVS by diversifying a review file into the review ID based files. After the configuration, the “.jupiter” file, which describes the sets of review ID, should be committed to the CVS.

The individual review phase which fits the “Preparation” phase in the CSDL code review process. This phase provides the individual review such as reading code, creating new issue, commenting it, and adding it into the Jupiter Issues View. First each reviewer is required to update the “.jupiter” file in the reviewing module from the CVS. To decide which review file is supposed to be written, reviewers have to choose one of review ID, which is described in the “.jupiter” file. In the individual phase, reviewers add issues as many as possible. For example, they select the part of the source code cited by right-clicking on the code, then select “Add new Jupiter issue” to open Jupiter editor. The below screen shot is an example of the process.



```
        if (structuredSelection.siz
          Object object = structure
//          log.debug("IStrucutured
          if (object instanceof IAd
            IAdaptable adaptable =
//            log.debug("IAdaptable
            IResource resource = (I
            if (resource instanceof
              IProject selectedProj
              FileManager.setSelect
              checkAndSetProjectFil
            }
            else if (resource insta
              IFile selectedFile =
              FileManager.setSelect
              log.debug(selectedFil
//              if ("review".equals
//                // Reads xml file
//                IFile[] iFiles =
//                try {
//                  CodeReviewPlugi
```

Reviewers select the "Type" and "Severity" categories, enter a summary and a description, and then click "Enter" button to add the issue to the Jupiter view table. The below screen shot is an example of the Jupiter Editor entry.



kagawaa : src/edu/hawaii/dew/service/jobposting/controller/PostJobAction.java : 58

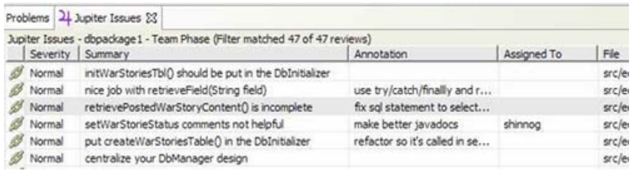
Type: Defect Severity: Normal

Summary: it seems that PostJobAction and EditAction share similar code

Description: Could the code from line 58-86 be refactored out?

Finally reviewers commit their own review ID based file to the CVS. While it takes a while to add a review number, package, class, method, and line number in case of text editor, the problem would be solved by Jupiter. It is because Jupiter fills the information automatically.

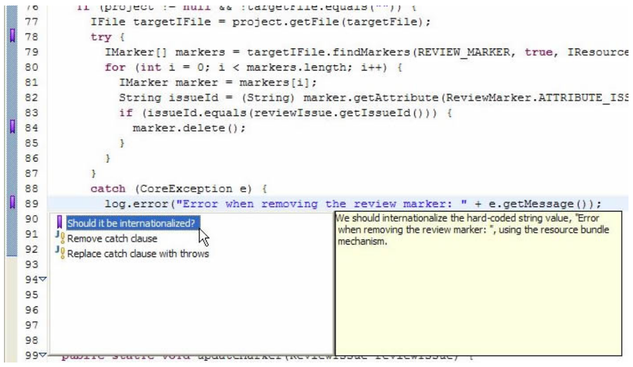
The team review phase, which fits the "Review" step in the CSDL code review phase, provides the group review. The team leader updates the ".jupiter" file in the reviewing module from the CVS, and selects the review ID specified in the configuration phase, and sorts by "Severity" in the Jupiter Issues View The below screen shot is an example of the Jupiter Issue View.



Jupiter Issues - dbpackage1 - Team Phase (Filter matched 47 of 47 reviews)

Severity	Summary	Annotation	Assigned To	File
Normal	initWarStoriesTbl() should be put in the DbInitializer			src/e
Normal	nice job with retrieveField(String field)	use try/catch/finallly and r…		src/e
Normal	retrievePostedWarStoryContent() is incomplete	fix sql statement to select…		src/e
Normal	setWarStorieStatus comments not helpful	make better javadocs	shinnog	src/e
Normal	put createWarStoriesTable() in the DbInitializer	refactor so it's called in se…		src/e
Normal	centralize your DbManager design			src/e

