UNIVERSITY OF HAWAII AT

MANOA

THESIS PROPOSAL

EVALUATION OF THE AGILE SOFTWARE REVIEW OPTIMIZATION USING JUPITER AND HACKYSTAT SYSTEM

Takuya Yamashita

In Software Engineering, Examining a code is the very important part of a software development process to find faults. The code review is defined to review both code and its documentation for misunderstandings, inconsistencies, and other faults. Some researchers have investigated the extent of the effectiveness of the code review for defect detections. Most researches say that many parts of finding defections fall in the review process no matter what percentage of the defect detection to the total software development process is. This means that the better review-covered the system is, the less found the potential defects are in the rest of software development process or in the final product. In other words, if external developers (reviewers) double check (review) the target system in an optimal manner, the system is less likely to have defects. However, how do we find or determine the optimal manner, which would maximize the efficiency of a software review?

In this study, I plan to investigate the evaluation of the agile software review optimization in terms of 4 main determinants of a review condition, using JUPITER and Hackystat System. In other words, I would like to see WHEN a review should be initialized, WHO among the developers should be involved in the review, WHAT files should be subject to be reviewed, HOW the reviewers should focus their energies during their review.

The JUPITER (JUst Point In Time Extreme Review) is the review system plug-in for the Eclipse IDE.system developed by me. It helps the creation of review data seamlessly. The Hackystat system is the software engineering statistical data collection and analysis system. Its Eclipse IDE sensor sends row data to the Hackystat system. They help analyze the rule-based optimization of a software review.

1.1 Motivation

Traditionally software has been reviewed in a centralized development process. For any software review process, a manager or leader is authorized to allocate staff and material resources to review, and development staff of reviewer has to be co-located, the development process has to be held as intermittent interruptions for the staff. Since the resources are gathered at once for review process, this traditional software review process would be called "centralized" process.

However in the internet era, developers tend to cooperate to develop software in inter-development review manner. Imagine an open source development project with hundreds of developers working on many different modules in a geographically distributed manner. Traditional approach would be failed in some sense for this inter-development process. Or more mildly speaking, the traditional software review methods are not easily applied to the inter-development situations. If so, how can we cope with these situations?

We propose a new approach to software review which is held with appropriate reviewers on appropriate time for appropriate artifacts in appropriate way. To determine the WHO, WHEN, WHAT, HOW, data about development process is gathered and analyzed via the Hackystat System. We will use the Hackystat system to attach sensor to Eclipse developer tool that sends data automatically. The data gathered will include information about structured of code, the developers that worked on it, the kinds of changes that have occurred to individual files, the reviews that have been done on the files and their review results, and defects that have been recorded. Given the data, the system will determine the WHO, WHEN, WHAT, HOW questions of software review, which depends upon the rule-based function. After the determination, the system will send email to developers when it judges that a new review should be initialized.

1.2 Experimental Design

1.2.1 Primary Objective

This study investigates making software review more agile and applicable to open source software development communities. The agile software review means that software review is initialized just in time by means of answering WHAT, WHEN, WHO, and HOW questions of review. In other words, it determines how optimally the resources such as artifacts, time, people, and situations, are allocated. After the investigation, we will contribute this results and system to open source software development communities.

The primary evaluation of this optimization is provided by questionnaires. The questionnaires focus on how much just in time the system initializes the optimal software review in the decentralized manner.

