Proceedings of the

First Hackystat Developer Boot Camp

May 10-14, 2004

Honolulu, Hawaii

Table of Contents

Welcome Message

Aloha and welcome to the First Hackystat Developer Boot Camp!

There are many goals for this workshop. For developers new to Hackystat, this workshop allows them to get an overview of many facets of Hackystat in a short period of time. At the end of the week, you should have a good understanding of what Hackystat does, what is currently doesn't do, what we are attempting to make it do in future, and where your goals and ideas fit into the system. Perhaps, at the end of the week, you will decide that Hackystat is not for you, although we hope this doesn't happen! What we do hope is that you will get enough insight into Hackystat's architecture, design, implementation, strengths, and weaknesses that you can determine how best to enhance it to support your own software engineering measurement and analysis agenda. We also hope you will volunteer to give a talk on your own research and development activities at some point during the week so that we can learn from you as well.

For developers who are experienced with Hackystat, this workshop allows you with an opportunity to lead discussions about the areas of the system you have worked on and research areas you are currently pursuing. This can be very valuable—the mere act of trying to tell someone else what you are doing is often the best way to push an idea forward. Plus, I hope that you receive valuable feedback that helps you move forward more quickly in the future. Finally, your efforts in this workshop can help by publicizing what you are doing and what you are interested in. It may lead to new collaboration opportunities for you.

The talks are deliberately spaced out across the week in order to provide opportunities for hacking. We hope that you will leave not only with conceptual information, but also with some concrete running code that you can build upon after your time at the Boot Camp. This is a rare opportunity to write some Hackystat code and get immediate face-to-face feedback from the Hackystat implementation team!

Please let me know if there are ways that this Boot Camp can be improved for the future.

Philip Johnson

Schedule

Monday, May 10:

10:00am – 12:00 pm	Introduction to Hackystat, *Philip Johnson* Anatomy of the Hackystat Component Architecture, *Philip Johnson*
12:00pm – 2:00pm	Lunch
2:00pm – 4:00pm	Anatomy of HackyKernel, *Philip Johnson* Anatomy of HackyStatistics and HackyReport, *Philip Johnson*

Tuesday, May 11:

10:00am – 12:00 pm	Anatomy of hackyVisualStudio, *Qin Zhang* Anatomy of hackyOffice, *Burt Leung*
12:00pm – 2:00pm	Lunch
2:00pm – 4:00pm	Key standard extension concepts, *Hongbing Kou* Anatomy of the Ant/LOCC sensor, *Mike Paulding*

Wednesday, May 12:

10:00am – 12:00 pm	Anatomy of hackyEclipse, *Takuya Yamashita* Anatomy of hackyPerf, *Aaron Kagawa*
12:00pm – 2:00pm	Lunch
2:00pm – 4:00pm	Research directions: Improving HPC Development, *Mike Paulding* Research directions: Sequence Analysis for TDD evaluation, *Hongbing Kou*

Thursday, May 13:

10:00am – 12:00 pm	Research directions: Software Telemetry, *Qin Zhang* Research directions: Improving software development of MDS, *Aaron Kagawa*
12:00pm – 2:00pm	Lunch
2:00pm – 4:00pm	Research directions: Improving Hackystat Evaluation Quality, *Melissa Rota* Research directions: Improving software review, *Takuya Yamashita*

Friday, May 14:

10:00am – 11:00 am	Commercialization of Hackystat, *Philip Johnson*
11:00am – 12:30pm	Board Meeting, Kuhio Beach, Waikiki
12:30pm – 2:00pm	Lunch, Duke's, Waikiki

Hackystat Hacker Certification Exam

The Boot Camp presentations and Developer Guides give you a map to Hackystat, but a map is not the terrain. To actually understand Hackystat, you need to walk around inside it for a while. The following exercises provide you with an incremental introduction to the concepts and APIs needed to modify and enhance Hackystat to suit your own needs. If you successfully complete all of the following exercises, you will definitely achieve "certified" Hackystat Hacker status. In the questions that follow, references to Developer Services web site refer to http://hackydev.ics.hawaii.edu/. We also indicate when one exercise depends upon successful completion of another, and provide an estimated minimum time requirement for each exercise. If your required time for an exercise varies significantly from our estimate, please let us know so that we can revise the value.

1. Install Local Server. [30+ minutes] Download the CVS modules associated with the Hackystat-UH configuration onto your local machine from the Developer Services web site. Use anonymous download from CVS. Configure hackystat.properties, build the system, and perform a hot deploy to a running Tomcat web server. Run all of the Junit tests to verify your installation. If not all of them pass, explain why.

2. Install Local Sensor. [30+ minutes, depends on 1] Download an editor sensor (Emacs, Eclipse) from your local server and install it into your local machine. Now register with your local server, configure your sensor.properties file, and perform some editing task in your editor. Verify in at least three distinct ways that data has been sent from your editor to your local server.

3. Local Server Administration. [30+ minutes, depends on 1] Login to your local server as the administrator. Use the administrator interface to: (a) automatically register a new user; (b) display a list of all registered users on your local server, and (c) login as any of the current registered users' accounts.

4. Interactive Sensor Shell. [30+ minutes, depends on 1] Download sensorshell.jar from your locally running server. Read through the Sensor Shell Developers Guide linked from the Developer Services web site. Now bring up the SensorShell command line interface using 'java –jar sensorshell.jar'. Type 'help' to get a list of commands available in this version of SensorShell. Manually type in commands to add sensor data. Send the data to the server. Exit the SensorShell interactive session. Find the log file associated with this interactive session, and see what it says.

5. Offline Data Storage. [30+ minutes, depends on 4] Bring down the Tomcat server running Hackystat. Now bring up an interactive SensorShell session. Manually type in commands to add sensor data. Exit the SensorShell. What has happened to the data you added? Locate the file where your data is stored. Now bring up Tomcat and deploy Hackystat to it. Next, bring up another interactive SensorShell session. What has now happened to the data that you added from the previous session? Invoke the 'send' command to send the data to the running Hackystat server, and check to see that the data that you added from the previous SensorShell session has been sent to the server. Exit SensorShell. Look in the directory where your 'offline' data used to be stored. What is different?

6. Local sensor data collection: Junit. [15+ minutes, depends on 1] Enable the Junit sensor for Hackystat, and run all of the Junit tests on hackystat itself. Check to make sure that UnitTest data regarding this test run has been sent to your local server. How can you verify that the UnitTest data has been sent from the client-side perspective? How can you verify that the UnitTest data has been sent from the server-side perspective?

7. Local sensor data collection: Jblanket and LOCC. [15+ minutes, depends on 6] Enable the Jblanket and LOCC sensors for Hackystat. Run the Junit tests over Hackystat, and check to make sure that Coverage data has been sent to your local server. Now run LOCC over Hackystat, and check to make sure that FileMetric data on Hackystat has been sent to your local server.

8. Build reports: Hackystat coverage and software testing quality analysis. [60+ minutes, depends on 7] Generate the JavaDocs and the Java2HTML reports on your local Hackystat installation. Now look at the Jblanket report you generated in 7. What packages/classes do not appear to have good coverage? Refer to their design via the JavaDocs and the code via Java2HTML. From your analysis, what would be the top three areas of Hackystat that you would focus testing attention on in order to improve its quality? Do these areas correspond to the areas of lowest coverage or not?

9. Editor configuration for Hackystat development. [60+ minutes] Most of the remaining exercises involve developing new code. To support this process, you will want to configure an editor to build the Hackystat system or parts thereof. Our preferred editor for Hackystat development is Eclipse (though we have a historical fondness for Emacs and have also used Jbuilder in the past.) In this exercise, configure your editor to support compilation of Hackystat. Begin by reading the "Developer Quick Start" document at the Developer Services web site. Then ensure that you can recompile your local configuration of Hackystat within your editor. Ensure that you install Hackystat sensors into your editor, so that you can begin collecting metrics on your hackystat hacking activities!

10. Sensor Data Type definition. [3+ hours, depends on 5, 9] For this exercise, you will define a new Sensor Data Type called PairProgramming. Begin by reading the Sensor Data Type Developers Guide from the Developer Services web site. The purpose of the PP SDT is to gather data on when developers are pair programming, what code was involved in the pair programming, what tool was used to do the pair programming, and how long the pair programming session lasted.

- Your SDT will have the following fields: a StartTime field, an EndTime field, a Tool field, and a FileList field. (It will also have the automatically generated tstamp field, of course.)
- For this exercise, you should define this SDT inside of the hackyStdExt module in the package org.hackystat.stdext.pairprogramming.sdt.
- Design and implement the components of this package: PairProgramming.java, PairProgrammingShellCommand.java, TestPairProgramming.java, doc.sdt.pairprogramming.html, package.html, and sdt.pairprogramming.xml.
- Check to see that you can build your local server to include this new SDT, and that the unit test passes successfully.

11. Integration of new SDT definitions into SensorShell. [15+ minutes, depends on 10] Download the sensorshell.jar file from your deployed server that now includes the PairProgramming SDT. Bring up an interactive sensorshell session using 'java –jar sensorshell.jar'. Invoke 'help' and make sure that your sensorshell "knows" about PairProgramming. Now manually add some PairProgramming data, and send it to your local server. Check to see that this data has been received by your local server.

12. New component definition and integration into build process. [3+ hours, depends on 11] Defining the PairProgramming SDT inside of the hackyStdExt module is bogus. For this exercise, you will fix this by defining a new Hackystat component called hackyPP. The hackyPP component module will contain all of the code related to PairProgramming, including sensor data type, analysis command, and sensor code.

- In the directory containing all of your other Hackystat modules, create a new directory called "hackyPP".

- Now transfer your SDT code from hackyStdExt into this new module. Rename the package containing your SDT code to org.hackystat.app.pairprogramming.sdt while copying the code to the new module.
- Create a local.build.xml file suited to your module by copying the example version in the hackyCli module (this version of local.build.xml is very basic and sufficient for your needs. Note you will need to change references to hackyCli to hackyPP.)
- Next, edit your local version of hackyBuild/build.xml to include this new component module.
- Edit your hackystat.properties file to include hackyPP.available = true.
- Now compile, build, and test your system using Ant. Ensure that your local server is being built with your hackyPP module, and that the SDT is defined and available within the sensorshell.jar file generated with this configuration.

13. Daily Analysis and Daily Diary Extension. [3+ hours, depends on 12] It would be helpful to users if the Daily Diary analysis included the ability to indicate the occurrence of PairProgramming data. For this exercise, create a package in your hackyPP module called org.hackystat.app.pairprogramming.dailyanalysis.

- Create the files PairProgrammingData.java, TestPairProgrammingData.java, dailyanalysis.pairprogramming.xml, and package.html in this directory. Use the files in the package org.hackystat.stdext.unittest.dailyanalysis located in the hackyStdExt module as templates.
- Your implementation should support a new column in the Daily Diary called "Pair Programming".
- Each cell in this column will contain one of the following: (1) Nothing, if no PP session started or ended during this five minute interval; (2) "PP started" if a PP session started during this five minute interval, (3) "PP ended" if a PP session ended during this five minute interval, or (4) "PP both started and ended" if a single session started and ended within five minutes, or if one session started while another session ended.
- Build some predefined PairProgramming test sensor data and install it as part of the testdataset user's test data to facilitate the testing of this daily diary. The data should be stored in src/org/hackystat/app/pairprogramming/testdataset. See the unittest/testdataset directory for an example of its internal structure.
- Build, deploy, and test your Daily Diary enhancement. Log in as the testdataset user and display the daily diary manually to admire your handiwork.

14. Hackystat analysis design and implementation. [6+ hours, depends on 13] Let's now create a full-fledged analysis for PairProgramming data. For this analysis, we want to know how much time was spent pair programming during a given interval of time on a given project. Begin by reading the "User Interface Developers Guide" available from the Developer Services web site.

- You will call this command "Pair Time", and it will be implemented in the org.hackystat.app.pairprogramming.pairtime package. It will contain the files PairTime.java, TestPairTime.java, PairTime.jsp, command.pairtime.xml, doc.pairtime.html, and package.html.
- Your command will include the following selectors: IntervalSelector (which allows the user to specify the interval and grain size of time over which they want the data), ReportTypeSelector (which allows the user to specify whether they want the data as XML, CSV, an HTML table, or a JfreeChart), and ProjectSelector (which allows the user to select constrain this analysis to only the PP data associated with a specific project.)

You will want to attack this exercise in pieces. I suggest you begin by studying the implementation of related analyses, such as Project Active Time located in the package org.hackystat.stdext.activity.analysis.projectactivetime, which requires the exact same set of selectors and computes a similar kind of value. Next, I would start by implementing a simplified version: one that implements only the "Day" grain size for Interval, the "Table" report type, and that ignores the "Project"

selector altogether and simply returns the total amount of PP time for the given interval. Once you have got that version working, move on to support other grain sizes (which is quite easy), other report types (also quite easy), and filters out all PP data not related to the specified Project (a little harder).

15. Client-side stand-alone sensor design and implementation. [6+ hours, depends on 13] So far, the only way for a user to send PP data to the server is by bringing up an interactive SensorShell session and manually typing in "add" commands for PP data. In this exercise, you will create a more usable client-side interface for your PP data, and also learn about programmatic manipulation of the SensorShell. Create a simple Java application that, when run, pops up a window allowing the user to enter a Start Time, an End Time, and a list of (fully qualified) files. It include a button called "Send" that, when pushed, does simple validity checks on the data (i.e. that Start Time < End Time), then creates an instance of SensorShell and sends the data to that instance. Place this code in your hackyPP component in the directory org.hackystat.app.pairprogramming.clienttool. Extend your local.build.xml file with a new Ant target that creates a jar file called pptool.jar containing all of the classes needed to run this application. Include a manifest mainclass entry that enables you to invoke this tool and bring up its window interface with 'java –jar pptool.jar'. See the hackyBuild/build.xml targets dealing with the construction of the sensorshell.jar file to see how to create manifest files and the mainclass entry. Test your PPTool manually to see that it successfully sends data to the local server.