By single-clicking each issue entry in the list, Jupiter opens the Jupiter editor so that they see the summary, discretion, type, severity, and so forth in it. The team leader can jump to the line of code which is referred in a review entry by either clicking "Jump" button in the Jupiter editor or double-clicking on the review entry in the Jupiter Issue view. This feature enables to solve the problem that it takes a lot of time to open the proper line of the code in the class because Jupiter can jump directly from the reviewing entry to the proper source code. Finally after adding "annotation" as a comment for the review entry, and select the resolution, they commit the changed review files to the CVS.



```
      IFile targetIFile = project.getFile(targetFile);
      try {
        IMarker[] markers = targetIFile.findMarkers(REVIEW_MARKER, true, IResource
        for (int i = 0; i < markers.length; i++) {
          IMarker marker = markers[i];
          String issueId = (String) marker.getAttribute(ReviewMarker.ATTRIBUTE_IS
          if (issueId.equals(reviewIssue.getIssueId())) {
            marker.delete();
          }
        }
      }
      catch (CoreException e) {
        log.error("Error when removing the review marker: " + e.getMessage());
```

Should it be internationalized?
Remove catch clause
Replace catch clause with throws

We should internationalize the hard-coded string value, "Error when removing the review marker: ", using the resource bundle mechanism.

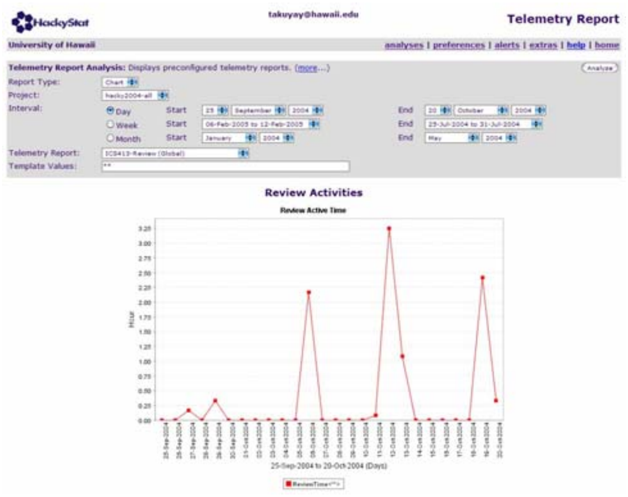
The rework phase, which fits the "Revision" step in the CSDL code review phase, provides the individual rework in order to fix the problems. After fixing problems or not deciding to fix problems, the assigned developer put comments in the revision field and change status from "unresolved" to "resolve". Finally they commit the changed review file to the CVS.

The verification phase, which does not exit in the Jupiter, provides the same process as the team review phase. A team leader (or the author) filters by the "solved" or "unsolved" status, then checks the unsolved problems or checks all revision comments if necessary.

1.4. Hackystat Review Analysis System

Hackystat has been developed by CSDL as an automated metric collection framework. The system provides the client side sensor which is embedded in an IDE or tool to send metric data to server as well as some metric analyses done in server side. For example, The Hackystat sensor for Eclipse can collect the activity of the file editing, unit test result, method coverage of the unit test, CK metrics for a java file, build failure information of a java file, and so forth. As a result, developers can be aware of their analysis in individual, in a project, or between projects.

As an extension to the Hackystat system, I will implement the review analysis package called hackyReview module. It will be one of the Hackystat architecture extension modules to provide the review analysis such as the review activity, number of review issues raised during a phase, review summary analysis, review comparison analysis, and review meeting analysis.



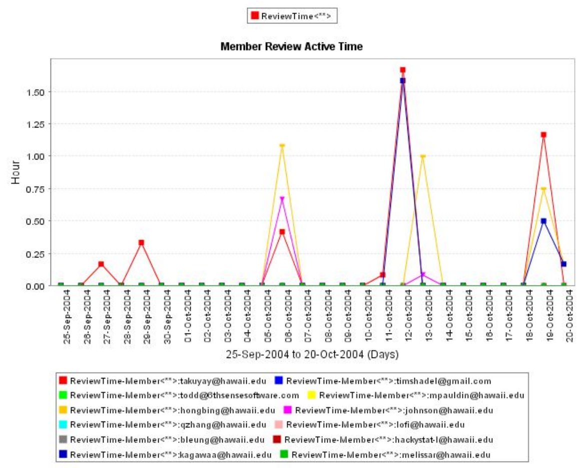
HackyStat takuyay@hawaii.edu **Telemetry Report**

University of Hawaii — analyses | preferences | alerts | extras | help | home

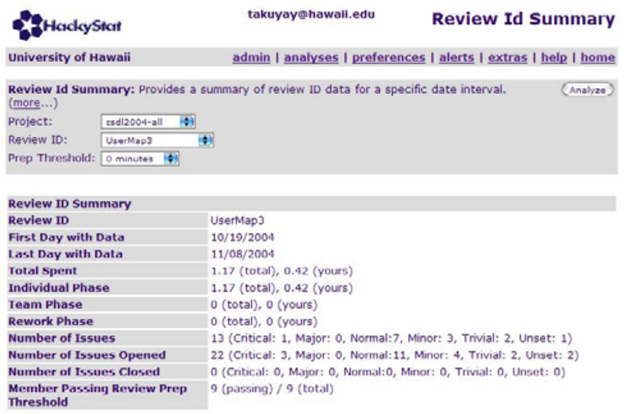
Telemetry Report Analysis: Displays preconfigured telemetry reports. (more...)