1.3 Control Variables

Review Tools

Eclipse IDE

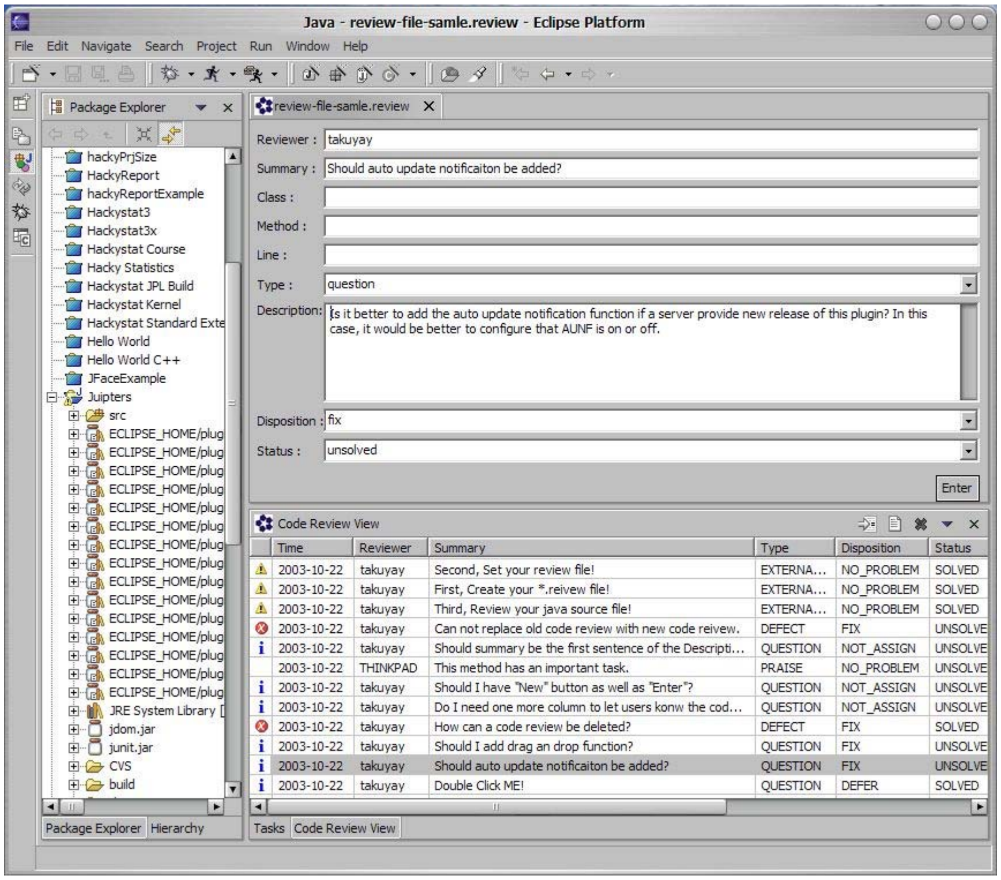
The reviewers are supposed to use the Eclipse IDE (http://eclipse.org) to take a look at a system. Eclipse gives reviewers great functionalities to review a system. Whatever they find, the reviews that have been done on the files and their results, and defects are recorded through JUPITER system plug-in for Eclipse. Hackystat Eclipse sensor hooks to the Eclipse to gather and record data about the structure of code, the developer that worked on it, the kinds of changes that have occurred to individual files.

JUPITER Plug-in for Eclipse

The JUPITER Plug-in for Eclipse would facilitate and automate code review process. In personal review process, reviewers can take a look at a source code to be reviewed. Whatever they find, right click on either (static, field, or local) variable, method, or class (as well as interface), then jump to the code review editor to record it. The reviewer name, class, method, and line number, would be automatically filled out in their fields. The reviewer's name is based upon system user name or define-able in the preferences of JUPITER. Once every fields are filled out, the review would be posted in the Code Review View by the enter button clicked. The reviewers can add the review record as much as possible.

In group meeting, a principle code inspector gathers all code reviews by selecting "Import Code Review Files" on popup menu so that all the selected reviews in a directory would be scanned and imported to the view list table automatically. This table is also planned to have some sort algorithms to figure out duplicated review part and list next to the first reviewer's review, or sort by type, reviewer's name, class, and so forth. He/she can jump to the particular line of the source code by double-clicking the review on the list if the line number and class are specified, while he/she can edit the review anytime by single-clicking the review on the list. In this review process, members can argue the proposition such as "fix", "defer", "duplicate", or "no problem".

After the meeting, reviewee can see the "unsolved" reviews by sorting by "status", and fix the bug if any, then finally set the status to the "solved" from the "unsolved".



Reviewer :	takuyay
Summary :	Should auto update notificaiton be added?
Class :	
Method :	
Line :	
Type :	question
Description:	Is it better to add the auto update notification function if a server provide new release of this plugin? In this case, it would be better to configure that AUNF is on or off.
Disposition :	fix
Status :	unsolved