16. Sensor integration into development environment tool. [6+ hours, depends on 15] The PPTool developed in exercise 15 is a stand-alone tool, which has advantages and disadvantages. It would also be nice to integrate PP recording directly into an editor. For this task, choose your favorite editor (such as Eclipse) and integrate the functionality of PPTool into it directly.

17. Software telemetry stream design and implementation. [6+ hours, depends upon 14] The stand-alone PP analysis has its uses, but the power of Hackystat is in its ability to combine together measurement data of multiple types from multiple sources. A particularly powerful approach to this is what we call "Software Telemetry". Begin by reading the HackyTelemetry Project Proposal at the Developer Services web site, then design one or more telemetry streams from your PP SDT. Finally, implement telemetry-style analysis for Pair Programming data. (Note: Hackystat support for software telemetry is advancing rapidly. It will be useful to you to contact Philip Johnson or Cedric Zhang to find out the latest status of telemetry infrastructure support before starting on this project.)

18. Research study design using Hackystat. [10+ hours, depends upon 14]. While the PairProgramming SDT has been a useful pedagogic mechanism, it is also the case that Pair Programming is a very active area of research in software engineering. There are many unanswered questions regarding the nature of Pair Programming that could be usefully investigated with appropriate Hackystat sensors, sensor data types, and analyses. For this exercise, review the literature on Pair Programming, generate one or more research questions concerning Pair Programming, and then design a study to gather data regarding that question. Does your study require you to extend or modify your current PairProgramming SDT? If so, how? If the SDT changes, how does that impact on your sensor? What kinds of analyses will you want to implement on the data you collect in order to answer your research question?

Introduction to Hackystat

Philip Johnson
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Overview of the Talk

Introduction: The Big Problem

Overview of Hackystat

Hackystat in Use:

- A basic scenario
- Managing cost of quality
- The Hackystat-UH Configuration
- The Hackystat-JPL Configuration

Getting involved: The HackyDev Environment

(2)

The Big Problem

Collection and analysis of software product and process data has been shown to be very useful for improving software quality and productivity.

BUT

Collecting and analyzing software product and process data is very hard to do in practice!

(3)

Typical approaches

Typical approaches to collecting and analyzing software product and process data include:

- Make the developer do it.
  - Difficult to get consistency
  - Difficult to maintain over time.
- Create a Software Process Group
  - Expensive to implement
  - Usually dissolved when budget cut-backs.
- Enforce use of large commercial environment with built-in metrics support
  - Expensive for organization (per seat fees)
  - Developers may be more efficient with other tools.
  - Hard to extend and customize

(4)

Our approach to the Big Problem: The Hackystat Project

Create a "smart" software development environment that "understands" collection and analysis of software product and process data.

Hackystat should "understand"

- **...the tools used by developers and managers.**
- **...who is working together, how, and why.**
- **...how to collect product and process measures.**
- **...how to be helpful to developers/managers:**
  - **–Don't make them collect data themselves.**
  - **–Tell them when something interesting occurs.**

(5)

Applications: Research, Practice

For empirical software engineering researchers:

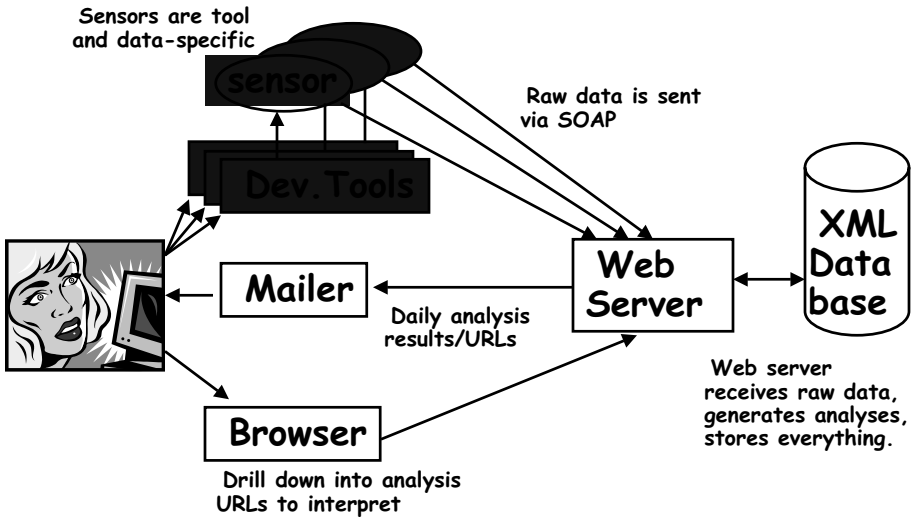
- **Provides a test bed infrastructure**
- **Low cost process/product data collection**
- **Tool/Environment agnostic**
- **Configurable for different environments.**

For practicing software developers/managers:

- **Open source, freely available.**
- **Low-overhead metrics collection**
- **Integrated with popular development tools**

(6)

Architecture in a nutshell



(7)

Key Innovations I

Low overhead for developers and managers
- Sensors attach to tools and unobtrusively collect information.

Tool agnostic.
- Eclipse, Emacs, Jbuilder, Ant, Jblanket, Vim, Visual Studio, JUnit, CCCC, Office, CVS, CLI

In-process feedback.
- Turn-around is hours/days, not weeks/months.

(8)

Key Innovations II

Accumulates in-process data.
- Supports "Control Chart" type analyses.
- Enables management using "Software Telemetry"

Configurable for different development contexts
- Current configurations for UH and Jet Propulsion Lab.

Java-based, open source tools/techniques:
- CVS, Ant, JUnit, HttpUnit, JSP, Tomcat, JDOM, Cruise Control, JFreeChart.

(9)

Key Innovations III

Exploration of structured Sensor Data Types:
- Activity: represents editor actions (file open, statechange, compile, etc.)
- FileMetric: represents the size and complexity of arbitrary file objects.
- Coverage: represents test case coverage.
- UnitTest: represents a test case invocation and its results (pass, fail, error)
- Commit: represents a file commit to a repository and number lines add/deleted.
- Build: represents an attempt to build a system and the results.
- BufferTransition: represents user change of focus of attention in an editor.

(10)

Current Status

Publically available since 2001.

In 5th architectural revision.

Approximately 40,000 LOC.

Funded by: Sun, IBM, NSF, NASA

Reaching "critical mass" in terms of architecture, sensors, analyses, understanding.

Hackystat deployments into multiple domains (HPC, Agile, Classroom, JPL).

(11)

Next steps

Evaluation of software telemetry as organizing principle for hackystat-guided project management.

Commercial spin-offs based upon open source code base. (Service-based, outsourcing, etc.)

Development of standards, best practices, and experience repositories for Hackystat-guided projects.

(12)

Hackystat in Use:

A basic scenario

(13)

Simple use

The following screen shots illustrate a typical sequence of events when using Hackystat:

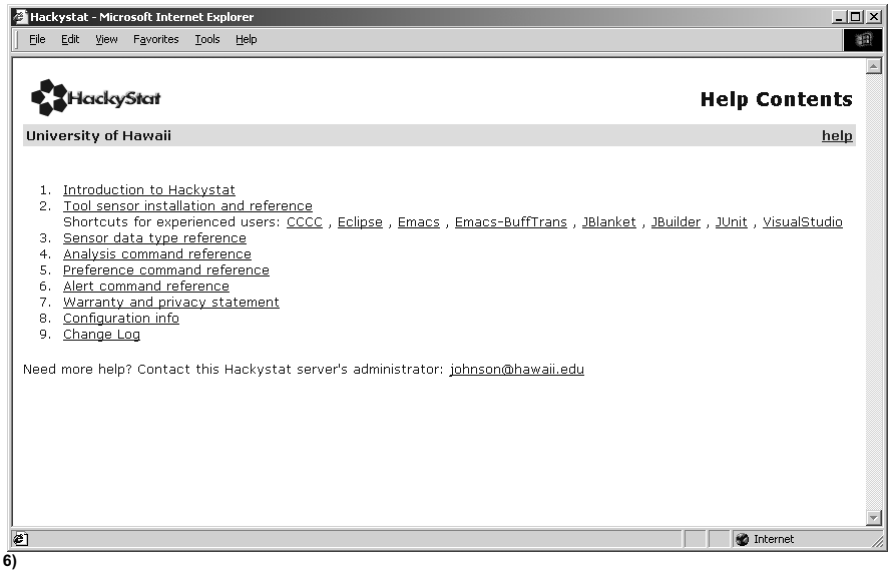
- Registration with server
- Installation of sensors
- Configuration of server (alerts, projects)
- Automatic process and product collection
- Automatic process and product analysis
- User interpretation of metric data to improve development

(14)

User registers with server

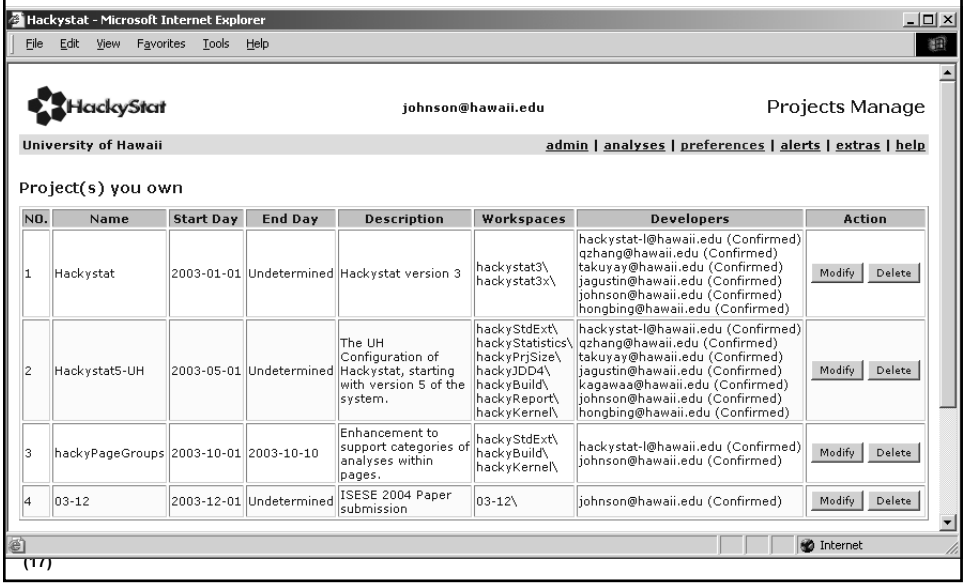
(15)

User installs client-side sensors



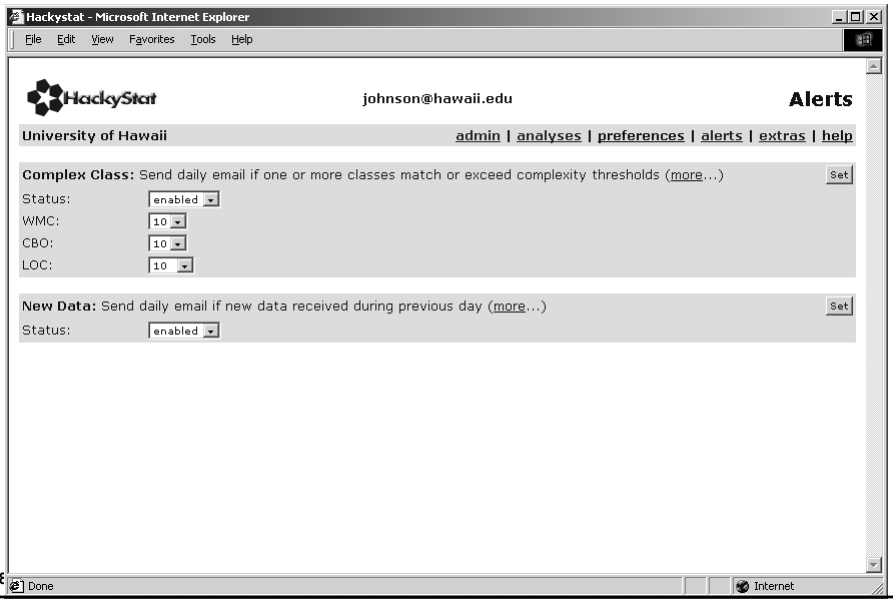
(16)

User configures projects



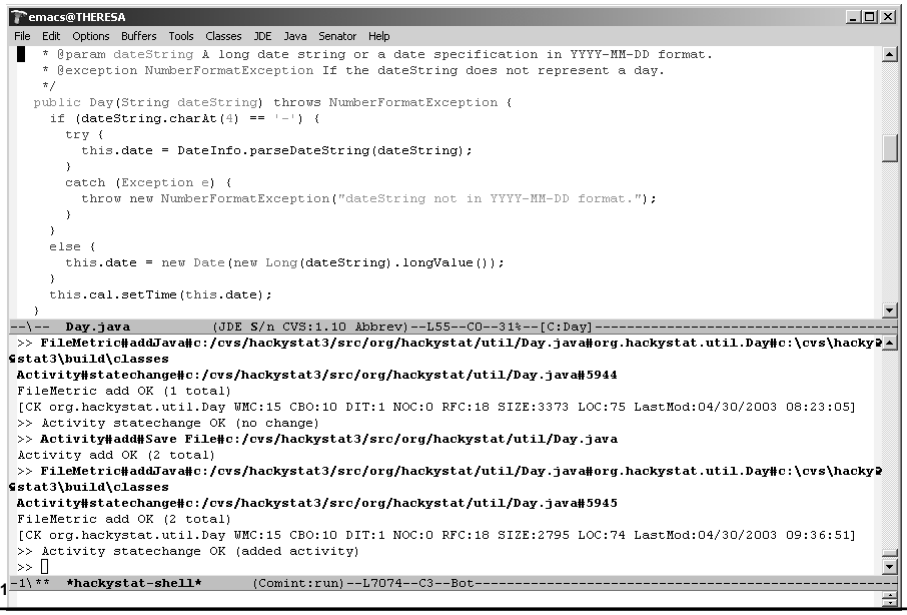
(17)