Report Type:	Chart
Project:	hacky2004-all
Interval:	● Day Start 25 September 2004 End 20 October 2004
	○ Week Start 06-Feb-2005 to 12-Feb-2005 End 25-Jul-2004 to 31-Jul-2004
	○ Month Start January 2004 End May 2004
Telemetry Report:	ICS413-Review (Global)
Template Values:	**

The review activity, say review active time, can measure how much individual reviewer can spend his activity for a review. The following figure shows the experimental review active time for a reviewer in the specific period within a project [10]. It presents that each review process happened constantly with a specific time period, and the first day review time (relatively large amount of time, i.e., individual phase) is followed by the second day review time (relatively small amount of time, i.e., team phase) for the last two data sets. On the other hand, the third set from the last has only one review day with relatively large amount of time, which shows the individual phase was held. These set of review time telemetry stream would show some useful insight for users.



The next figure shows the review active time between reviewers in the specific period in a project. Each color stands for a reviewer. This would provide the useful insight on the pre-review meeting decision visually. If nobody spent a enough time for individual phase, it would be better for moderator or project manager to make a decision to postpone the review meeting because reviewers could not understand the source code and meeting content. For example, if a team decide to request reviewers to spend at least a half hour for at least three reviewers out of four, the team lead can make a decision from the review active time telemetry stream.



HackyStat takuyay@hawaii.edu **Review Id Summary**

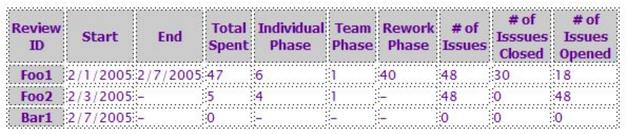
University of Hawaii — admin | analyses | preferences | alerts | extras | help | home

Review Id Summary: Provides a summary of review ID data for a specific date interval. (more...)

Project:	csdl2004-all
Review ID:	UserMap3
Prep Threshold:	0 minutes

Review ID Summary	
Review ID	UserMap3
First Day with Data	10/19/2004
Last Day with Data	11/08/2004
Total Spent	1.17 (total), 0.42 (yours)
Individual Phase	1.17 (total), 0.42 (yours)
Team Phase	0 (total), 0 (yours)
Rework Phase	0 (total), 0 (yours)
Number of Issues	13 (Critical: 1, Major: 0, Normal:7, Minor: 3, Trivial: 2, Unset: 1)
Number of Issues Opened	22 (Critical: 3, Major: 0, Normal:11, Minor: 4, Trivial: 2, Unset: 2)
Number of Issues Closed	0 (Critical: 0, Major: 0, Normal:0, Minor: 0, Trivial: 0, Unset: 0)
Member Passing Review Prep Threshold	9 (passing) / 9 (total)

The next analysis that would be provided in the Hackystat system is the review ID summary for a specific review ID. The Hackystat analysis section provides this analysis with the project selection and review ID field to determine a specific review ID. The example provides the detail summary for a specific review ID. The start and end time provides the review process span, say the period from the individual phase to the rework phase done. The next following four rows provide the review active time for total and each phase. The last three rows provide the number of issues, issues closed, and issues opened with severity level. For example, in the # of Issues row, there are forty eight issues with the critical issue three, the major issue five, the normal issue twenty, the minor issue eight, and the trivial issue twelve. These rows would provide a team lead or assigned person to manage the confirmed review issues by seeing if how many confirmed review issues are still not closed or not.



