

EMPIRICAL SOFTWARE ENGINEERING USING SOFTWARE INTENSIVE CARE
UNIT

A THESIS PROPOSAL SUBMITTED TO MY THESIS COMMITTEE

MASTER OF SCIENCE

IN

INFORMATION AND COMPUTER SCIENCES

By
Shaoxuan Zhang

Thesis Committee:

Philip M. Johnson, Chairperson
Henri Casanova
Scott Robertson

September 4, 2009
Version 0.5.0

©Copyright 2009

by

Shaoxuan Zhang

To myself,
Shaoxuan Zhang,
the only person worthy of my company.

Acknowledgments

I want to “thank” my committee, without whose ridiculous demands, I would have graduated so, so, very much faster.

Abstract

Abstract goes here.

Table of Contents

Acknowledgments	iv
Abstract	v
List of Tables	vii
List of Figures	viii
1 Introduction	1
1.1 The Problem	1
1.2 Software Intensive Care Unit Approach	1
1.3 Thesis Claim	1
1.4 Evaluation	1
1.5 Thesis Structure	1
2 Related Work	2
2.1 TSP/PSP	2
2.2 Research Based on Automated Data Collection	2
2.3 Commercial Dashboards	2
2.4 Previous Classroom Studies of Hackystat	2
3 Hackystat	4
3.1 Hackystat Framework	4
3.1.1 Daily Project Data Analysis	4
3.1.2 Telemetry Analysis	4
3.2 Project Browser and Wicket	5
3.3 Software Intensive Care Unit	5
4 Design and Implementation of Software ICU	6
4.1 Software Intensive Care Unit	6
4.2 Vital Signs	6
4.2.1 Vital Sign from latest value	7
4.2.2 Vital Sign from historical trend via spark-line	7
4.2.2.1 Stream Trend Evaluation	7
4.2.2.2 Participation Evaluation	8
5 Classroom Evaluation	9
6 Contribution and Future Directions	10
A 2008 Classroom Evaluation Questionnaire of Hackystat	11
B Results form 2008 Classroom Evaluation Questionnaire of Hackystat	14
Bibliography	28

List of Tables

Table

Page ■

List of Figures

Figure

Page ■

Chapter 1

Introduction

This paper

1.1 The Problem

In most of the software project development cases, software metrics are one of the essential tools for performance measuring and quality control. But they are not comprehensive enough to make judgement with only one or two. So, we need to utilize multiple metrics in order to acquire insight of the health state of the software project. The more software metrics we look into, the more comprehensive insight we will get, but also the more effort it will need to collect measure data and interpret metrics values. My research tries to overcome this challenge by developing a system to help observer the health state of the software project from multiple software metrics in an effective way.

1.2 Software Intensive Care Unit Approach

What is SICU

1.3 Thesis Claim

We claim it is the most powerful tool ever in the world! =P

1.4 Evaluation

We evaluate in a classroom.

1.5 Thesis Structure

Here is a paragraph that saying the same thing as TOC.

Chapter 2

Related Work

This chapter presents some related work of my research. Four parts of relative work will be discussed in the chapter.

First part discusses previous research of empirical software engineering concepts. Most of previous research of measurement-based software engineering are focus on the methodology. Effective approaches are developed and deployed to actual practice. However, the lack of automation data collection adds significant overhead to developers, thus lead the impression that they are hard.

Second part discusses some recent researches start to focus on automated data collection. But they are only focus on introduction level programming course and lack of system extensibility, which means no use to senior software development or professional settings.

Third part discusses some commercial “dashboards” for software project data. Software ICU is, of course, one example of a project dashboard. However, it differs from commercial approaches with its intensive metrics, high extensibility and open source development and distribution.

Four part discusses two previous classroom studies of Hackystat system.

2.1 TSP/PSP

The Personal Software Process(PSP)[?] and the Team Software Process(TSP)[?], created by Watts Humphrey, are among the most extensively studied approaches for measurement-based software engineering.

2.2 Research Based on Automated Data Collection

Project ClockIt and Retina are two recent researches based on automated data collection to support programming courses.

2.3 Commercial Dashboards

2.4 Previous Classroom Studies of Hackystat

The classroom study presented in this thesis is the third case study of Hackystat system in a classroom setting.

The first case study is performed in 2003 using an early version of Hackystat.

The second case study happened in 2006 as a partial replication of the first case study.

In 2007, Hackystat has been re-implemented to a new architecture version, using a service-oriented architecture. The case study presented in this thesis is based on this newest version. Since the system has changed dramatically from either 2003 or 2006, some questions in the survey of the case study change to better suit the current situation.

Chapter 3

Hackystat

In this research, I utilize Hackystat to implement the Software ICU. This chapter briefly introduce the Hackystat system, which was invented by Professor Philip M. Johnson, in the Collaborative Software Development Laboratory, Department of Information and Computer Sciences, University of Hawaii at Manoa.

3.1 Hackystat Framework

Hackystat is an open source framework for collection, analysis, visualization, interpretation, annotation, and dissemination of software development process and product data.