User configures alerts

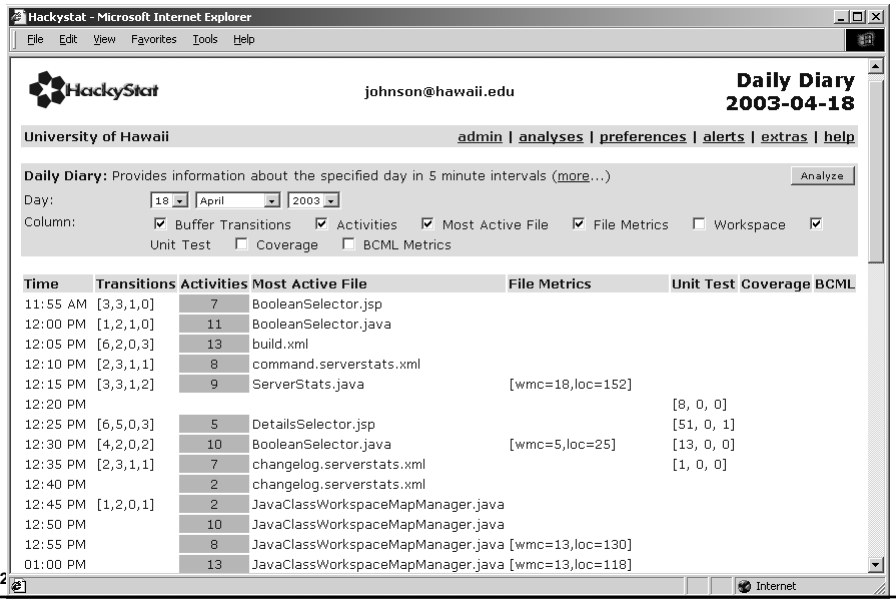


(18

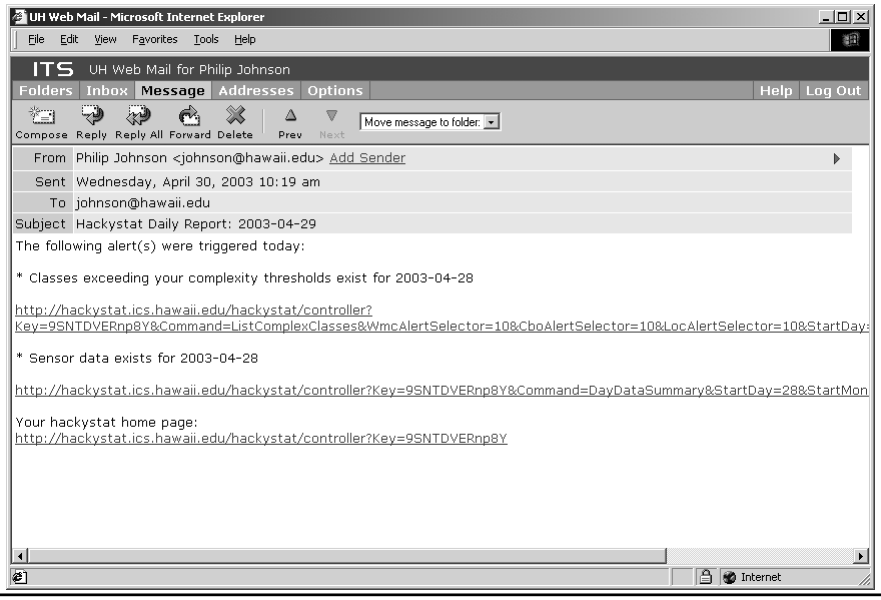
System automatically collects data



System models the work day

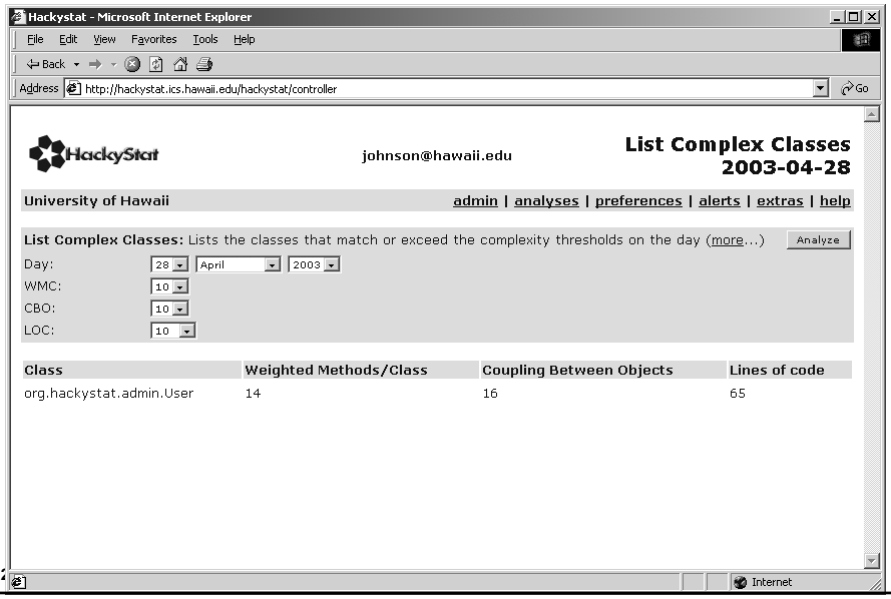


System notifies user of "interesting" data



(21)

User investigates and interprets



(22

Hackystat in Use:

Managing cost of quality

(23)

How to keep quality cheap?

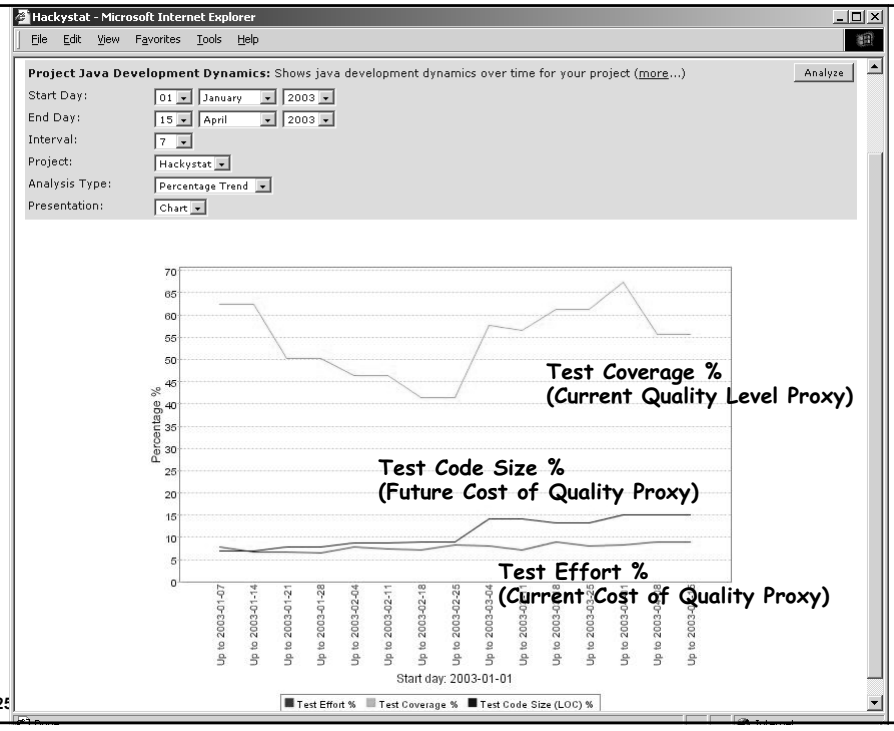
Assume:

- Current cost of quality
  - Proxy: % development effort on test code
- Future cost of quality
  - Proxy: % system code dedicated to testing
- Current level of quality
  - Proxy: % coverage of system by test code

Track values to implement a CoQ Policy, such as:

- Maximize current quality level, minimize future cost.
- Keep current quality level above threshold value while minimizing current and future cost.

(24)



(25

Current status and results: CoQ

Data available from Hackystat project and student projects.

Results:

- Hackystat has low cost of quality (>20%), but low coverage quality (~80%).
- Student projects incur significantly higher cost of quality (~50%) in order to get high coverage quality (~95%).

Future directions:

- Does coverage quality accurately predict in-field defects?

(26)

Hackystat in Use:

The Hackystat-UH Configuration

(27)

Hackystat-UH

A configuration for classroom use.

Features:

- Collects student active time, code size, and unit testing data.
- Provides students with information about their group members process.
- Provides groups with information about other groups process.

(28)

Project Member Active Time

Group process question:

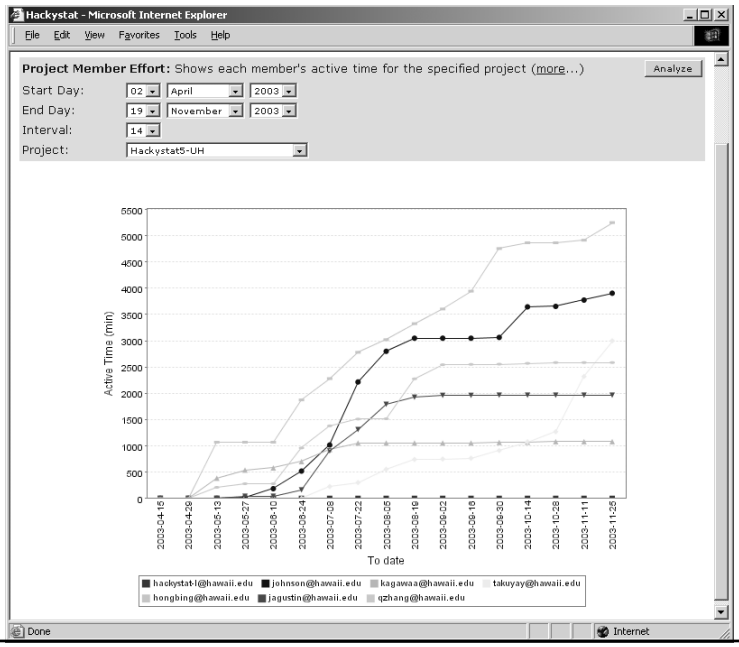
- Are all members of the group putting in equal effort over time?
- Are all members working consistently on the project over time?

In an "ideal" group process, all members would work consistently and equally on the project.

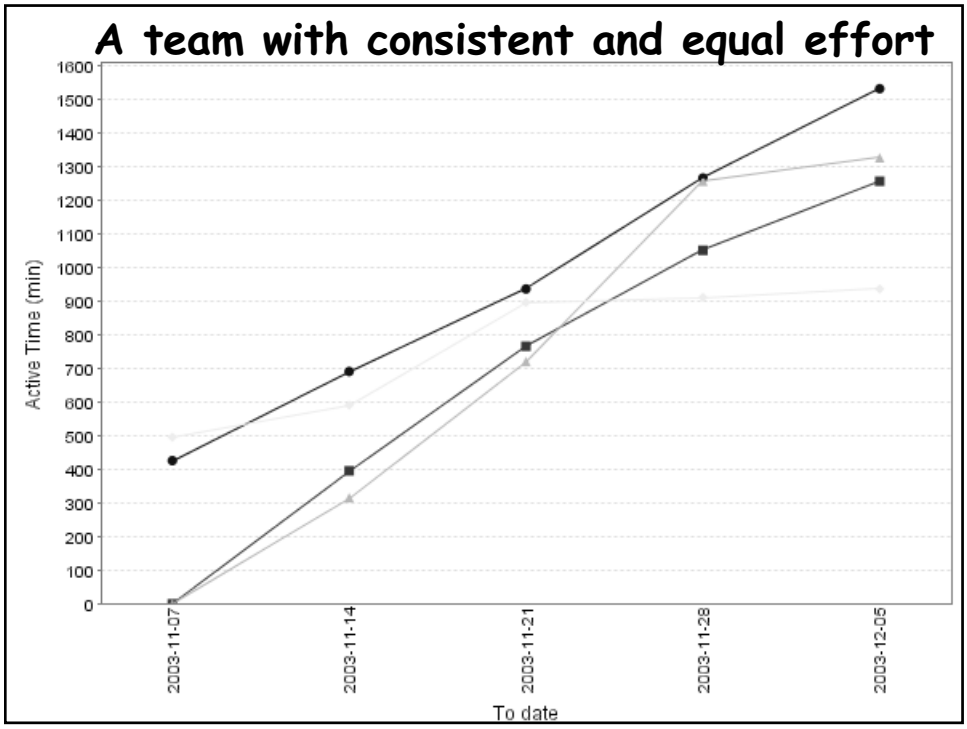
This analysis allows each member of the group to see how they are doing relative to others, and whether effort is consistent.