Review ID	Start	End	Total Spent	Individual Phase	Team Phase	Rework Phase	# of Issues	# of Isssues Closed	# of Issues Opened
Foo1	2/1/2005	2/7/2005	47	6	1	40	48	30	18
Foo2	2/3/2005	–	5	4	1	–	48	0	48
Bar1	2/7/2005	–	0	–	–	–	0	0	0

The final analysis that would be provided in the Hackystat system is the review comparison between review IDs. This would give users good insight on statistic information about review processes. Each review ID row has the start and end time, the total review active time spent, the active time spent for individual, team, and rework phase. The total review active time would be the summation of the review active time for the individual phase, that for the team phase, and that for the rework phase. The row also provides the number of issues raised, closed, and opened. The number of review issues raise would be the summation of the review issues closed and the review issues opened. It would be useful for users to understand which review process would not be completed yet, how many un-closed issues exist, and so forth.

2. Related Work

There are a number of tool comparisons among review tools [8]. The web-based review system called Codestriker can run on Windows and Unix (including Linux), support database system (such as Oracle, MySQL, and so forth), and support many version control system such as CVS [8]. It also provides CVS diff function to see the difference between the versions, mail notification system. One of the problems in the system is that it can deal with only pure text files [8]. Second problem is that the mail notification system sends a lot of emails and does not allow users to customize their email preferences [8].

While there are a number of tool comparisons among review tools, there is a comparison between the tool-based and paper-based review inspection [7]. The representative tool of the tool-based called ASIST (Asynchronous / Synchronous Software Inspection Tool) is compared with the paper based with the controlled experiment to see the significant difference between them about defect detection, false positives, and meeting gain and loss, and as a result, there is no significant difference between them about all of them [7]. The only 22 percent of respondents for the usability questionnaire of the individual inspection claimed to have performed better with ASSIST while the 39 percent claimed to have performed better with the paper-based inspection [7]. On the other hand, the 61 percent of respondents for the same questionnaire of the group meeting clamed to have performed better with ASSIST while only 19.5 percent is for the paper based inspection [7]. The reason is that "a number of people found it awkward moving between the code, specification and checklist windows of ASSIST".

There are some tools to provide the review metrics in the review system. Codestriker provides the fundamental review metrics such as how large each topic is, who participated, how long they spent, and how many defects they found as well as the overview meeting time and preparation time [8]. The article says that they "use the inspection metrics and justify the long-term use of inspection and to monitor their effectiveness", but does not describe how the metrics is used to justify the long-tem use and what is the effectiveness of the review process [8].

5. Research Question

As mentioned in section 1.2, exercising software review with text editors was limitations because it takes time to open a proper code review fragment and there are no sort and filter functionalities. In addition, CSDL developers have moved to use Eclipse platform to develop software, so it was good chance to have integrated software review tool as a plug-in of Eclipse. Thus we designed and implemented Jupiter to compliment the requirements.

As described in section 1.3, Jupiter provides the facility to jump to a proper code review segment from a review entry by just clicking jump button in the Jupiter editor pane or double-clicking on a review entry in the list of review entries. It also provides the sort and filter functions to distinguish between higher and lower important review entries. From a developer point of view, I have implemented the functions to be useful for users.

A research question I raised after the implementation is that the Jupiter system does provide the usefulness functionalities for review. In other words, given the Eclipse integrated software review tool, does the system provide a beneficial and practical way to conduct a code review?

Another research question I raised is that the Hackystat review analysis system does provide better understanding of review process, given the Jupiter provides the useful functionalities. In other words, does the Hackystat review analysis system give a good insight on understanding review process detail such as active time spend for each phase, the number of total, closed, opened issues, and so froth?

The next section describes the experimental evaluation methodology.

6. Experiment Design

In order to evaluate if the automated review framework provides the better understanding of inspection process as a result of the fact that developers continuously use the lightweight functional tool rather than text editor based tool, I propose the two hypotheses:

1) Claim 1: After mature use of Jupiter, software developers well find it more useful and more usable than the text-based review.
2) Claim 2: The review data and analysis provided by Jupiter and Hackystat are both useful and usable.

To test the hypotheses, I propose two stages to evaluate claim 1 and 2: 1) evaluation of the usefulness of Jupiter review tool and 2) evaluation of better understanding of the automated review process with

Jupiter and Hackystat. The second evaluation is supposed to be conducted after the significant result for the evaluation of Jupiter review tool.

6.1 Experiment Design for Jupiter review tool

To evaluate the usefulness and usability of Jupiter plug-in for Eclipse (Claim 1), I propose that the evaluation is considered to be based upon qualitative approach: Questionnaire. The questionnaire will be conducted after the use of the text-based editor and the use of Jupiter.