	Time	Reviewer	Summary	Type	Disposition	Status
⚠	2003-10-22	takuyay	Second, Set your review file!	EXTERNA...	NO_PROBLEM	SOLVED
⚠	2003-10-22	takuyay	First, Create your *.reivew file!	EXTERNA...	NO_PROBLEM	SOLVED
⚠	2003-10-22	takuyay	Third, Review your java source file!	EXTERNA...	NO_PROBLEM	SOLVED
⊗	2003-10-22	takuyay	Can not replace old code review with new code reivew.	DEFECT	FIX	UNSOLVE
ℹ	2003-10-22	takuyay	Should summary be the first sentence of the Descripti...	QUESTION	NOT_ASSIGN	UNSOLVE
	2003-10-22	THINKPAD	This method has an important task.	PRAISE	NO_PROBLEM	UNSOLVE
ℹ	2003-10-22	takuyay	Should I have "New" button as well as "Enter"?	QUESTION	NOT_ASSIGN	UNSOLVE
ℹ	2003-10-22	takuyay	Do I need one more column to let users konw the cod...	QUESTION	NOT_ASSIGN	UNSOLVE
⊗	2003-10-22	takuyay	How can a code review be deleted?	DEFECT	FIX	SOLVED
ℹ	2003-10-22	takuyay	Should I add drag an drop function?	QUESTION	FIX	UNSOLVE
ℹ	2003-10-22	takuyay	Should auto update notificaiton be added?	QUESTION	FIX	UNSOLVE
ℹ	2003-10-22	takuyay	Double Click ME!	QUESTION	DEFER	SOLVED

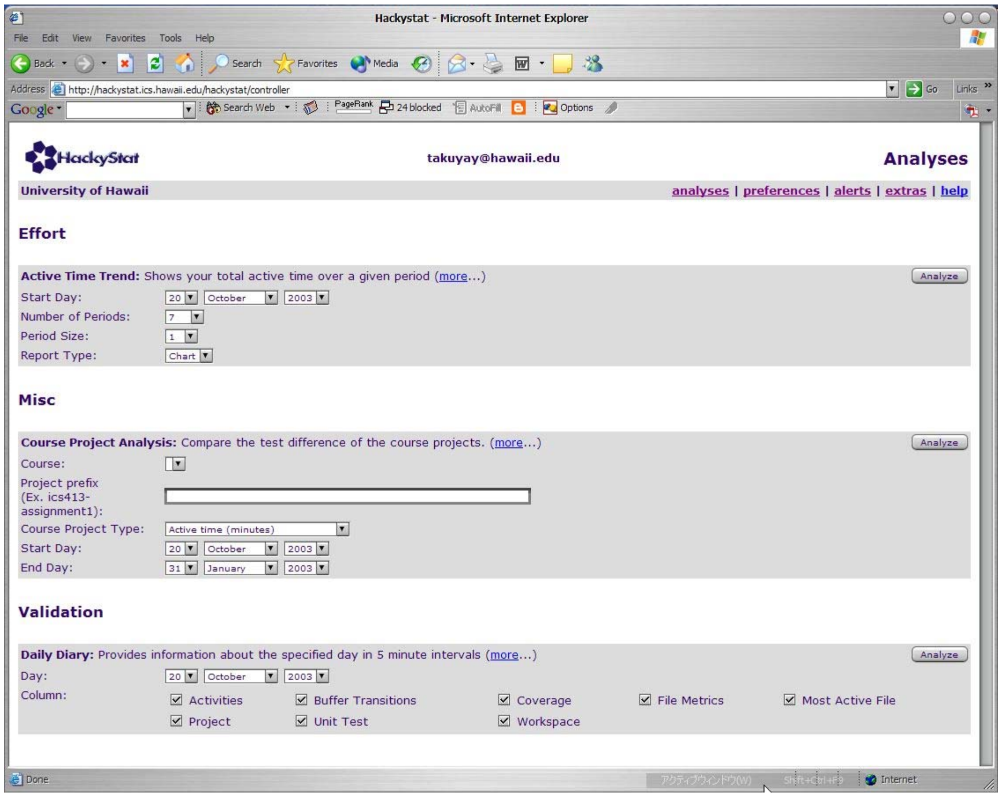
Hackystat Eclipse Sensor

The Hackystat Eclipse sensor currently can gather the active file a developer is working on, metrics such as CK metrics, JUnit test results. These data are sent to the Hackystat system server automatically in a certain interval, and stored there for some analyses. For this study, we will define additional new sensor data type to collect the necessary data. The new sensor data type would be consist of some sets of data such as structures of code (import statement, etc).

This sensor will contain the JUPITER sensor, and CVS sensor. The JUPITER sensor will gather the defects that have been recorded during review process. The CVS sensor will recorded what kind of change in a file occurred, and who was involved in making the changes.

Hackystat System

The server side Hackystat System provides a function to analyze data based upon some rules that developers defined as well as the function to receive data from sensors and store the data. For this study, we will add additional new analysis page of the JUPITER project. This page will contain the rule-base definition part to determine WHAT, WHEN, WHO, HOW questions. After the determination, The Hackystat system analyzes data depending upon the determinants, send developers to know when it judges that a new review should be initialized.