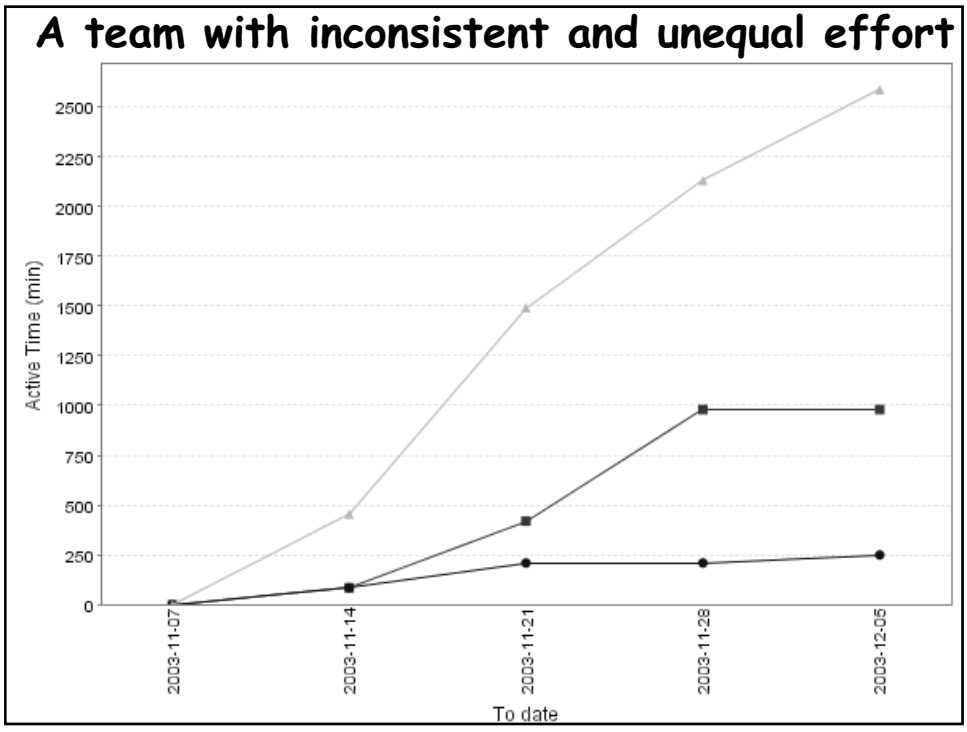
(29)

Example: Hackystat Project



(30)


A team with consistent and equal effort
Active Time (min)
1600
1500
1400
1300
1200
1100
1000
900
800
700
600
500
400
300
200
100
0
2003-11-07
2003-11-14
2003-11-21
2003-11-28
2003-12-05
To date


A team with inconsistent and unequal effort
Active Time (min)
2500
2250
2000
1750
1500
1250
1000
750
500
250
0
2003-11-07
2003-11-14
2003-11-21
2003-11-28
2003-12-05
To date

Comparative Project Analyses

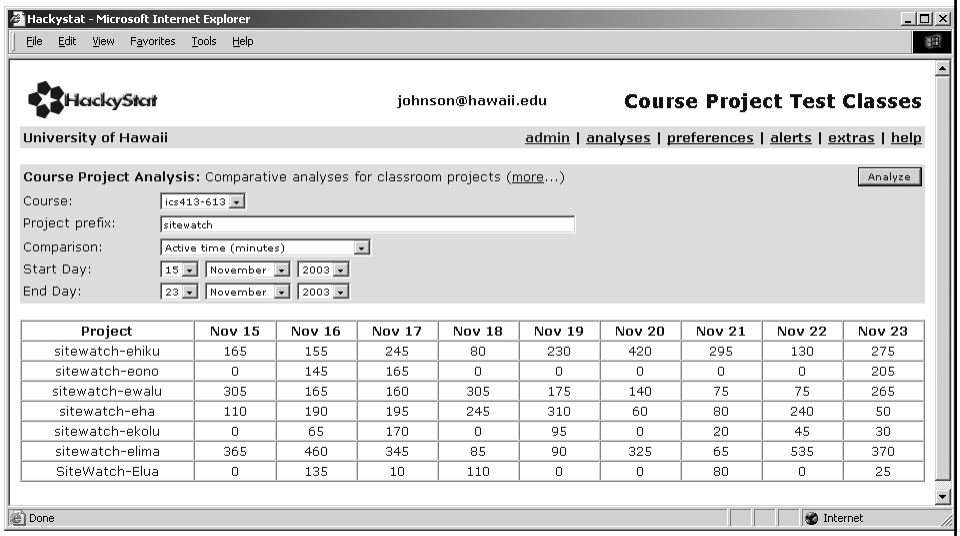
In a classroom setting, students may be split up into multiple groups, each of which work on the same (or similar) projects:

Question:

- Can we help students by enabling them to compare their group's progress against other groups?

(33)

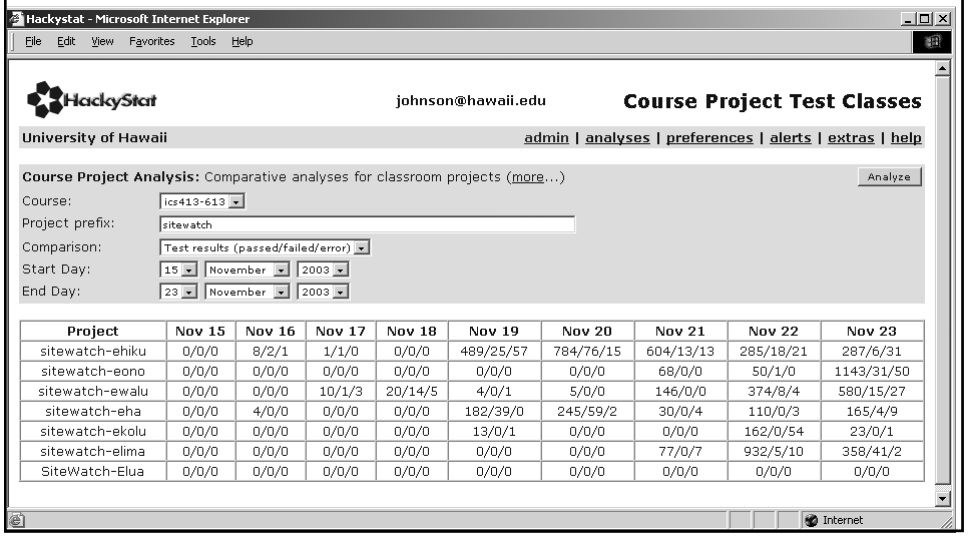
Project Active Time Trends



Project	Nov 15	Nov 16	Nov 17	Nov 18	Nov 19	Nov 20	Nov 21	Nov 22	Nov 23
sitewatch-ehiku	165	155	245	80	230	420	295	130	275
sitewatch-eono	0	145	165	0	0	0	0	0	205
sitewatch-ewalu	305	165	160	305	175	140	75	75	265
sitewatch-eha	110	190	195	245	310	60	80	240	50
sitewatch-ekolu	0	65	170	0	95	0	20	45	30
sitewatch-elima	365	460	345	85	90	325	65	535	370
SiteWatch-Elua	0	135	10	110	0	0	80	0	25

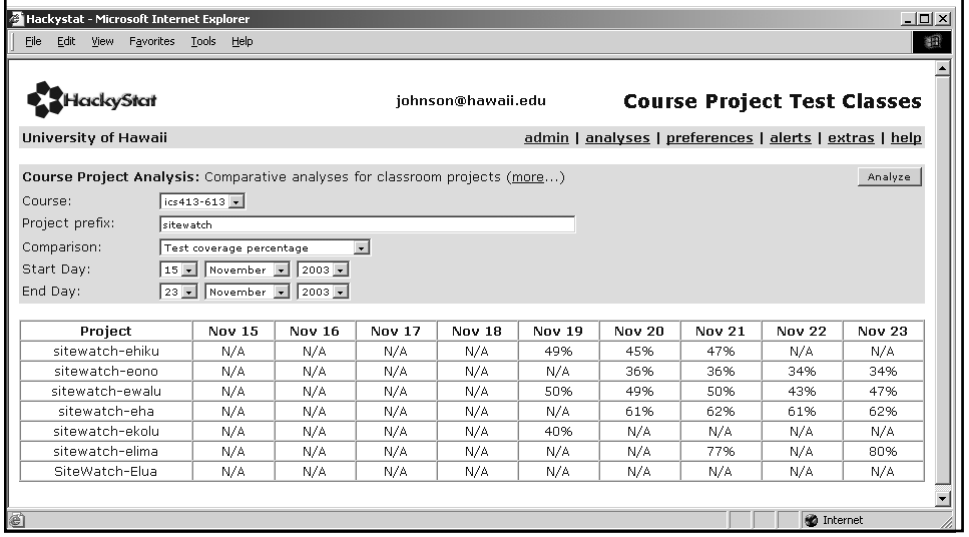
(34)

Test Results Trend (passed/failed/error)



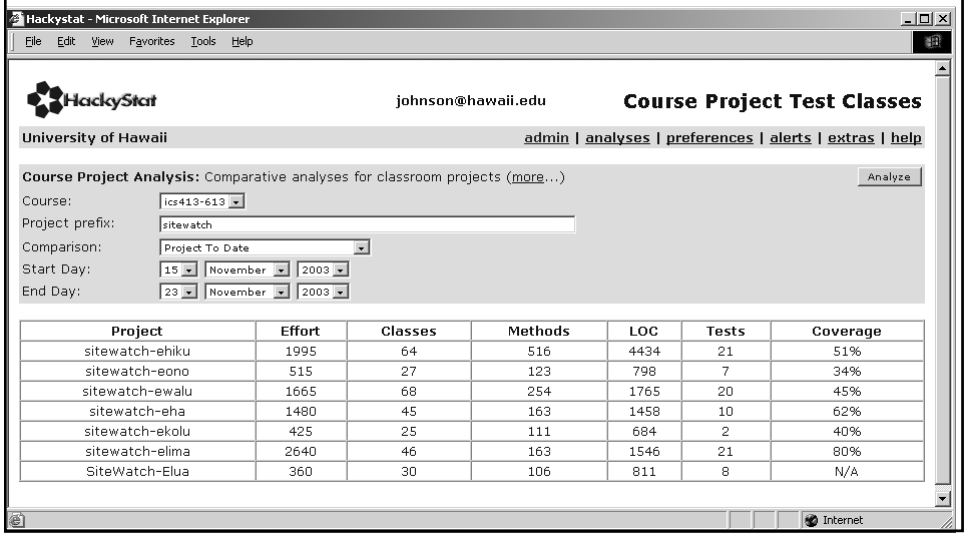
Project	Nov 15	Nov 16	Nov 17	Nov 18	Nov 19	Nov 20	Nov 21	Nov 22	Nov 23
sitewatch-ehiku	0/0/0	8/2/1	1/1/0	0/0/0	489/25/57	784/76/15	604/13/13	285/18/21	287/6/31
sitewatch-eono	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	68/0/0	50/1/0	1143/31/50
sitewatch-ewalu	0/0/0	0/0/0	10/1/3	20/14/5	4/0/1	5/0/0	146/0/0	374/8/4	580/15/27
sitewatch-eha	0/0/0	4/0/0	0/0/0	0/0/0	182/39/0	245/59/2	30/0/4	110/0/3	165/4/9
sitewatch-ekolu	0/0/0	0/0/0	0/0/0	0/0/0	13/0/1	0/0/0	0/0/0	162/0/54	23/0/1
sitewatch-elima	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	77/0/7	932/5/10	358/41/2
SiteWatch-Elua	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0	0/0/0

Test Coverage Percentage



Project	Nov 15	Nov 16	Nov 17	Nov 18	Nov 19	Nov 20	Nov 21	Nov 22	Nov 23
sitewatch-ehiku	N/A	N/A	N/A	N/A	49%	45%	47%	N/A	N/A
sitewatch-eono	N/A	N/A	N/A	N/A	N/A	36%	36%	34%	34%
sitewatch-ewalu	N/A	N/A	N/A	N/A	50%	49%	50%	43%	47%
sitewatch-eha	N/A	N/A	N/A	N/A	N/A	61%	62%	61%	62%
sitewatch-ekolu	N/A	N/A	N/A	N/A	40%	N/A	N/A	N/A	N/A
sitewatch-elima	N/A	N/A	N/A	N/A	N/A	N/A	77%	N/A	80%
SiteWatch-Elua	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A

Project To-Date Summary



Project	Effort	Classes	Methods	LOC	Tests	Coverage
sitewatch-ehiku	1995	64	516	4434	21	51%
sitewatch-eono	515	27	123	798	7	34%
sitewatch-ewalu	1665	68	254	1765	20	45%
sitewatch-eha	1480	45	163	1458	10	62%
sitewatch-ekolu	425	25	111	684	2	40%
sitewatch-elima	2640	46	163	1546	21	80%
SiteWatch-Elua	360	30	106	811	8	N/A

Current Status and Results: Hackystat-UH

Current Status:

- Used in Fall 2003 with 27 students.
- Used in Spring 2004 with 6 students.
- Interest from other universities
  - U. North Carolina; U. Torino, Italy

Results:

- Students found analyses useful.
- Installation problems occurred.
- Very low overhead in daily use.
- Privacy concerns.

(38)

Hackystat in Use:

The Hackystat-JPL Configuration

(39)

Mission Data System

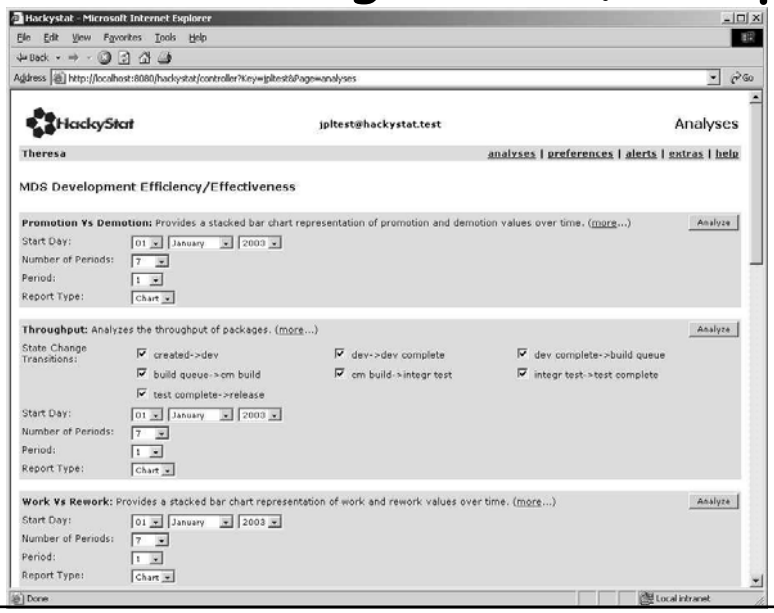
The Mission Data System is a NASA-sponsored project to develop next-generation flight control system for use in the Mars 2008 Science Lab mission.