6.1.1 Subjects

The experiment will be carried out during the spring semester in 2004 from January to May as a part of both introductory Software Engineering course for approximately 10 undergraduates and 10 graduates in University of Hawaii at Manoa's information and Computer Sciences department. It is assumed that the subjects has the background of the required computer science such as programming language, algorithm. However, there will be a room that different students have different ability of skills. To minimize the deficiency of skills, students are supposed to review the basic knowledge which would be necessary to evaluate the claim 1. That is to say, the basic knowledge of Java language, Text editor, Eclipse IDE, and so forth will be reviewed. All students are supposed to have the identical educational tutorial on how to use the text editor in Eclipse IDE before the post questionnaire for Text editor is conducted. They are also supposed to have the identical educational tutorial on how to use the Jupiter after the text editor review process is mastered and before the post questionnaire for the Jupiter editor is conducted.

6.1.2 Materials

The material to be used for review is the code actually students wrote in Java during the course. For each session of review including preparation, individual, and rework phase, the same material is used for the paired team. The appropriate material to be used in each session will be determined by the course instructor and/or students in such a way that it contains if 1) there are proper lines of code, 2) there exist enough defects, and 3) there are enough parts that educate students. I will also prepare for each session the backup materials to have all of them just in case that there exist no materials as such. It is considered that this approach would be better way than the approach that all materials are prepared before course starts. It is because 1) Instructor has own plan to organize his class and 2) the source codes students wrote are more useful for them to understand and recognize the defects.

6.1.3 Instruments

The main instrument is Eclipse IDE, which is the integrated software development tool, and Jupiter, which is the code review plug-in to the Eclipse. As student experience software engineering course, some software engineering tools such as unit test framework, ant build too, configuration management tool, and web application tool are introduced step by step.

To eliminate the external factor of usability of editor, Text editor embedded in Eclipse is used for the text based review. So students are supposed to experience both text editor review and Jupiter based review within Eclipse IDE.

To see the estimated review time in individual phase, Jupiter sensor for Hackystat is used. This sensor records the review time without any special action such as writing start time and end time or even pushing start and end button in the tool. Text editor used is a special editor for review, whose most function is the same as regular Eclipse editor, but it can be opened by Jupiter so that the review time for text editor and Jupiter is recorded.

6.1.4 Experiment Execution

Before the experiment starts, subjects are given lecture on how to conduct review process. The lecture includes the explanation of the research purpose, review process, review iteration period, review tool, and so forth.

To iterate though review sessions, the first round is set as three review sessions. The three sessions include two general review process and one individual group review process. A general review process is determined as the process that one review material would be reviewed in individual phase and team phase by all students. On the contrary, an individual group review process is determined as the process that each small groups has own individual materials to be reviewed in the individual, team and rework phases. The reason why there are two different type of review process is that only individual review phase can conduct the rework phase since the author of the materials would be the rework person. A session is determined as the process that includes individual phase, team phase, and rework phase. The small groups conduct the first round with the text editor in Eclipse.

In individual phase, students are given a source code and supposed to review for just an hour. They are supposed to finish the review regardless of any reason. Jupiter review sensor helps seeing the review time for students. If there is a significant time difference from the time period, students are asked for the reason, and the problem would be solved in the next iteration. After the review done, the review files for students are emailed to Instructor or me.

In team phase, students conduct the team review with the small team members. Instructor or I measure one hour with physical timer in the class. They review all raised issues and validate issues as much as time is allowed. They can the validate issues by setting the resolution status such as “Vaid-NeedsFixing”, “Valid-FixedLater”, “Valid-WontFix”, and so forth. They are supposed to finish the team review in an hour regardless of any reason and send the review file to Instructor or me.

In rework phase, students conduct the rework review only with the individual group review process. The time spent for the rework phase is not calculated because it would be hard to measure the time for rework phase. However, students are asked to fix the all confirmed review issues in a specific period of time. This phase is especially evaluated by the questionnaire.

The review process including as individual, team, and rework phase is the same for the text editor based review and Jupiter based review.

To evaluate the distinct usefulness of Jupiter compared to text editor in Eclipse, I will ask students to have questionnaire after both tool are used. The first questionnaire will be conducted after the first round with the Text editor is done. This would be the evaluation for the usefulness of the Text editor and its review. The second questionnaire will be conducted after the second round with Jupiter is done. This would be the evaluation for the usefulness of the Jupiter and its review.

6.2 Experiment Design for automated review process with Hackystat

To evaluate the usefulness and the usability of the review data and analysis provided by Jupiter and Hackystat, I propose that the evaluation is considered to be based upon qualitative approach: Questionnaire. The questionnaire will be conducted after the use of the Jupiter and Hackystat review analysis system.