Hackystat users typically attach software 'sensors' to their development tools, which unobtrusively collect and send "raw" data about development to a web service called the Hackystat SensorBase for storage.

The SensorBase repository can be queried by other web services to form higher level abstractions of this raw data, and/or integrate it with other internet-based communication or coordination mechanisms, and/or generate visualizations of the raw data, abstractions, or annotations.

3.1.1 Daily Project Data Analysis

The DailyProjectData(DPD) service is one of Hackystat's most important fundamental analysis service. From the raw sensor data in the SensorBase repository, it creates various abstractions of sensor data associated with a single project for a single 24 hour period, which usually represents a simple software development metric in a single day.

3.1.2 Telemetry Analysis

The Telemetry service is another fundamental analysis of Hackystat. Based on data from DPD service, it supports the creation of trend lines that show how various characteristics of software development are changing over time. To support the work practices of different organizations, it provides a domain specific language that allows the creation of custom trend lines (called telemetry "streams") and their visualization together in a specific telemetry "chart".

Telemetry streams support various numbers of parameters. User can use them to generate more specific streams. In our Project Portfolio Analysis, which based on Telemetry service, user can configure the parameters of each Telemetry analysis. More detail will be discuss in later part.

3.2 Project Browser and Wicket

Wicket is one of the dozens of Java web frameworks, but a outstanding one. It use HTML attributes to denote components, enabling easy editing with ordinary HTML editors. The internal object structure is similar to Swing, it give easy but powerful way to develop functionality on both server and client side.

Upon it we built a web application UI to view collected data and run various analysis provided by Hackystat services, that is called Project Browser. Its goal is to simplify the usage of Hackystat service as well as provide better visualization over the analysis data.

3.3 Software Intensive Care Unit

The Software Intensive Care Unit(SICU) consist a table of metrics over each of the projects. Each metrics are shown with a spark-line and a number value, each of which will be colored conditionally. The spark-line is generated using the Google Chart API.[5]

Chapter 4

Design and Implementation of Software ICU

4.1 Software Intensive Care Unit

In order to utilize the well developed software engineering measures to achieve good management of software development projects, we introduce a system called Software Intensive Care Unit. It consists of a number of software development metrics and uses these metrics to generate a number of vital signs that state represent "health" state of software development projects.

4.2 Vital Signs

Vital signs of software projects are measured by various software development process or product metrics. Each of these metrics reveals an aspect of the state of the software project.

Coverage Coverage is a measure used in software testing. It describes the degree to which the source code of a program has been tested. It measures the quality of the tests, which is an essential part of program quality insurance.

Cyclomatic complexity Cyclomatic complexity, was developed by Thomas J. McCabe and is used to measure the complexity of a program. It directly measures the number of linearly independent paths through a program's source code.[7] Higher cyclomatic complexity means more distinct control paths in a program module, thus it is more difficult to fully test them and the untested paths may have deficiency inside. Therefore, program modules are preferred to have lower complexity.

Coupling Coupling, or say dependency, is the degree to which each program module relies on each one of the other modules.[2] It is actually a measure of the complexity of the whole system's module reference tree. Whenever one module is modified, there will be a chance that the changes may cause bugs in one of the modules that relies on this one. Therefore, higher coupling value implies higher risk of introducing bugs when making changes, thus more difficult to maintain the system.

Churn Churn is a measure of the total number of lines of code deleted and added. It represents the amount of changes made to the system, which reflects the work to the system. Interpretation

of this metric is conditional. Low churn means low work load to the system, but also means consistency of the system, thus is good for systems under maintenance. However, high churn implies higher inconsistency of the source code of the system, but also means high work load, which is the normal case of a intensively developing project.

Size The size of software program is measured by the source lines of code (SLOC), which counts the number of lines in the text of the program's source code. SLOC is typically used to predict the amount of effort that will be required to develop a program, as well as to estimate programming productivity or effort once the software is produced.

Commit Commit measure the number of commitments made to the source code repository. It is recommended that programmer should make commit often and make it small and keep the integration consistent.

DevTime DevTime, abbreviation of Develop Time, is a measure of total human work put into the system.

Build Build is a count of ant build task invoked in a period of time.

Test Test is a count of unit tests invoked in a period of time.

4.2.1 Vital Sign from latest value

Latest values of software metrics are important enough to be shown separately because they represent the latest state of a project. We assign colors to latest value of the measures to indicate their performing.

4.2.2 Vital Sign from historical trend via spark-line

Historical data of software metric is as good as the latest one. It provides more information of the performance of the project over time. But in the same time it bring a great amount of data overwhelming as well. In order to reduce the historical data to an acceptable small but meaningful degree, we use spark-line to represent them. Then we assign colors to the spark-lines with several evaluation strategies: Stream Trend and Participation.