Features:
- Provides hardware-independency
- About 40 developers
- About 5M lines of C++ code

Question:
- Can Hackystat monitor build process and perform analyses useful for improving build process efficiency?

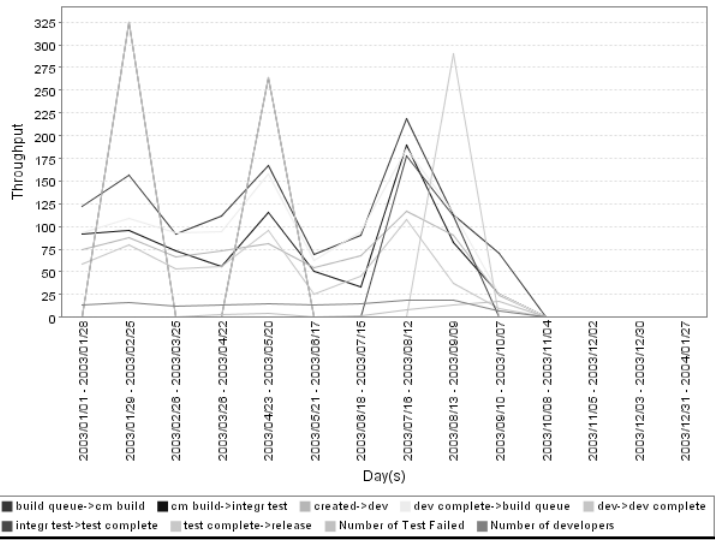
(40)

The JPL configuration (excerpt)



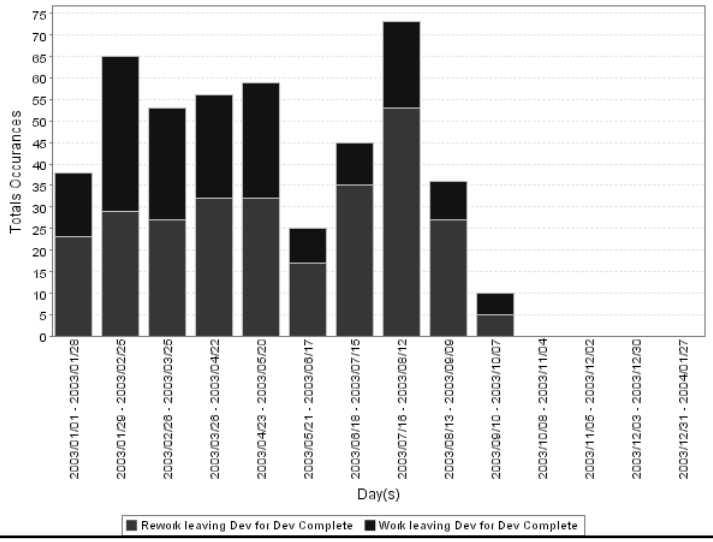
(41)

Build state changes show regularities



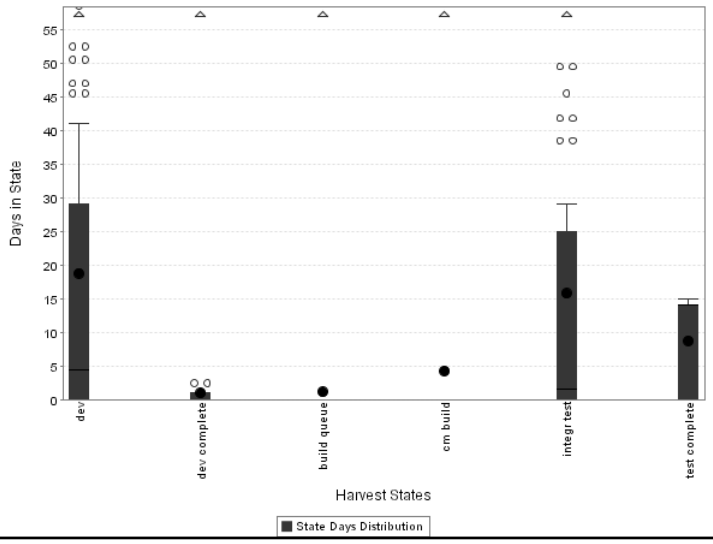
(42)

Work vs. Rework shows substantial rework



(43)

Box-and-Whiskers identifies workflow states with high variability



(44)

Current status and results: Hackystat-JPL

Current status:

- Deployed on test data in Fall, 2003.
- Scheduled for live deployment in MDS in Summer, 2004.
- Interest from other NASA groups.

Results:

- Discovered previously unknown process issues in MDS workflow.
- Now developing process changes based upon data.

(45)

Getting Involved:

The HackyDev Environment

(46)

Supporting distributed development of Hackystat

We want a way to enable other researchers to become involved in Hackystat development.

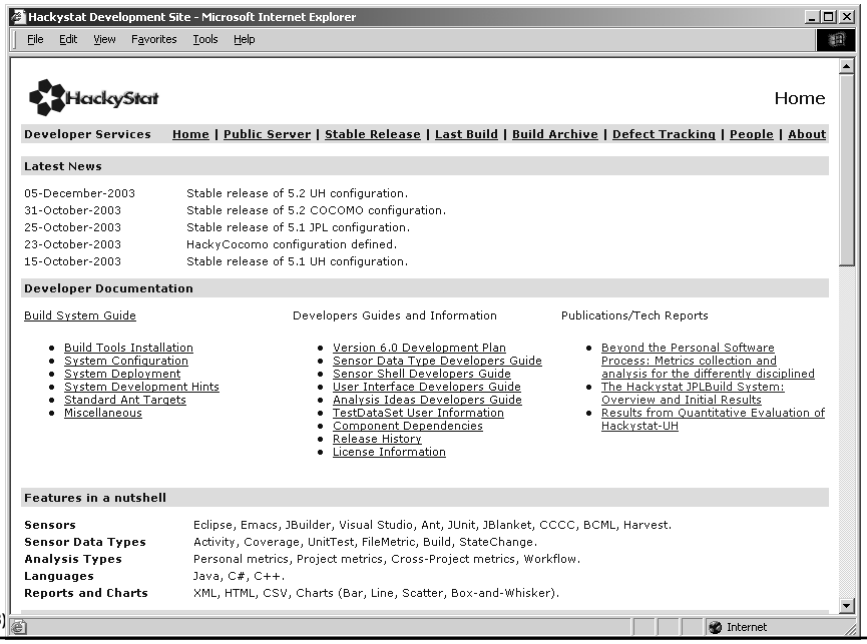
The HackyDevSite web service facilitates distributed development.

Features:

- Documentation server
- Automated nightly builds and tests
- Automated daily report

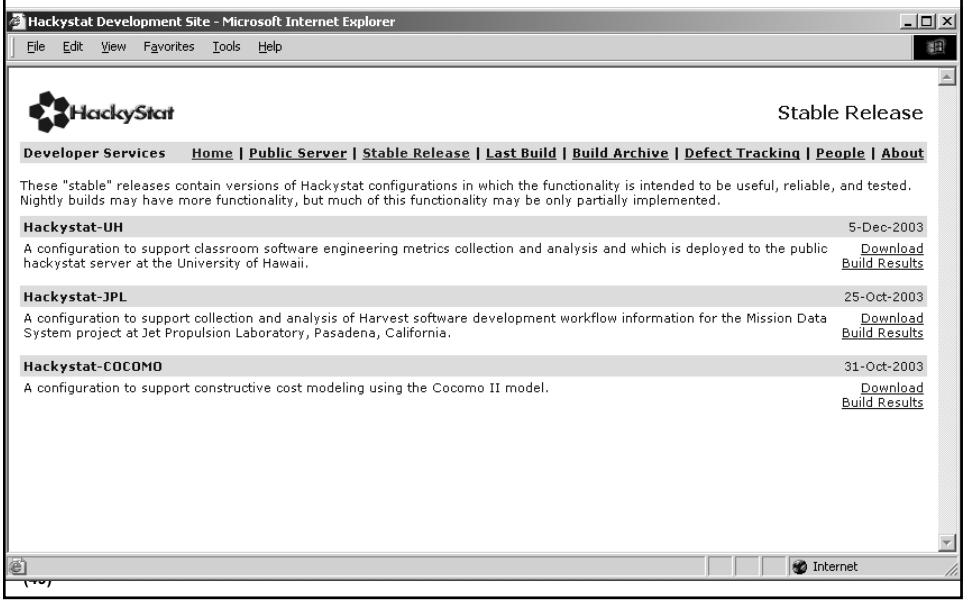
(47)

http://hackydev.ics.hawaii.edu/

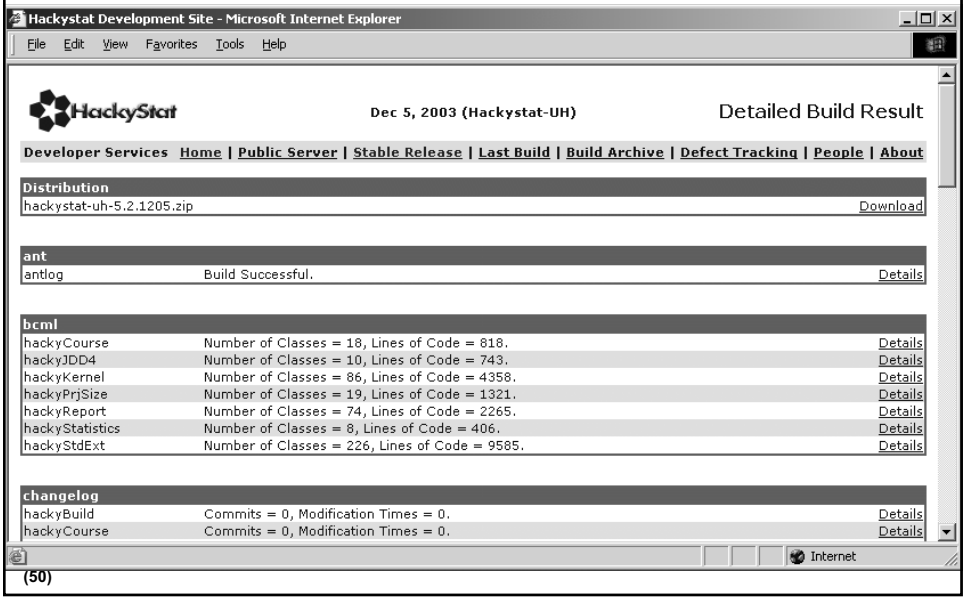


(48)

Online Stable Release Page

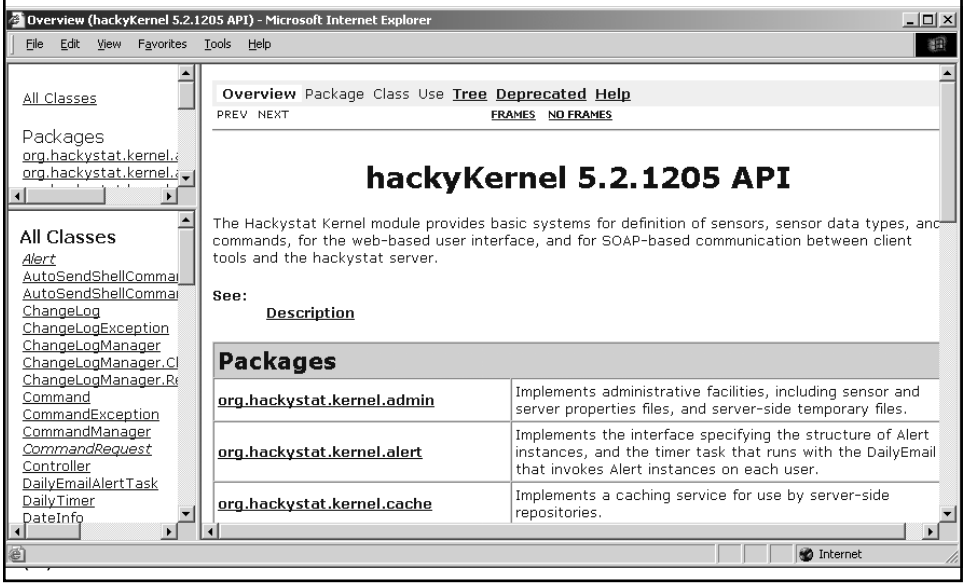


Online Build Reports

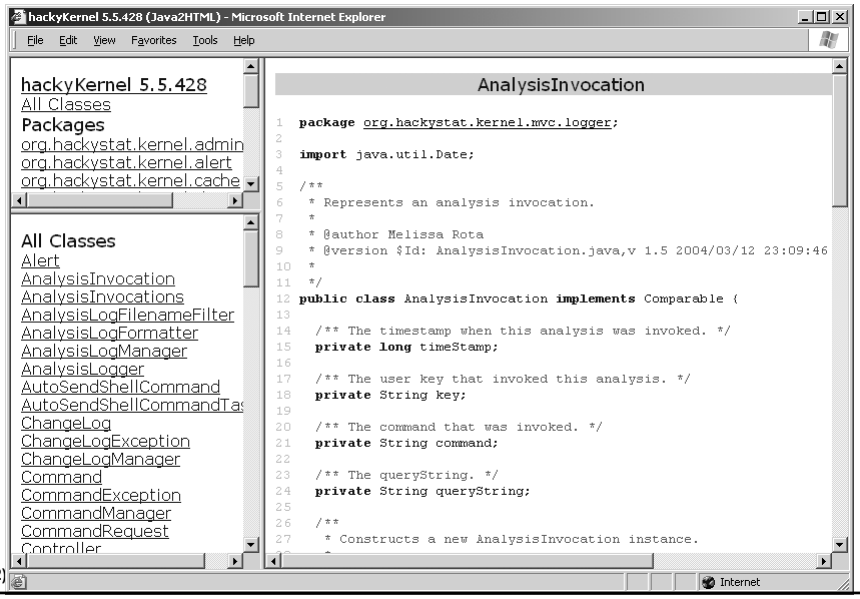


(50)