6.2.1 Subjects

The subjects for the evaluation would be the exact same as that of the Jupiter tool evaluation.

6.2.2 Materials

The material to be used for review is the review data of the students who wrote in Java during the course. After the Jupiter use, the students are asked to install Jupiter review sensor to collect the review activity and review issues. Since these data would be collected without the reviewer’s consciousness, I could avoid the situation that reviewer forget pushing the start button to start sensor or the end button to stop sensor. The valid data would be the review data in a project which is associated with the class room setting. In other words, the data for the personal programming would be ignored.

6.2.4 Experiment Execution

To evaluate the review analysis system, the students are supposed to use the review system analysis after each review session done.

For the first several analysis task, I or course instructor can ask the homework to check if they reached the specific review analysis in Hackystat and invoke the analyses. I explain the main goal of each analysis and functions so that they can understand the analysis. After several experiment done, I provides the questionnaire to evaluate the usefulness and usability of the review analysis system.

I believe that the review analysis system provides the additional value to help or give good insight on the review and review process.

For example, the preparation review threshold analysis provides to see if the review meeting phase should be held or not by the amount of reviewers who conducted in the individual phase. This analysis would help the review members to discuss the review material in the team phase because it would be wasting time if nobody conducted individual review before the team meeting.

Another example to take is that the review ID summary analysis provides the perspective summary for a specific review session. The number of review issues closed would give a good insight on how many confirmed review issues in the team phase have not been resolved yet in each severity level. So the students can check to see if all review issues confirmed were solved from the analysis.

The last example is that the review comparison analysis provides the comparison summary between review IDs. The trend for the active time and number of issues is reported in the analysis. Thus the students would see visually see the progress of review among the review IDs.

7. Time line

This thesis is going to be done by not later than May, 2005 with first milestone for the implementation of analysis tool by the middle of February 2005, second milestone for the evaluation of the review tool by March 2005, and final milestone for the evaluation of the automated review system by April 2005.

First Milestone (Feb/2005)	Review Analysis Implementation

Second Milestone (March/2005)	Jupiter Review Tool Evaluation
Final Milestone (April/2005)	Review Analysis System Evaluation
Defense (July/2005)	Defense and Submission to Grad Division

9. References

[1] Parnas, David L., *"The Role of Inspection in Software Quality Assurance"*, IEEE TRANSACTIONS ON SOFTWARE ENGINEERING VOL.29, NO.8 AUGUST 2003, IEEE Computer Society, August 2003, pp. 674-676.

[2] Pfleeger, Shari lawrence, *"Software Engineering: theory and practice"*, 2nd ed, Publisher, Location, Date.

[3] Wiegers, Karl E., *"Seven Deadly Sins of Software Reviews"*, Software Development magazine March, 1998

[4] Barnard, Jack, *"Managing Code Inspection Information"*, IEEE Software, Vol.11, No.2, Mar. 1994, pp.59-69.

[5] Anderson, P., Reps, T., Teitelbaum, T., and Zarins, M., *"Tool Support for Fine-Grained Software Inspection"* IEEE Software , Vol. 20, No. 4, July/auguest 2003, pp. 42-50.

[6] Filippo Lanubile, and Teresa Mallardo, *"Tool Support for Distributed Inspection"*, Proceedings of the 26th Annual International Computer Software and Applications Conference - August 26 - 29, 2002, Oxford, England pp. 1071

[7] F.Macdonald and J.Miller, *"A Comparison of Tool-Based and Paper-Based Software Inspection"*, ISERN-98-17/EFoCS-25-97 [RR/97/203]

[8] Jason Remillard, "Source Code Review Systems", IEEE Software , Vol. 22, No. 1, January/February 2005, pp. 74-77.

[9] Takuya Yamashita, "Jupiter Version2 User's Guild", http://csdl.ics.hawaii.edu/Tools/Jupiter/Core/doc/UsersGuide.html

[10]
Philip M. Johnson and Hongbing Kou and Michael G. Paulding and Qin Zhang and Aaron Kagawa and Takuya Yamashita, "*Improving Software Development Management through Software Project Telemetry*", Accepted for publication in IEEE Software, August, 2004.