4.2.2.1 Stream Trend Evaluation

Stream trend evaluation determines the health of a stream by its trend. It takes one parameter called HigherBetter. If user define the higher value the better, then increasing trends will be considered as a healthy trend and decreasing trends will be considered as unhealthy. Stable trends are always considered as healthy because in that case it is as good as health that one don't need to pay too much attention to it, while the actual value will be shown in the latest value where the value will be judged to be good or not. And unstable trend is marked as average because it is not easy to tell if it indicates a good state or not.

4.2.2.2 Participation Evaluation

Participation evaluation determines the health of a stream by the participations of all members in the project. When working as a group, it is not good that some of the member do most of the work and others do little, thus in this situation the stream will be classified as unhealthy stream, which mean to attract attention.

Chapter 5

Classroom Evaluation

We evaluate this system in a classroom setting. The class contain 20 students. They are learning software development methods in the class and will use the system for about a month till the end of the semester. During this period, they work as small group of 3 4 people, and each group will work on 2 software projects.

We gather research data in two ways: 1. We will monitor and log their usage of the system. From this data we can find out their frequency and habit of using the system. And compared with their performance of the project, we can analysis the helpfulness of the system in developer's aspect. 2. We will give out survey at the end of the semester to gather their opinions of the system. From this we can find out users' attitude towards the system. In additional, they may provide insightful suggestion to improve the system.

We will analyze this data along with the data from previous Hackystat studies.

Chapter 6

Contribution and Future Directions

Appendix A

2008 Classroom Evaluation Questionnaire of Hackystat

Hackystat Evaluation

Hackystat is a long term research project concerned with improving the effectiveness and efficiency of software engineering metrics collection and analysis. Since 2003, we have periodically conducted a survey of students in ICS software engineering classes to assess the current strengths and weaknesses of the system.

To preserve anonymity, while also ensuring that only ICS students respond and respond only once, we ask you to provide the "secret code" that you randomly selected in class. To enable credit for completing this evaluation, only the graduate student researcher on this project (Shaoxuan Zhang) will know which code corresponds to you. He will provide a list of names who should be awarded credit to the class instructor without identifying individual responses. You can also contact Shaoxuan if you want your data deleted from analysis after you've submitted it.

If you want to go back and change your responses, simply fill out the entire form again. We will discard all but the most recently submitted entry for a given code.

This survey contains 17 questions and we expect that you will need about 10 minutes to complete it.

Thank you very much for your help! We take your views very seriously: prior responses to this survey have led to far-reaching changes in Hackystat.

Before filling out this questionnaire, you might want to take a look at the following image for the Software ICU to refresh your memory:

<http://csdl.ics.hawaii.edu/~johnson/portfolio.gif>

** Required*

*1. Installing the Eclipse IDE sensor was: **

- Very Easy*
- Easy*
- Neither easy nor difficult*
- Difficult*



- *Very Difficult*
2. *Installing the Ant sensors (JUnit, SCLC, Emma, etc.) was: **
 - *Very Easy*
 - *Easy*
 - *Neither easy nor difficult*
 - *Difficult*
 - *Very Difficult*
 3. *Please provide any feedback you can on the problems you experienced during sensor installation and server configuration, as well as any suggestions you have to make this easier in future.*
 4. *The amount of overhead required to collect Hackystat data (after successful installation and configuration of sensors) was: **
 - *Very Low*
 - *Low*
 - *Neither low nor high*
 - *High*
 - *Very High*
 5. *The amount of overhead required to run Hackystat analyses was: **
 - *Very Low*
 - *Low*
 - *Neither low nor high*
 - *High*
 - *Very High*
 6. *Please provide any feedback you can on Hackystat overhead, as well as any suggestions you have to reduce the overhead in future.*
 7. *Did you encounter any problems while collecting data? Was there any kind of data that you failed to collect? If yes, please explain.*
 8. *How did you feel about sharing your software development data with other members of the class? **
 9. *How frequently did you use the telemetry page? **
 - *Every day or more*
 - *2-3 times a week*
 - *Once a week*
 - *Less than once a week*
 - *Never*
 10. *If you used the Telemetry page, what were you trying to find out?*
 11. *How frequently did you use the Software ICU? **
 - *Every day or more*
 - *2-3 times a week*

- Once a week
 - Less than once a week
 - Never
12. *If you used the Software ICU, please check the vital signs that were useful to you.*
*
- Coverage
 - Complexity
 - Coupling
 - Churn
 - Size
 - DevTime
 - Commit
 - Build
 - Test
 - None of the above
13. *Did you feel the Software ICU colors accurately reflected the "health" of your project? If not, why not? **
14. *Were you able to use the Software ICU to improve your software's quality and/or your team's process? If so, in what ways? If not, why not? **
15. *Please provide any other feedback you would like regarding Telemetry and the Software ICU, as well as any suggestions you have on how we can improve the system.*
16. *If I was a professional software developer, using Hackstat at my job would be: **
- Very feasible
 - Somewhat feasible
 - Neither feasible nor infeasible
 - Somewhat infeasible
 - Very infeasible
17. *Please provide any other feedback you can on the feasibility of Hackstat in a professional setting, as well as any suggestions you have on how its feasibility could be improved.*

Appendix B

Results form 2008 Classroom Evaluation Questionnaire of Hackystat

This section presents the responses from the respondents to each of the questions. For the "short answer" questions, I corrected misspellings and minor grammatical errors to improve readability.