Online Design (Javadocs)

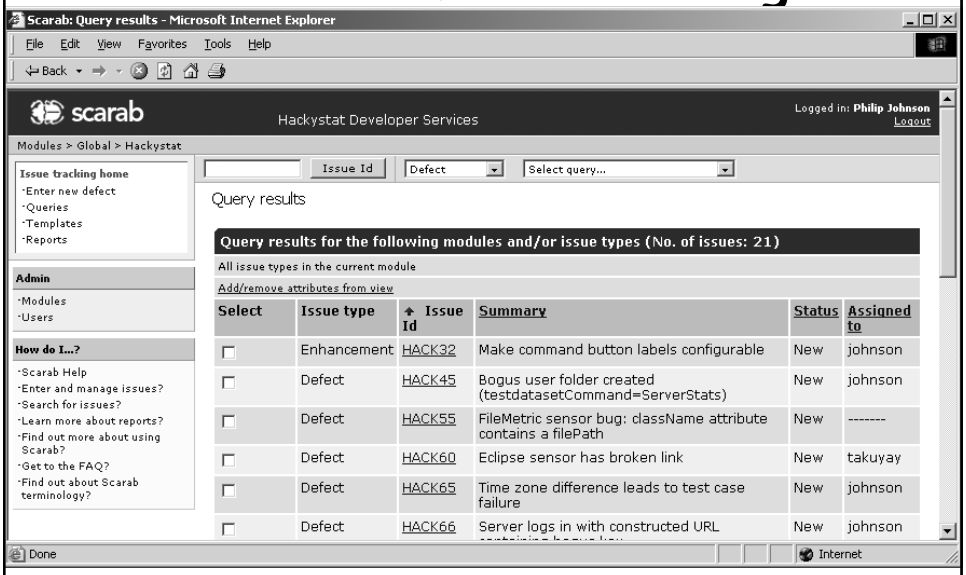


Online Source Code



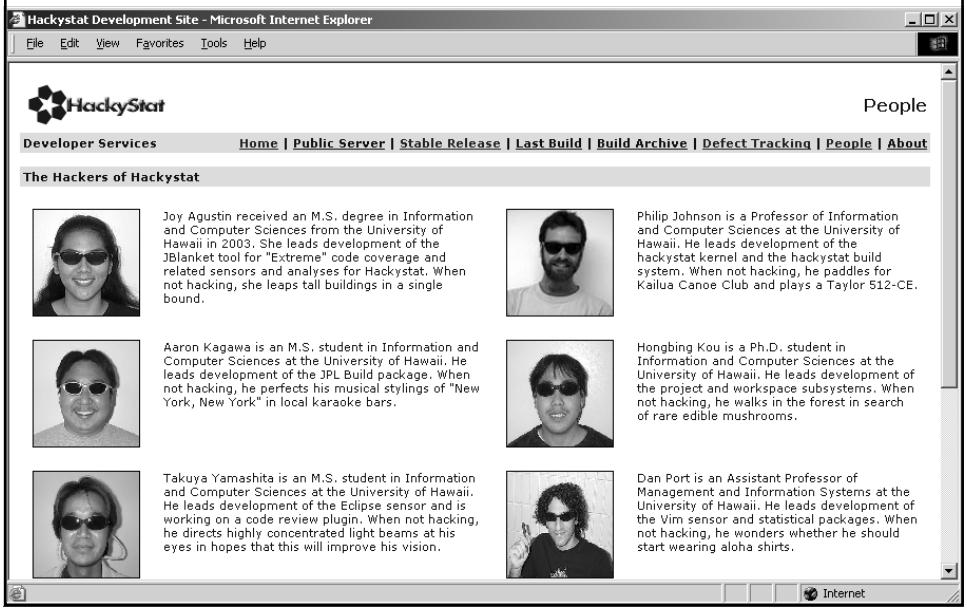
(52)

Online Defect Tracking



(53)

Your name here!



Links to more information

Hackystat public server:
- http://hackystat.ics.hawaii.edu/

Hackystat development site
- http://hackydev.ics.hawaii.edu/

Hackystat research page:
- http://csdl.ics.hawaii.edu/Research/Hackystat

CSDL Home page:
- http://csdl.ics.hawaii.edu/

(55)

Thank you!

Any Questions?

(56)

Anatomy of the Hackystat Component Architecture

Philip Johnson
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Overview of the Talk

Motivation for the component architecture.

Current components

How to integrate new components.

(2)

Motivation

Software measurement and analysis is not a "one size fits all" situation.

Different organizations require:

- Sensors specific for the tools they use.
- Sensor data types specific to the kinds of data they want to collect.
- Analyses specific to the kinds of questions they want to answer.

We went through several different architectures trying to address this problem

(3)

Architectural History

A-1. Spike Solution Architecture (2001)
- We ignored component issues entirely

A-2. Framework Architecture (2002)
- Extension via inheritance and composition.
- Monolithic source code.

A-3. SDK Architecture (early 2003)
- Two layers: Kernel and Standard Extensions.

A-4. Three Layer Architecture (June, 2003)
- Three layers: Kernel, StdExt, App layers.

A-5. Component Architecture (August, 2003)
- Arbitrary numbers of layers
- Named "configurations" for subsets
- Automated build support for each configuration.
  - Not practical without tool support!

(4)

Components and Dependencies

Each hackystat component is also a CVS module.
- hackyKernel, hackyReport, hackyVCS, etc.

Each component has dependencies:
- hackyStdExt depends upon hackyReport, hackyStatistics, hackyKernel.
- hackyEclipse depends upon hackyStdExt.

Dependencies declared in hackyBuild/build.xml.

Ant compile/build targets verify dependencies.

(5)

Configurations

A set of components can be grouped together into a configuration.

Hackystat-*COCOMO* configuration:
- hackyKernel, hackyCocomo

Hackystat-JPL configuration:
- hackyKernel, hackyReport, hackyStatistics, hackyStdExt, hackyJPLBuild

Each component can participate in many configurations.

There is no limit to the number of configurations that can be defined.

(6)

Configurations and HackyDevSite

Configurations are defined as <project> instances in cruise control.

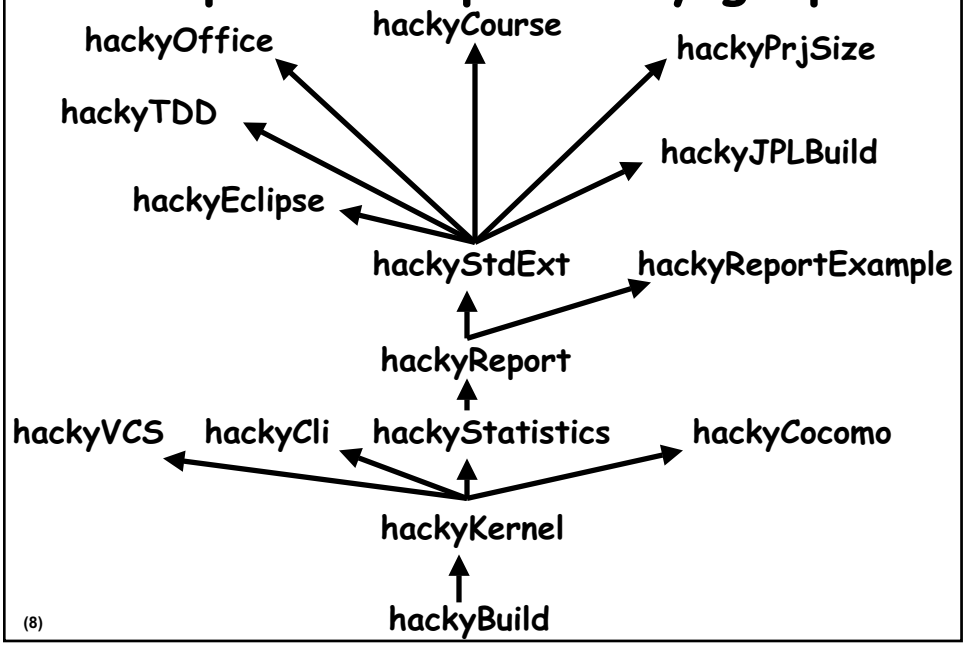
Daily build process builds and tests each configuration.

One configuration, hackystat-ALL, builds and tests all component modules at once to make sure all components are build-level compatible.

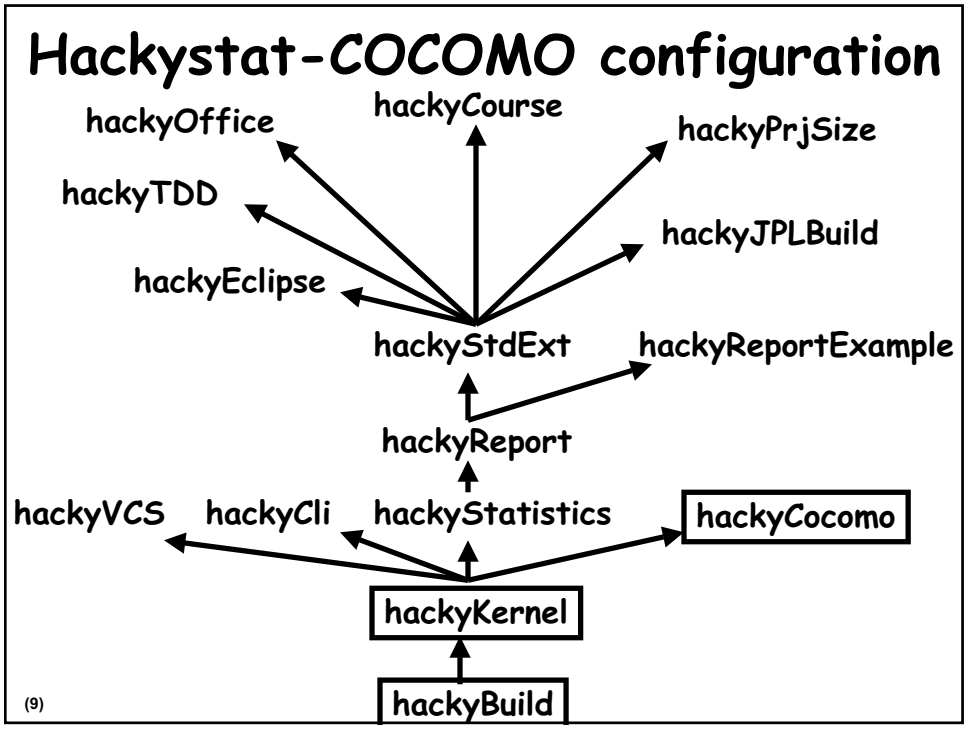
Metrics are collected on hackystat-ALL configuration.

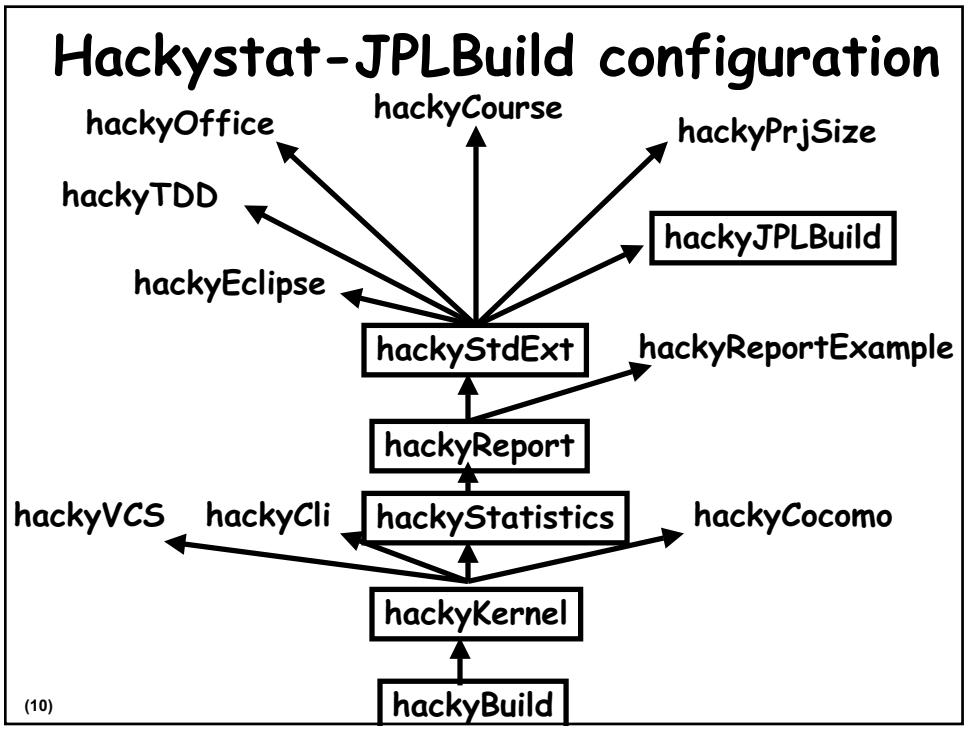
(7)