Hackystat - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://hackystat.ics.hawaii.edu/hackystat/controller
Go Links
Google Search Web PageRank 24 blocked AutoFill Options
HackyStat
Analyses
analyses | preferences | alerts | extras | help
Analyze
Done
Internet

takuyay@hawaii.edu

University of Hawaii

Effort

Active Time Trend: Shows your total active time over a given period (more...)

Start Day:	20 October 2003
Number of Periods:	7
Period Size:	1
Report Type:	Chart

Misc

Course Project Analysis: Compare the test difference of the course projects. (more...)

Course:	
Project prefix (Ex. ics413-assignment1):	
Course Project Type:	Active time (minutes)
Start Day:	20 October 2003
End Day:	31 January 2003

Validation

Daily Diary: Provides information about the specified day in 5 minute intervals (more...)

Day: 20 October 2003

Column: ☑ Activities ☑ Buffer Transitions ☑ Coverage ☑ File Metrics ☑ Most Active File ☑ Project ☑ Unit Test ☑ Workspace

Decision-making Components

The optimal software review will be determined by the rule-based components, which can be configured to the need of the particular organization and development context. The rules can be developed to determine the answers to the questions of the optimal software review underlying the review process. For the sake of developers, the following determinants will be prepared as a default. The answer will be initialized depending upon the organization's preferences. It will differ if the preferences differ.

1. What artifacts (i.e. files) should be reviewed?

* All artifacts.
* Only artifacts that have had defects.
* All artifacts that have been changed since the last time they were reviewed.
* All artifacts above a given complexity threshold.

2. When should an artifact be reviewed?

* When there are sufficient reviewers available to perform the review.
* When the artifact in question is no longer under modification.
* When too many defects have been posted against this artifact.
* When the artifact has been extensively modified.

3. Who should be involved in review?

* All developers involved with this project.
* All developers who have edited this file.
* Any developer with experience with the APIs used in this file.
* Developers who are not otherwise occupied in review.

4. How should the review be conducted?

* Developers should look for any errors.
* Developers should look for specific defect types.
* Different developers should look for different defect types.
* Developers should get as much time as they need to review.
* Developers should allocate only a specific number of hours to review.
* Developers should work until a specific date and time on the review.

* The review should be conducted until a threshold level of reviewers that have finished their review.
* The review should be conducted until a threshold level of defects that have been found in the artifact.
* The review should be conducted until a threshold level of size that has been recorded.

Target Subjects

The subject to be experimented would include around 20 ICS414 undergraduate students plus around 20 ICS613 graduate students, Spring 2004 enrollment. The target subject would be assigned into group f size 2 each for 10 groups for each class. All students would be supposed to use all the tools that are described in the previous section.

Target Objects

Target objects would be based upon the any materials inside the system module. Not only source code and documentation but also javadoc, html source code, code formatter error would be the object for the study.

1.4 Experimental Method

The objects (students in both ICS 414 and ICS613) are supposed to use JUPITER plug-in for Eclipse in the process of code review as well as Eclipse IDE. Any other tools are not allowed for the experiment. We will prepare our own preference settings of the determinants to be explained for the objects, and gather and analyze JUPITER project-related data with the Hackystat system.

The questionnaires are provided for students before and after this experiment in order to determine the usability of this system and how optimally the system initializes the software review based upon the rule.

1.5 Experimental Procedures

The subjects have 4 months or 11 to 12 weeks to study Software Engineering in a semester. Actual assignment would include non-programming materials as well as programming materials so that only programming related materials are experimented. First some hours would be spent for explanation of review process method and JUPITER project. For the student motivation, review process would be graded as credit, and the review understanding would be evaluated as two extra credit quizzes.

1.7 Experimental Limitations

One potential limitation in this experimentation is insufficient number of experimental subjects and size of the experimental project group. Due to the number of class (only one class for each academic level), different academic level (undergraduate, graduate), and group size (each module will be development by at most 3 developers), the result of questionnaire would be affected from the factors.

However, I hope that the result of the evaluation of the JUPITER system helps many developers to understand the rule-based code review optimization and use it in regular basis for open source software development communities. I hope this tool is used to investigate software engineering field as well.

1.8 Anticipated Results

I anticipate that the experiment would provide a great evaluation of the usability of JUPITER system from some questionnaires. I hope that there exists the positive evaluation of it and more convenience to use the system repeatedly.

If I can not find any significant value for the system, the definition of the thesis design must be reviewed and need some replicated experimentations in the future.