Question	Response
1. Installing the Eclipse IDE sensor was: <ul style="list-style-type: none">• Very Easy• Easy• Neither easy nor difficult• Difficult• Very Difficult	 figures/Q01-InstallEclipseSensor.eps
2. Installing the Ant sensors (JUnit, SCLC, Emma, etc.) was: <ul style="list-style-type: none">• Very Easy• Easy• Neither easy nor difficult• Difficult• Very Difficult	 figures/Q02-InstallAntSensor.eps

3. Please provide any feedback you can on the problems you experienced during sensor installation and server configuration, as well as any suggestions you have to make this easier in future.

- I could not figure out what step makes a .hackystat directory. My .hackystat directory automatically generated in my Documents and Settings directory which has a blank space in directory name. I am still not sure how to move this folder to other. The installation of all sensors was pretty well described at the project homepage and there was no problems I have met during the installation.
- Both the installation and sending sensor data was easy. However, tracking down whenever there is a problem with the sensor is not so easy. A troubleshooting page in the near future?
- Installing the sensors was pretty straightforward. I didn't have any problems.
- Case sensitivity was one problem between user and Hackystat, but it was fixed.

If it is possible to have a .EXE that will automatically create environment variables and also install files into a local directory will be awesome.

- I did have one small hang up when installing the Ant sensors: If I remember correctly I was getting a NoClassDefinition error whenever a sensor ran. I was running java 1.5. I fixed it by downloading the jaxb libraries since the errors were referring to that. It could be not related to jaxb at all, but it worked after that. Otherwise, I had no problems whatsoever installing the sensors.
- Everything went smooth with the instructions given and the verification after each step.
- Personally I didn't run into any problems but some of the other students did. The sensors aren't difficult to install per se, but there are a lot of steps involved and it's easy to get lost while installing them. Maybe an automated installer can be created that searches for the Ant tools (maybe the user can provide a search directory) and will configure and install the sensors for the user.
- What made it hard was that all the instructions were not in one page. I had to go from one page to another and then to another. There should be instructions from STEP 1 to the end and provide proper links to the step by step process.
- First of all, the manual is too long. I do like your goal to analyze the software project, but if it wasn't required by this class, maybe I wouldn't think I want to use it, because it looks too complicated.

Also there are too many things that we need to download and install. If you want to encourage people to use this more, maybe you should provide a package of all the tools somehow.

For example, before it took a long time to install Apache, MySQL, PHP, and Perl, but now somebody offers a package called XAMPP, which is a combination of all of those, and entire installation finishes in 3 minutes. Something like that should be given.

- There is a lot of documentation in a lot of different places. It was confusing trying to figure out what to read in what order, and whether or not it was relevant to me.

- Some the installation instructions could benefit from "write once, use many times" as they're repeated, which causes some people to start glossing over the instructions and then there's a couple that are slightly different and people (like me) won't notice the difference.
- The walkthrough was great, which made the installation easy.
- The only problem I had was the installation of the ant sensor. I mean configuring it on Eclipse was easy especially when I try to run Emma, JUnit, FindBugs and all that from Eclipse it is sending stuff to Hackstat but when I checked my software ICU I didn't have any data on Build (all it says was N/A). And little did I know that when you run the ant sensors on Eclipse it only registers all the data to Hackstat JUnit, Emma, Checkstyle and such except BUILD. And I was told that running the BUILD on the command line works but not on Eclipse. So I tried that and YES that works. So is there a way to make it work on Eclipse when you run all the ant sensors and it sends all the data to Hackstat including the BUILD data?
- When we ran the svn sensor, the build would fail if there are any commits from members not identified in our local Usermap.xml. Instead of looking for all commit records from all users within 24 hours, perhaps it could filter out and only look for records inside our UserMap.xml.
- The installation documentation must be read carefully. It may be easier to create a hackstat.build.xml with all the build targets, then import that file into each *.build.xml and call the sensor from the tasks.
- The most challenging sensor to get up and running was the SVN sensor. Other than that, the others seemed fairly easy to install.

Question	Response
<p>4. The amount of overhead required to collect Hackstat data (after successful installation and configuration of sensors) was: *</p> <ul style="list-style-type: none"> • Very Low • Low • Neither low nor high • High • Very High 	<p>figures/Q04-DataCollectOverhead.eps</p>

Question	Response
<p>5. The amount of overhead required to run Hackystat analyses was: *</p> <ul style="list-style-type: none"> • Very Low • Low • Neither low nor high • High • Very High 	<p>figures/Q05-AnalysisOverhead.eps</p>

6. Please provide any feedback you can on Hackystat overhead, as well as any suggestions you have to reduce the overhead in future.