Component dependency graph

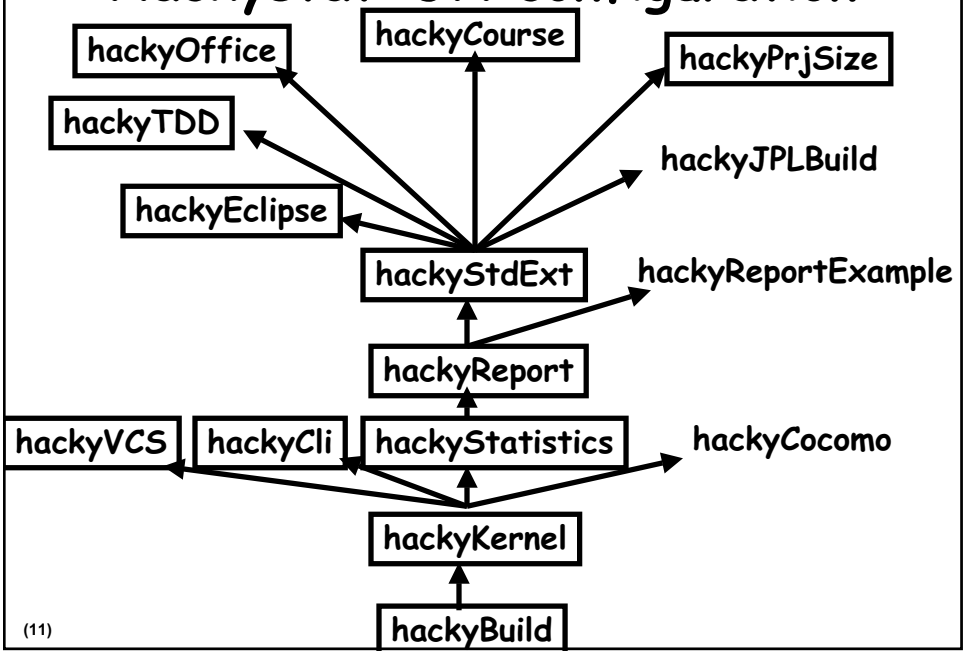


(8)


Hackystat-COCOMO configuration
hackyOffice
hackyCourse
hackyPrjSize
hackyTDD
hackyJPLBuild
hackyEclipse
hackyStdExt
hackyReportExample
hackyReport
hackyVCS
hackyCli
hackyStatistics
hackyCocomo
hackyKernel
hackyBuild
(9)


Hackystat-JPLBuild configuration
hackyOffice
hackyCourse
hackyPrjSize
hackyTDD
hackyJPLBuild
hackyEclipse
hackyStdExt
hackyReportExample
hackyReport
hackyVCS
hackyCli
hackyStatistics
hackyCocomo
hackyKernel
hackyBuild
(10)

Hackystat-UH configuration



(11)

Dependency Graph Issues

hackyKernel and hackyStdExt are the two primary "platforms" from which other components are built.

hackyKernel implements basic infrastructure: UI, Soap/XML data transmission, persistency, caching.

hackyStdExt implements basic SDTs, Sensors, and Commands.

(12)

Components and Sensors

Currently hackyStdExt contains some sensors:
- Jbuilder, Emacs, Ant, VisualStudio

Other sensors are in their own module:
- hackyEclipse, hackyOffice, hackyVCS, hackyCLI

The latter is better!

Eventually all sensors will migrate out of hackyStdExt and into their own component.

(13)

Custom Configurations

Each developer can define their own configurations while doing development to include only those components necessary for the development they are doing.

Procedure:
- Check out components of interest from CVS.
- Edit hackystat.properties file to specify your configuration.
- Build, test, run your local configuration.

(14)

hackystat.properties

```
# 3. Ant Module Settings
hackyKernel.available=true
hackyStatistics.available=true
hackyReport.available=true
hackyStdExt.available=true
#hackyEclipse.available=true
#hackyReportExample.available=true
#hackyJPLBuild.available=true
#hackyJDD4.available=true
#hackyPrjSize.available=true
#hackyEstimate.available=true
hackyCourse.available=true
#hackyCocomo.available=true
#hackyTDD.available=true
#hackyCli.available=true
#hackyVCS.available=true
```

This custom configuration consists of hackyKernel, hackyStatistics, hackyReport, hackyStdExt, and hackyCourse.

To add a module to your local configuration, uncomment the corresponding line.

Note that the value (true or false) has no effect!

(15)

Defining new components

To extend Hackystat with a brand new component involves the following:

1. Create a new CVS module containing your component. Place your component in the same directory as other hackystat components. Define a local.build.xml for your component.

2. Edit hackyBuild/build.xml to define what component(s) yours depends upon.

3. Edit hackystat.properties to add a line referring to your component:
   - `hacky<Component>.available = true`

(16)

hackyBuild/build.xml

Target checkModuleAvailability:

```
• <antcall target="doCheckModuleAvailability">
    <param name="moduleName" value="hackyKernel"/>
   <param name="moduleAvailable" value="${hackyKernel.available}"/>
   </antcall>
```

Targets doAll and doInternal_hacky<Component>:

```
• <target name="doInternal_hackyStatistics"
     depends="doInternal_hackyKernel"
     if="hackyStatistics.available">

     <fail message="hackyStatistics requires module hackyKernel."
           unless="hackyKernel.available"/>

     <echo message="(${ant.project.name}) Invoking ${target} in
                    hackyStatistics."/>

     <ant dir="../hackyStatistics"
          antfile="local.build.xml"
          target="${target}"/>
   </target>
```

(17)

Incorporating a "custom" configuration into HackyDevSite

Steps:

- Edit cruisecontrol files with new <project>
- Create directory to hold configuration build.
- Edit webapp to specify stable release (if any).

Consult with the Grand HackyBuildMaster (Cedric) for details and help.

(18)

Future configurations

Hackystat-HPC:

- A configuration to support measurement and experimentation of high performance computing systems. Includes special analyses for parallel metrics (speedup, etc.)

Hackystat-Torino:

- A configuration to support studies of TDD at the University of Torino.

Configurations are lightweight; we can support many more.

(19)

Status

The overall Hackystat component architecture, combined with the hackydevsite and the daily build server, appears adequate for our current and future needs.

We do anticipate changes to build.xml and hackystat.properties over time to make the system even easier to use, modify, and extend.

(20)

Anatomy of HackyKernel

Philip Johnson
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Overview of the Talk

Overview and motivation

Problems the kernel tries to solve include:

- Definition and transmission of sensor data.
- Simplifying user interface coding for the hackystat command developer.
- Storing sensor data.
- Efficient and effective command testing

(2)

Motivation

HackyKernel emerged to provide "domain independent" core services to all other modules.

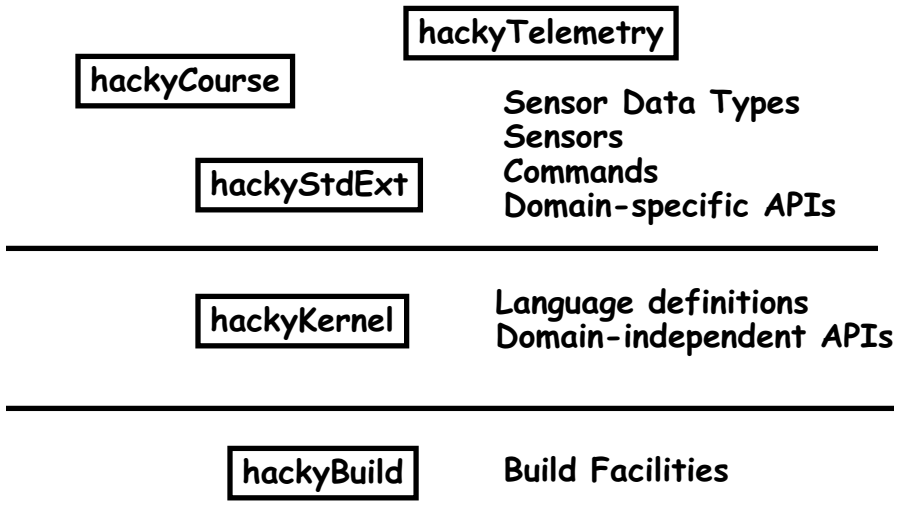
HackyKernel includes NO sensor data type definitions, sensor definitions, command definitions, etc.

HackyKernel provides the implementation of the definition mechanisms for sensor data types, sensors, commands, etc.

All other modules require hackyKernel.

(3)

The Pie in Three Slices



(4)

org.hackystat.kernel packages

admin	ServerProperties, SensorProperties
alert	Alert API
cache	thread-safe, soft ref, three key caching
changelog	Help page ChangeLog
command	Command API
mvc	model-view-controller UI
sdt	SensorDataType API
sensor	Sensor API
sensordata	sensor data storage API
shell	client-side SensorShell
soap	client->server data transmission
test	testing framework
timer	periodic analysis invocation
user	User ID and properties
util	Day, Mailer, StringListCodec, etc.

(5)

Problem 1: sensor data

New sensor data types always emerging
- Cannot hardwire, must support extension.
- Different configurations require different SDT sets.

Sensor data is manipulated by both client sensor code and server web application
- A single declaration must propogate to both.

The server (i.e. network) is not always available.
- Sensor data must be cached at client at times.

Sensor data always represents a time-stamped event and is manipulated by its timestamp.

(6)

Solution 1: SensorShell and SDTs

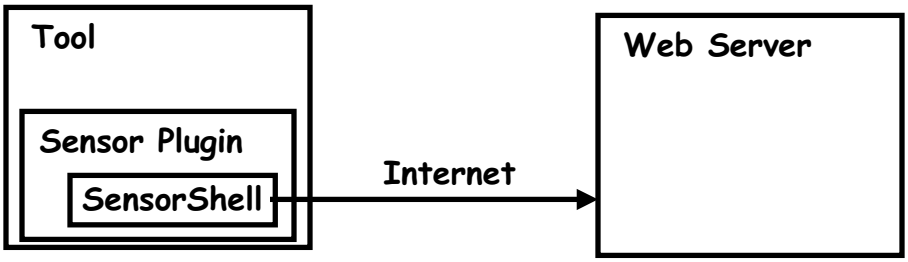
SensorShell.java:
- A client-side service for accepting SDT instance data and sending it to the server.

Sensor Data Type:
- A semi-declarative language construct for defining structured time series data that can be transmitted over unreliable networks.

(7)

The SensorShell

Provides client-side "middleware".
- Knows about all currently defined SDTs.
- Performs SOAP/XML transmission to server.
- Automatically stores data if client offline.
- Provides interactive CLI and programmatic API
- Simplifies sensor development.



(8)

SensorShell CLI

```
C:\cvs\hackyBuild\build\war\download>java -jar sensorshell.jar
Hackystat Version: 5.6.506 (May 6 2004 11:29:20)
SensorShell started at: 05/08/2004 15:40:31
Type 'help' for a list of commands.
Host: http://hackystat.ics.hawaii.edu/ is available and key is valid.
Defined shell command: Activity
Defined shell command: Coverage
Defined shell command: UnitTest
AutoSend enabled every 10 minutes.
Checking for offline data to recover.
No offline data found.
>> Activity#add#c:\cvs\hackyDevSite\webapp\doc\bootcamp\CoverPages.doc
Activity add OK (1 total)
>> send
Sending sensor data (05/08 15:41:26)
  Activity: Send OK (1 entries)
  Ping: Ping OK (contacted server http://hackystat.ics.hawaii.edu/...)
  Coverage: Send OK (No entries to send.)
  AutoSend: AutoSend OK ('send' command ignored)
  UnitTest: Send OK (No entries to send.)
>> >>
```

(9)

SensorShell features

sensorshell.jar is standalone Java app.

Reads sensor.properties, makes connection to server (if available) upon startup.

Knows about SensorDataTypes present when it was built.

Can both save and restore offline data.

Buffers data additions locally.

Does both "auto sending" (every X minutes) or explicit sending (via send command).

(10)

Extending SensorShell

When defining an SDT, you also define a "ShellCommand" class that implements the commands to process your SDT within the SensorShell.

At build time, your code is packaged into sensorshell.jar.

At sensorshell invocation time, your code is "discovered" as part of the startup process and becomes part of the available sensorshell command set.

(11)

Sensor Data Types

A sensor data type defines:

- a set of typed fields
- the SensorShell commands required for sensors to manipulate this new type.

Files required to define the Activity SDT:

- sdt.activity.xml
- Activity.java
- ActivityShellCommand.java
- doc.sdt.activity.html

(12)

sdt.activity.xml (modified)

```
<sensordatatype>
  name="Activity"
  enabled="true"
  wrapper="org.hackystat...Activity"
  shellcommand="org.hackystat...ActivityShellCommand"
  docstring="Represents events occuring during editing."
  docfile="doc.sdt.activity.html"
  version="1.0.0"
  contact="Philip Johnson (johnson@hawaii.edu)">

  <entryattribute name="type"
    type="org.hackystat...ActivityType"
    converter="org.hackystat...ActivityType.getInstance"/>

  <entryattribute name="data" />
</sensordatatype>
```

(13)

sdt.activity.xml features

The Activity SDT has two fields:
- "type", an instance of ActivityType.
- "data", an instance of String (default).

Each Activity SDT entry is an instance of the Activity.java class.

SDT instances are always converted to a set of strings for transmission via SOAP/XML, then reconstituted into Java class instances.

SDT implementations must help this process.

(14)

Problem 2: User Interface

Each configuration will require its own set of:
- Analyses: interpretations of data
- Preferences: persistent "settings"
- Alerts: anomaly detectors w/email notification
- Help: documentation

Parameters to commands often standardized:
- Day/Week/Month time interval
- Project selection
- Report type (chart, XML, CSV, html table)

How to minimize development effort on UI?

(15)

Solution 2: Commands and MVC

MVC:
- An implementation of the model-view-controller design pattern with customizations for Hackystat.

Commands:
- A language construct for semi-declarative specification of analyses, preferences, and alerts.

(16)

org.hackystat.kernel.mvc

Controller.java:
- A servlet to which all requests to the hackystat web application are routed.
- Checks validity of request parameters, correctness of user key.
- Dispatches to appropriate help page or CommandRequest instance.

CommandRequest.java:
- Interface containing a process() method. All Commands implement CommandRequest.

Page.java:
- Interface representing the JSP page returned by a successful Command invocation.

(17)

Commands

Commands define:
- The parameters to be provided by the user.
- The class to process the command.
- The page to be returned to the user.
- The documentation for the command.