INFORMED CONSENT FORM
Collaborative Software Development Laboratory
University of Hawaii at Manoa

Takuya Yamashita
Primary Investigator
(808) 956-6920

This research is being conducted to examine the usability of the Jupiter plug-in for Eclipse. The purpose of the study is to investigate the usability of the Jupiter plug-in in Software Engineering class in University of Hawaii at Manoa, Information and Computer Sciences department, for the purpose of determining how Jupiter plug-in are easily used by the users. You are being asked to participate because you are computer science students, who are taking ICS 414 and ICS 613 software engineering class. Your participation will help see the usefulness of Jupiter plug-in.

Participation in the questionnaire will take approximately fifteen minutes. The questionnaire will involve completing a simple task to evaluate the code review and Jupiter functions.

Confidentiality will be applied to the questionnaire. The names of the participants will not be published or recorded at the questionnaire. The materials gathered during the questionnaire will be destroyed after the data analysis and report are complete.

Participation is voluntary, and refusal to participate will not result in any loss of benefits to which the participant is otherwise entitled. Participants may at any time withdraw from the test with no penalty or loss of benefits to which the subject is otherwise entitled. At the point of withdrawal, all information gathered from the participant will be destroyed.

The participant will be required to use the Eclipse IDE and Jupiter plug-in, and will be held responsible for any injuries that are a result of using the Eclipse IDE and Jupiter plug-in. By participating in this questionnaire, the participant accepts all responsibilities that may occur as a result of completing the questionnaire.

The participant is entitled to see the final research and report upon request. For additional information about the questionnaire, procedures, and/or results, please contact the primary investigator, Takuya Yamashita, at (808) 956-6920.

If you have any questions regarding your rights as a research participant, please contact the UH Committee on Human Studies, at (808) 956-5007.

Participant:
I, hereby, understood the above information, and agree to participate in this research project.

____
Name (printed)

____________________________________ __________________
Signature Date

INSTRUCTIONS

The following is a questionnaire on your experiences using Jupiter plug-in, code review system. There are 22 questions, and it will probably take you between 15 minutes to fill out. If you have any questions, please don't hesitate to ask.

You can put an answering number by select one proper answer and circle it.

It is important to remind you that it doesn't matter what your answers are. So please be honest; that will make it most useful for this research. If you have questions, please ask them before you begin. If you get stuck, then ask questions on a need-to basis.

In order to better understand the strengths and weaknesses of the current version of Jupiter code review system, and to help guide future improvements, please take a few minutes to answer the following questions. It is important to remind you that your identity will be removed before performing any analysis on this data. There are no right or wrong answers: We want to know what your personal experience was.

Some questions ask you to respond with a number from 1 to 5, where the 1
Indicates the "best" and the 5 indicate the "worst". The last question of this questionnaire requests your comments as the response.

I. DEMOGRAPHIC QUESTIONS

1. Have you ever reviewed a source code with an editor such as MS Note pad, Textpad, or the equivalent text editor?
   1 Yes
   2 No

2. Have you ever used Jupiter code review plug-in for Eclipse?
   1 Yes
   2 No

II. USABILITY and UTILITY for a text-based review tool.

This section asks for your opinion on the usability and utility of the primary functions used to track your progress. I define “text-based review” to mean the code review recoded in a text file by using a text editor. I define "usability" to mean the ease of invoking text-based review tool functions and understanding what the results mean. I define "utility" to mean the usefulness of the text-based review tool functions; do the text-based review tool functions support the review process that is actually helpful to you. I also ask for your opinion on the "overhead" of code review you experienced with the text-based review tool, in other words, how much work was required after installation and configuration to gather data and perform analyses:

INDIVIDUAL REVIEW PHASE

3. The amount of overhead required to add an issue with the text-based review tool was:
(Very Low) 1 2 3 4 5 (Very High)

4. The amount of overhead required to fill an issue information with the text-based review tool was:
(Very Low) 1 2 3 4 5 (Very High)

5. The individual phase functions of the text-based review tool was (including creating, editing, editing review issues,
) was:
(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

TEAM REVIEW PHASE

6. The amount of overhead required to review the list of issues with the text-based review tool in a team was:
(Very Low) 1 2 3 4 5 (Very High)

7. The amount of overhead required to review each issue information such as the summary, description, and so forth with the text-based review tool in a team was:
(Very Low) 1 2 3 4 5 (Very High)

8. The amount of overhead required to jump (switching from an issue in the text file opened in the text editor to the target point of a source code in Eclipse IDE) was:
(Very Low) 1 2 3 4 5 (Very High)

9. The team phase functions (including managing review issues (sorts, filters), jumping to a target file) was:
(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

REWORK PHASE

10. The amount of overhead required to review the assigned issues after team review done was:
(Very Low) 1 2 3 4 5 (Very High)

11. The amount of overhead required to fix the problem which was assigned to you was:
(i.e. find the point of problem, and change (fix) the source code)
(Very Low) 1 2 3 4 5 (Very High)

12. The rework phase functions (including managing opened or closed review issues (sorts, filters), jumping to a target file, having review markers) was:

(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

III. INSTALLATION/CONFIGURATION

Please provide us with your opinions regarding the installation and configuration of the Jupiter plug-in.