- Since the verify command runs all the tests, I'd think that it should send data for all tests run. Rather, in the portfolio analysis, the Unit Test portion only retrieved data for any JUnit builds that were run. It doesn't really make sense why we'd have to run it separately when verify does it anyway.

- If I am correct, overhead - the processing time required by a device prior to the execution of a command. Then it all depends on what computer the user is using, I am using a single-core processor laptop it did not take long.

- Since Dr. Johnson provided us with Ant sensor examples, it was quite easy to set up everything to send data to the sensorbase. I did the hackystat tutorial and everything worked fine. However, I missed the part about creating a usermap.xml file for the svn sensors through Ant. That confused me a bit later on but I figured it out.

What made getting data quite easy as well was having Hudson installed on a dedicated continuous integration server. Daily builds would auto-send data to Hackystat and this made it super easy to get daily info.

- The sensors ran automatically and it was fast with sending the data.
- Maybe there can be a link on <http://dasha.ics.hawaii.edu> to both the Hudson and Hackystat server, that way we don't have to memorize the port numbers. Also, allowing us to create an account and password would go a long way towards usability. I had to put the Hackystat login information in a text file because I can't remember a randomly-generated string for the password.
- Sending sensor data was often quite slow. Generating reports in the web application was sometimes also slow – the page wouldn't load until you refreshed it.
- The overhead to collect data was generally small, however long enough that would generally run multiple (DOS) terminals so that I could continue working while it was sending data. Analysis was no overhead since that was just pulling up a browser page.

- When sending hackystat data, it was fairly quick on my computer, MacBook Pro. Tho, there were some students I saw which had a LONG wait time on the same laptop.
- I love Hackystat! It is a very great tool especially for a developer like me.
- Since Ant takes care of running Hackystat sensors, this made it very easy to accomplish.

7. Did you encounter any problems while collecting data? Was there any kind of data that you failed to collect? If yes, please explain.

- I had a problem with sending commit data to hackystat when I worked on a group project. That was because I did not update my sensors to newer version.
- At first during the implementation of DueDates 2.0, it was not collecting commit data from my account. It was due to the account on hackystat, it included the @gmail.com part of my gmail account. So it was not matching up with each other, the hackystat account and my gmail account.
- Running an analyses on my machine was slow, it would take over 3 minutes to run a build. I am not sure why it took so long to send the build data so I can't make a suggestion.
- Only JUnit data as mentioned previously.
- Case sensitivity was an issue at first, but it was corrected so I did not get problems after that. Hudson did not send to Hackystat number of commits, but that was fixed after a little modification with build.xml file.
- I was lucky. I rarely had any problems collecting data during all the time I worked with Hackystat. The one time something got screwed up was with my development time for one day. It said 0 when I checked and I had put in a bunch of time that day so it should have said otherwise.

I don't remember exactly but, that night I believe had worked in eclipse till after 12 at night, so it went to the next day before I closed the program. That could possibly be a reason for the missing data initially. The next day I just cleared the cache and it was all fine.

- There was a small issue when I first started collecting data, but it was quickly corrected when checking the xml files.
- Personally I ran into no problems collecting data.
- Sometimes it didn't collect build data for some reason.
- Occasional problems with SVN collection, I think, was a bit hard to tell.
- Everything was great except collecting data for my BUILD (please refer to above statement for more detailed problem regarding this). Thank you.
- I did with commit records but it was my fault. I wish subversion with Google Project Hosting would be more strict. I was able to check out the project with or without the "@gmail.com" suffix (i.e. "test" and "test@gmail.com"). Thus making me two different authors.

- Yes, the build data. I needed to set more environmental variables.
- For some unknown reason, my user name picked up the @gmail.com, so both my user name with and without @gmail.com needed to be added to the projects.

8. How did you feel about sharing your software development data with other members of the class?

- We could see how other groups were doing by sharing our software development data with other people. We also could find out what kinds of problems with our project by comparing graphs with other groups and this helped a lot.
- I was not offended if it was low, and was quite intrigued with others data.
- I did not have a problem with sharing data with other people in class. I thought it was needed tool to keep tabs on everyone to assure they're doing their fair share.
- It felt good if your data was better than others. And if it wasn't, then you felt bad.
- Did not really like it because it is showing my programming habits, like starting on a project on the last couple of days.
- I felt alright about sharing my data with the class. It was interesting for me to see how other people worked on stuff. Some were consistent and others were not. Some people spend a lot of time working on stuff yet do not commit as much as others that work half the time. I think its good to see this data.
- I am okay with sharing my data.
- I didn't think it was a particularly good idea because it then forces group members to become competitive with each other, especially if one person is able to put in more time than all the others. Also, the data doesn't reflect the amount of work put in, maybe someone spent 5 hours doing research and only 1 hour programming, but the sensor data will only show 1 hour of development time and a minor code commit, versus someone who, say, just changes around the package structure for 3 hours and has a huge commit amount.
- Actually hackystat (or hacky-stalk as what my teammates and I called it) caused a lot of arguments and trash talk. Some guys were more concerned about collecting stats on hackystat than actually finishing the project. Some members would start competing on who had more commits or move development time. The project turned out to be more of a competition of stats, which wasn't healthy for the team at all.
- It will be obvious that who worked on the project, so it is nice in terms of grading students. At the same time I feel some pressure that I need to work on the development, so if team leader require everybody to work well, this is good.
- Didn't really care.
- I had no problem with this, and it encouraged me to be aware of my time management and coding style.

- It was good in a sense that they can help you with test cases and coverage.
- It was fun..because you can see how everyone is doing within your group.
- Before taking this class, I didn't think that there was a way to track software development process. After learning about software continuous integration and working in a larger group project, I have a better insight in sharing the development process. I feel that it is a must in every software development environment, big or small to be able to communicate frequently and effectively.
- I was nervous because certain individual of the class seemed able to put in ridiculous long hours. I was concerned my amount of time (which seemed reasonable) would make me look as though I'm not working as hard.
- Good, I can see how I and others rank with each other.
- I am fine with this. All group projects in all schools (e.g., Architecture) should be required to use such a system. This is great for facilitating fair evaluations of students who participate, and those who 'get the grade' by riding on the laurels, blood, sweat, and tears of others.

Question	Response
<p>9. How frequently did you use the telemetry page? *</p> <ul style="list-style-type: none"> • Every day or more • 2-3 times a week • Once a week • Less than once a week • Never 	<p>figures/Q09-telemetryFrequency.eps</p>

10. If you used the Telemetry page, what were you trying to find out?


- I tried to find out how was I doing for the project by looking hackstat data.
- Seeing how much time i spent on the development of the program, and also others in my group.
- When I used the telemetry page I was trying to find out if I was on par with other groups members in terms of development, build, and commit numbers.
- Whether or not, my sensors were reading, and the work output of my group members (especially on days we didn't meet together).
- If my development time was up to par with my team members.

- I usually used the telemetry page to evaluate how my team was working overall, and what my part was in that data. I also checked it to make sure everyone's data was being sent.
- It helps me see how I measure up with my partners.
- Member dev time mostly, to compare the amount of development time I put in vs. my group members.
- It supposed to show us how healthy individuals are in the group. So if one person is slacking, the members need to tell him to step it up. It wasn't used that way in our group. One person really wanted a good grade for the class so he just used the telemetry to watch himself; making sure no one gets more builds/devTime/commits than him (yes he said "i need more dev time because i need an A"). I remember we had dinner as a group and one of our group members didn't go to dinner. another group members then said "oh if he ups his stats more than mine, tomorrow I'm gonna hack all day."

Sad, but true.

- member commit, member dev time
- Curious about trends in dev time, commits.
- Usually MemberDevTime, MemberBuilds, and MemberCommits. Basically just seeing how everyone was progressing.
- graphs, line trends of other group members
- My status and the status of our group and make sure everyone is doing their part.
- Mostly trends in individual performance, as well as overall project outlook.
- Basically if everyone was putting in the same amount of effort. Also it helped indicate if everyone is on track. If they have regular activity, then the chances of them on track is higher.
- Was the coverage, complexity and coupling getting bad?
- I tried to review each telemetry page daily to understand what I could do to improve the project health and focus efforts.

Question	Response
<p>11. How frequently did you use the Software ICU? *</p> <ul style="list-style-type: none"> • Every day or more • 2-3 times a week • Once a week • Less than once a week • Never 	<p>figures/Q11-ICUFrequency.eps</p> <p>21</p>

Question	Response
<p>12. If you used the Software ICU, please check the vital signs that were useful to you. *</p> <ul style="list-style-type: none"> • Coverage • Complexity • Coupling • Churn • Size • DevTime • Commit • Build • Test • None of the above 	 <p>figures/Q12-UsefulVitalSigns.eps</p>

13. Did you feel the Software ICU colors accurately reflected the health of your project?
If not, why not?

- I felt most of colors accurately reflected the health of the project. For the Coverage data, since we can write test cases just for increasing of the rates, we cannot assume that the project is in healthy condition even if the coverage data displayed in green color. However, I think this is not a problem of hackystat.
- Yes
- The only issue I had with the ICU colors was with the coupling. In both versions of DueDates we had to add extra classes at the last minute which would cause the coupling ICU to turn red. I am not sure how to address that because the coupling does need tracking.
- Not really, I don't think having a high churn amount is necessarily bad. Of course, it's a case-by-case thing. For my group, it wasn't about not committing frequently; we were just rehashing code because something just didn't work.
- Yes, reflected accurately on the health of the project. Showed how much coverage we had.
- I feel that the Software ICU did accurately reflect the health of my projects. For Due Dates 2.0, which was a longer project, the data was getting increasingly more meaningful as the trends were over a larger period of time. It is good to look at things like devtime, commits, coupling, and coverage to see the color and the past trend because i think they really say something about the current state of the project.

To make it simpler, whenever I knew our project wasn't doing good and people weren't working regularly, the software ICU would have lots of reds and yellows. When I knew the project was doing better and people were working regularly, there were greens. It makes sense.
- The ICU was accurate with our project because it showed drastic spikes in all signs. This reflects our project in poor health.