13. Installing the Jupiter code review plug-in was:
(Very Easy) 1 2 3 4 5 (Very Difficult)

14. Configuring Jupiter to add the new review ID
(Very Easy) 1 2 3 4 5 (Very Difficult)

15. Configuring Jupiter to manager the Review ID (i.e. modification, and deletion of a Review ID)
(Very Easy) 1 2 3 4 5 (Very Difficult)

IV. USABILITY and UTILITY for Jupiter code review tool.

This section asks for your opinion on the usability and utility of the primary functions used to track your progress. I define "usability" to mean the ease of invoking Jupiter functions and understanding what the results mean. I define "utility" to mean the usefulness of the Jupiter functions; do the Jupiter functions support the review process that is actually helpful to you. I also ask for your opinion on the "overhead" of code review you experienced with the Jupiter plug-in, in other words, how much work was required after installation and configuration to gather data and perform analyses:

INDIVIDUAL REVIEW PHASE

16. The amount of overhead required to add an issue with Jupiter was:
(Very Low) 1 2 3 4 5 (Very High)

17. The amount of overhead required to fill an issue information with Jupiter was:
(Very Low) 1 2 3 4 5 (Very High)

18. The individual phase functions (including creating, editing, editing review issues,

) was:
(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

TEAM REVIEW PHASE

19. The amount of overhead required to review the list of issues with Jupiter in a team was:
(Very Low) 1 2 3 4 5 (Very High)

20. The amount of overhead required to review each issue information such as the summary, description, and so forth with Jupiter in a team was:
(Very Low) 1 2 3 4 5 (Very High)

21. The amount of overhead required to jump to the target point of a source code from the list of issues (i.e. Jupiter Issue View, double-clicking on an issue in the list) was:
(Very Low) 1 2 3 4 5 (Very High)

22. The amount of overhead required to jump to the target point of a source code from the reviewing issue (i.e. Jupiter Editor, and clicking Jump button) was:
(Very Low) 1 2 3 4 5 (Very High)

23. The team phase functions (including managing review issues (sorts, filters), jumping to a target file, having review markers) was:
(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

REWORK PHASE

24. The amount of overhead required to review the assigned issues after team review done was:

(Very Low) 1 2 3 4 5 (Very High)

25. The amount of overhead required to fix the problem which was assigned to you was:
(i.e. find the point of problem, and change (fix) the source code)
(Very Low) 1 2 3 4 5 (Very High)

26. The rework phase functions (including managing opened or closed review issues (sorts, filters), jumping to a target file, having review markers) was:

(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

V. USABILITY and UTILITY for Hackystat Review Analysis system.

This section asks for your opinion on the usability and utility of the primary analyses used to track your progress. I define "usability" to mean the ease of invoking an analysis and understanding what the results mean. I define "utility" to mean the usefulness of the analysis; does the review analysis provide information that is actually helpful to you.

27. The Review Active Time telemetry analysis (showing review active time,
) was:
(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

28. The Review Issue telemetry analysis (showing review issue raised, issue closed) was:
(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

29. The Review ID summary analysis (showing the summary of a review ID) was:
(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

30. The Review Comparison analysis (showing review trend between review IDs) was:

(Highly Usable) 1 2 3 4 5 (Not Usable At All)
(Highly Useful) 1 2 3 4 5 (Not Useful At All)

31. Please provide any other feedback you can on the usability and utility of these or other Hackystat Review analyses, as well as any suggestions you have on how we can improve usability and utility in future.

VI. FUTURE USE

In this section, we are interested in learning whether you would consider the Jupiter plug-in to be feasible (i.e. appropriate, useful, beneficial) for use in a professional software development context.

32. If you were a professional software developer, using Jupiter at your job would be:
(Very Feasible) 1 2 3 4 5 (Not Feasible at All)

33. If you were a professional team (or project) leader, using Jupiter at your job would be:
(Very Feasible) 1 2 3 4 5 (Not Feasible at All)

34. Please provide any other feedback you could have experience on the use of Jupiter, as well as any suggestions you have on how we could improve its use in future.