- Not particularly because a project's health cannot easily be determined by just measuring numbers alone. For example, it's easy to increase coverage, but if a class has nothing but getters and setters and a toString method, does it really need to be tested? Of course not, but someone might feel compelled to do it in order to increase coverage and get a better health, but it's just a waste of time in my opinion. Also, DevTime is only measured from Eclipse but that doesn't measure things such as someone reading a book or looking up websites for information. It only measures active development in one program, forcing people to only use whatever IDE's Hackystat supports. The figures for complexity and coupling are hard to evaluate too. We want complexity to be low but sometimes it's unavoidable for it to be high, and should Hackystat show an absolute cut-off point where the complexity must be below a certain point for the project to be considered acceptable? Coupling is another one that falls under this category, if your program relies on a lot of outside libraries, can someone really determine an absolute value that the project's coupling must be under?
- Yes.
- maybe
- Coverage: perhaps too sensitive to drops/bounces in coverage. Churn: while you're working on a project, churn is going to vary, sometimes a lot. The trend colors were not helpful.
- Yes, I felt it was a relatively healthy project, and this generally showed, in the end. In the first half the colors reflected not as health of a project, which I'd agree as well. I'm not sure rising coupling was entirely a bad sign as things went along and functionality was added, as it was a slow steady rise.
- Sometimes. Hard to determine what will fall into green, red, or yellow.
- Yes definitely.
- It somewhat reflected the quality of our project. Maybe in some dark corner something is not thoroughly being depicted through the colors. Perhaps a suggestion is to use different color hues.
- Yes it was pretty accurately reflected.
- No, since I did not correctly configure the sensors.
- This is subjective... Usually the colors were spot on, however, they are quick to turn one way or the other depending on events that are being managed by the team (e.g., large code churns due to removal of unused code/imported code, etc.).

14. Were you able to use the Software ICU to improve your software's quality and/or your team's process? If so, in what ways? If not, why not?

- We can check how other members are doing for the project through the Software ICU and this helps a lot especially when we are working on the team project.
- Yes, for tracking if members were working on their tasks. Also how complex the program is increasing or decreasing.

- In my opinion, it is not clear if the ICU improved our system. Because other tools such as junit, findbugs, and pmd was easier to use to improve the application.
- If anything, keeping an eye on coverage helped us look out for what was being tested and what wasn't. Yes, showed how much coverage we had, and improve on that.
- I think for sure the Software ICU improves team process. More than just keeping people "in check" when grades are at stake, it provides an accurate way to assess what's being done and by whom. Our team got a lot out of checking up on the software ICU and assessing our team process. It seemed to get better over time.

As far as the software's quality, I think the Software ICU could be very useful in improving this. If my project for instance was in the red for complexity and coupling, and there were some code issues, I could see all this automatically through hackystat. Besides coverage stats though, my team did not really use the ICU to improve the software's quality.

- ICU was able to help us because it told us what needs to be focused or corrected.
- Personally, I only found Hudson useful because it's like running your code on someone else's computer to see if your environment is set up differently from a generic machine. I feel that the data for Hackystat is more something to look at out of curiosity rather than something to determine how well a project's status is because it's hard to base a project's health based on numbers alone and it might put unrealistic pressures on the team to make the project healthy for Hackystat when they can better spend their time developing instead.
- Yes.
- Yes, coverage tells me if we didn't write enough test cases.
- No. Coverage: already aware from Emma. DevTime, Commit, Build, Test: either team members did not look at the statistics, or they didn't care, because their habits did not change much. Others: not much we could do about the other statistics.
- Yes because able to manage our time and development fairly equally, and also notice spikes indicating bigger changes or problems.
- Yes, shows were we could improve as a group and improve as a programmer.
- Like in my case last time, I saw on Software ICU that I don't have a data on my BUILD. So because of that information I know what the problem is and it helped me to find a solution and figure everything out before it is too late.
- Our project ICU definitely described our lacking and late attempt to improve coverage. Due to the ICU, we were able to distinguish this fact quick and easy.
- The amount of activity helped us identify who was falling behind. Without offending our members by outrageously claiming their not working, we could tell by the sensors. Members can be more self-critical by looking at their individual data compared to the groups.
- Yes, by checking the coverage, complexity and coupling.

- Yes. By targeting coverage, dev time, coupling, and complexity, my team was able to improve all these into areas that were acceptable to us.

15. Please provide any other feedback you would like regarding Telemetry and the Software ICU, as well as any suggestions you have on how we can improve the system.

- I do not think the commits, builds, tests should be colored in because it all depends on how much the user does on the project. Is it possible to show line coverage instead of method coverage? The software ICU and telemetry was awesome tools in helping out with the project. It gave me visual stats on the project.

- What I think would be cool is to implement something to view the trend for each category in larger format but in the same style as the software ICU. I know this is shown on the telemetry page when you select it to show. However, I would be nice if there was some sort of rollover function that brought up a slightly larger window with a blown up overall trend. I can see how this isn't really needed but I would mostly likely check it a lot if it was there.

A minor thing that I noticed when using the Telemetry page was that when I selected a new statistic to view, the page would always jump back to the top and I'd have to scroll down each time. Its not really a biggie, but it makes navigating a bit slower when your going through all the project statistics.

- Consistent colors for each members can help.
- In addition to everything I mentioned above, it might help to somehow make the sensors configurable in some way, for example if two people are doing pair programming, there should be an option to set the sensors to send data for both people. Perhaps complexity can be measured somehow to only include methods that, say, start with get or set and toString. This way people aren't forced to write pointless test cases in order to increase coverage.
- Help page should be provided inside project browser. It should describe how to use it, what telemetry, what churn is, something like that.

Also your explanation should be simple so that people want to read it. If it is complicated and long explanation, nobody will read it.

- The different color bars and randomness might be fun and interesting, but I think having a bit more consistent scheme might be better. I would suggest if possible giving each developer a specific color that they always have during the project, either random, or chosen at the beginning.
- Does not capture development outside of Eclipse. For example, IMHO, MS Visual Studio is much better in the capacity as a web development IDE, which the dev time here was not recorded.

Question	Response
<p>16. If I was a professional software developer, using Hackstat at my job would be: *</p> <ul style="list-style-type: none"> • Very feasible • Somewhat feasible • Neither feasible nor infeasible • Somewhat infeasible • Very infeasible 	<p>figures/Q16-FuturePredict.eps</p>

17. Please provide any other feedback you can on the feasibility of Hackstat in a professional setting, as well as any suggestions you have on how its feasibility could be improved.

- I think its good to have this in a professional environment, cause the employer or client can check on how the progress of the program is going. With out having to make so much visits or hovering over workers.
- Cannot think of any off of my head. The Software ICU is already great for us programming students.
- I think Hackstat is definitely feasible in a professional setting, as long as it is supported in some way. For instance, if a team of developers is working on a project and they are all for having Hackstat manage project stats, that would be great. If, however, your the only person on your team that wants to use it, then it would be hard to send data that would assess team process.

I could see project managers wanting to have Hackstat data to evaluate everyone's input into the project, as well as the health of the project. Hackstat, I think, is perfect for new open source projects if releases are made early and often. It could be essential to seeing the overall health of the project.

- Overall, I feel like Hackstat would be an interesting tool to gather data to look at for curiosity's sake from time to time, but it should not be used as a basis for determining a project's health or to determine something such as member contribution. The sensors can only gather information from a few sources and these readings cannot account for a person's full contributions to a project. As for determining a project's health, I do not believe the sensor readings can provide an accurate measurement because the sensors can only measure numbers based on algorithms, but it takes a person to really determine how good the code is.
- When I start to use hackstat, I need to get password from you and then eclipse send my data to your server. Some developers might have concern that hackstat steal source code.
- I think it depends a lot on the culture of job setting. I'm not too sure, but I think I may try setting it up on my own job site, even if just for myself to see my own trends.

- It is a very useful tool to keep track the health of a project so I would say it is feasible to have it in a job.
- My only wish is that ICU's should have a feature to support pair programming. Possibly a feature to indicate to the system that two people may be working on the same problem on the same system, rather than two individual machines. You might want to call this "collaborative mode," or something along the lines of that. These settings of course should be turned on or off easily from the developer's IDE (Eclipse).
- I work in a one person shop, so it would be difficult to say how useful this would be. As a lone developer, many metrics I am very cognizant of, however, having such a system would allow me to view those statistics that I do not have a "gut" feeling for. It would be great for my boss to measure the amount of time I spend on a project however.

Bibliography

- [1] Hlio R. Costa, Marcio de O. Barros, and Guilherme H. Travassos. A risk based economical approach for evaluating software project portfolios. *ACM SIGSOFT Software Engineering Notes*, 30(4):1–5, July 2005.
- [2] Coupling (computer science). [http://en.wikipedia.org/wiki/Coupling_\(computer_science\)](http://en.wikipedia.org/wiki/Coupling_(computer_science)).
- [3] Code coverage. http://en.wikipedia.org/wiki/Code_coverage.
- [4] Development governance for software management. <http://www.ibm.com/developerworks/rational/library/oct07/dunn/index.html>.
- [5] Google chart api. <http://code.google.com/apis/chart/>.
- [6] Watts S. Humphery. *A Discipline For Software Engineering*. Addison-Wesley Publishing Company, Inc., 1995.
- [7] Watts S. Humphery. *Introduction to the Teasm Software Process*. Addison-Wesley Longman, Inc., 2000.
- [8] Wiboon Jiamthubthugsin and Daricha Sutivong. Portfolio management of software development projects using cocomo ii. In *Proceedings of the 28th International Conference on Software Engineering*, pages 889–892, 2006.
- [9] Thomas J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, December 1976.
- [10] Source lines of code. http://en.wikipedia.org/wiki/Lines_of_code.