Files required for List Sensor Data command:
- ListSensorData.java
- ListSensorData.jsp
- command.listsensordata.xml
- doc.listsensordata.html
- TestListSensorData.java

(18)

command.listsensordata.xml

```
<command
  type="analysis"
  page="extras"
  label="List Sensor Data"
  enabled="true"
  commandrequest="org.hackystat...ListSensorData"
  docstring="Lists your sensor data of the given type for the day"
  docfile="doc.listsensordata.html"
  group="Validation"
  contact="Philip Johnson (johnson@hawaii.edu)">

    <parameter name="Type"
      file="SensorDataTypeSelector.jsp"
      requesthook="org.hackystat.stdext...SensorDataTypeSelector"/

    <parameter name="Day"
      file="StartDaySelector.jsp"
      requesthook="org.hackystat.stdext...StartDaySelector"/>

   <resultpage file="ListSensorData.jsp"/>
</command>
```

(19)

command.*.xml features

Specifies:

- Name of the command
- Page where it appears
- Class to process the command
- Parameters to be displayed with command
  - Called "Selectors" in Hackystat
- Page to be returned
- Documentation (short and long) for command.

(20)

Goal of UI API

Minimize developer effort required to define new commands in Hackystat with standard "look and feel".

- Minimize distraction on low-impact UI frills.

Maximize developer effort available to work on the real problem--how to produce meaningful interpretations of sensor data.

- This is the real engineering issue for Hackystat.

(21)

Problem 3: Server-side data

How does server access/manipulate sensor data?

Solution:

- Store sensor data in daily XML logs by SDT.
  - –org.hackystat.kernel.sensordata
- Provide thread-safe, soft reference in-memory caches, indexed by user, sdt, day and timestamp.
  - –org.hackystat.kernel.cache

Hackystat does not have a back-end RDBMS.

- Probably the most controversial design decision.
- Not clear what problem we actually have that an RDBMS will solve.
- Architecture would support this move if needed.

(22)

Problem 4: Testing

Problem: testing installed commands by manipulating web interface via httpunit and junit takes many lines of code.

- **Must login as user.**
- **Find correct page.**
- **Set parameters (selectors) appropriately.**
- **Generate response page**
- **Parse response page for correct output.**

Solution: org.hackystat.kernel.test

- **Provides high level testing facility.**
- **Generally reduces test code by 25-50%.**
- **Resulting tests are more thorough.**
- **Test log facilitates debugging.**

(23)

Anatomy of HackyStatistics and HackyReport

Hongbing Kou
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Overview

CVS Modules and description

Jargon for Report

Statistics
- Design
- Regression

Report
- Design
- Chart support (Normal, extraordinary, more)

Report Example
- How to use chart
- Report Example Module

(2)

Statistics Module

Module name:
- hackyStatistics

Functionalities:
- Isolated from hackyReport
- Regression
  - Linear
  - Logarithm
  - Power
  - Exponential
- Predication

(3)

Report Module

Module Name:
- hackyReport

Functionalities:
- Provides three kinds of charts -- category, XY and pie charts
- Provides four report types
  - Chart with tool tips and drill down URLs
  - CSV file (Comma Separated Variable)
  - XML file
  - Table (HTML Table)

(4)

Report Example

Module Name:
- hackyReportExample

Functionalities:
- A play ground for hackyReport
- Two commands
  - Category chart demo and test (adjustable category size and series size, category chart type selector)
  - XY chart (adjustable sample size, and XY chart type selector)

(5)

Jargon of Reports

Category
- A category can be a day, a person etc. A category carries category name, category value, [tool tip and/or drill down URL]

Category Series
- Is a list of universal categories. Ex, active time over several days.

Data Model
- Is a collection of category series or point series, which is used to create report.

Point
- A single (x,y) value pair represents a dot in X-Y plane. Both values are "Number" object

Point Series
- A series of points in X-Y chart.

(6)

Jargon of Reports (cont'd)

View

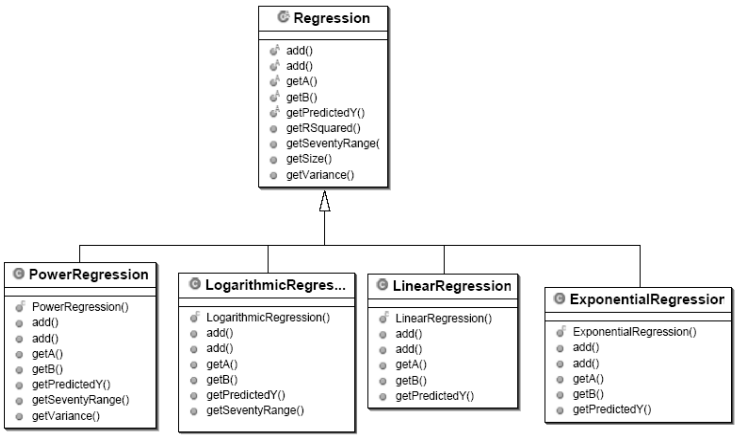
- Using data model to generate reports, including,
  - Chart view
  - XML view
  - CSV view
  - Table view

(7)

Part 1. Statistics

Package structure

- org.hackystat.stdext.statistic



(8)

Regression

Linear
- y = b + a * x

Logarithm
- y = b + a * ln(x)

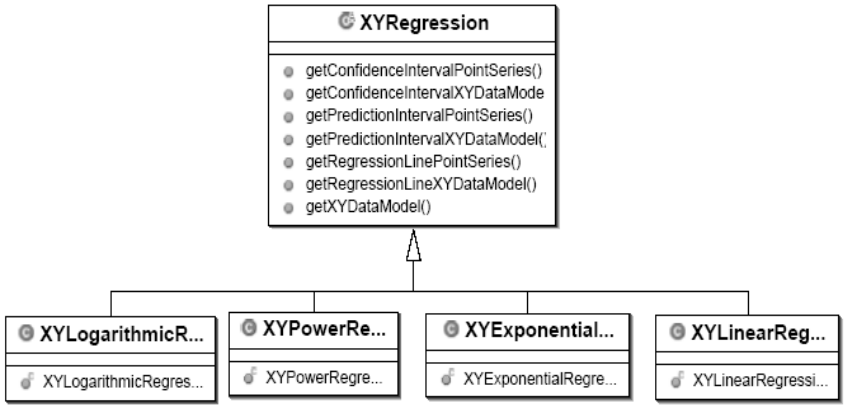
Power
- y = a * (x ^ b)

Exponential
- y = b * e ^ (a * x)

(9)

Statistics to Report

org.hackystat.stdext.report.xy.regression



One to one map from statistics regression to XY regression analysis

(10)

New Regression

1. Extends Regression.java, implements the abstract methods.
2. Add regression wrapper in hackyReport module at org.hackystat.stdext.report.xy.regression

(11)

Part 2. Report

Category
- Horizontal axis is the category, vertical is the value to the category or vice versa

Pie
- Similar as category report except that it has only one series
- Pie Chart display

X-Y
- Both axes are numeric value
- Can do regression analysis

(12)

Model-View Design

Model

- Data model which design how data is organized to generate reports.
  - CategoryDataModel
  - XYDataModel

View

- Chart view (create PNG chart)
- Table view (2-D display of data model)
- XML view (XML presentation of the data model)
- CSV view (Create comma separated variable file)

(13)

Package Structure

org.hackystat.stdext.report (Generic views)

- category (data model, views)
  - bin (assembly categories equally)
  - boxwhisker (implements box and whisker chart)
  - combine (combined chart for parallel comparison)
  - gantt (Gannt chart for display planning)
- pie (Provide pie chart to see sharing)
- xy (xy chart and regression)
- selector (Hackystat report type selector)

(14)

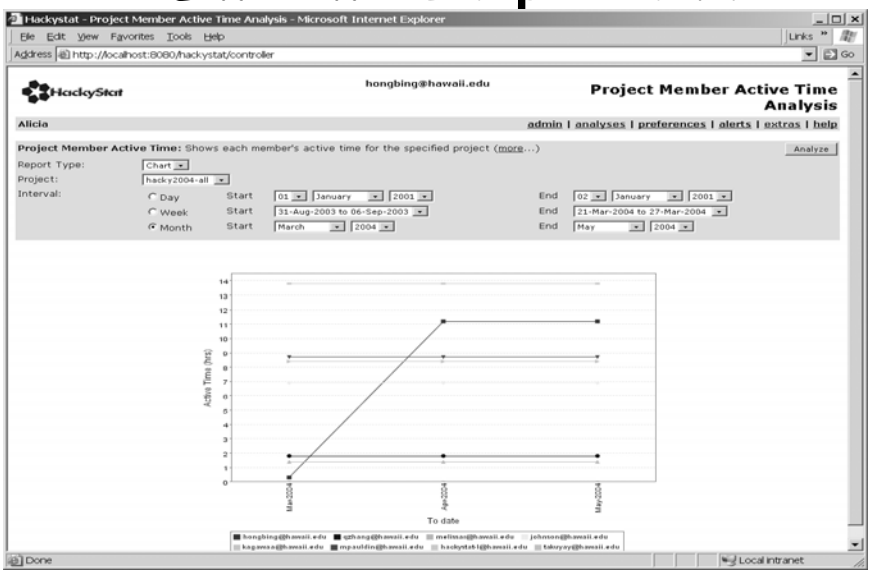
Regular Category Charts

Defined in CategoryChartView.java

- Line (Active Trend Analysis)
- Line and shape
  - Emphasize each value with solid square, triangle
- Vertical Bar
  - Histogram chart
- Stacked Vertical Bar
  - Stacked series for comparison

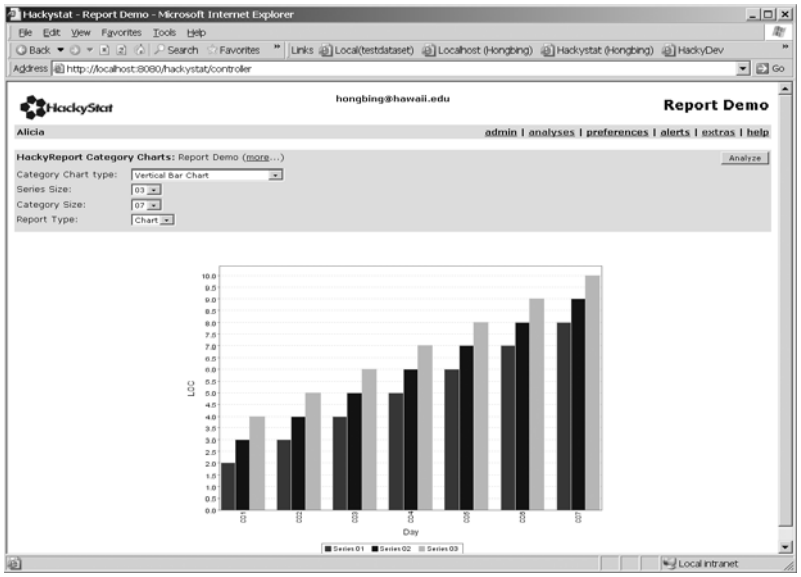
(15)

Line and Shape Chart



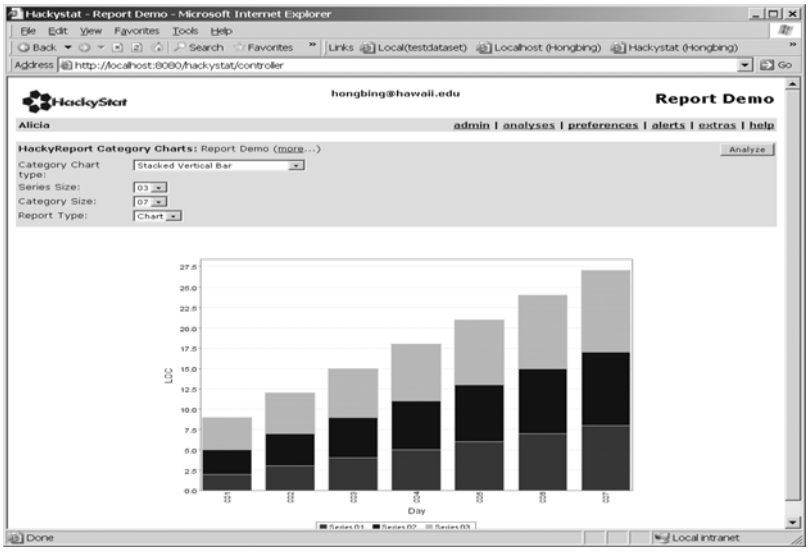
(16)

Vertical Bar



(17)

Stacked Vertical Bar



(18)

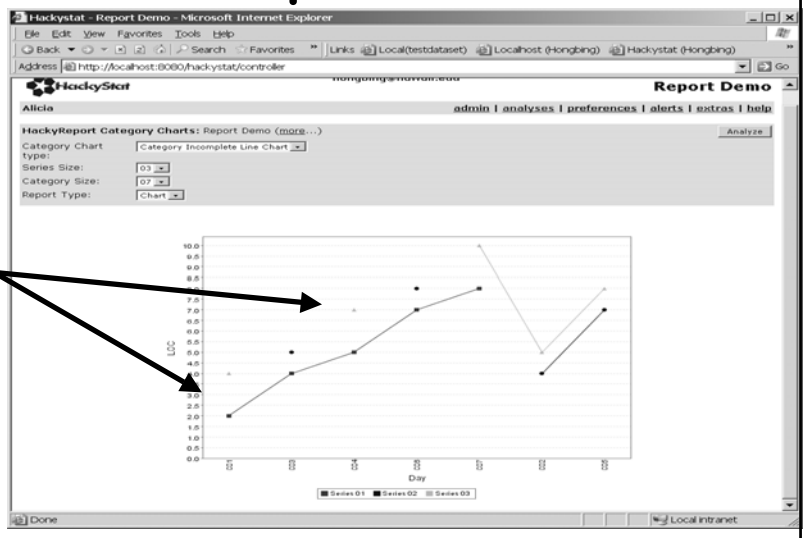
More Cool Stuff

Line chart and be incomplete

Histogram chart can be incomplete too

(19)

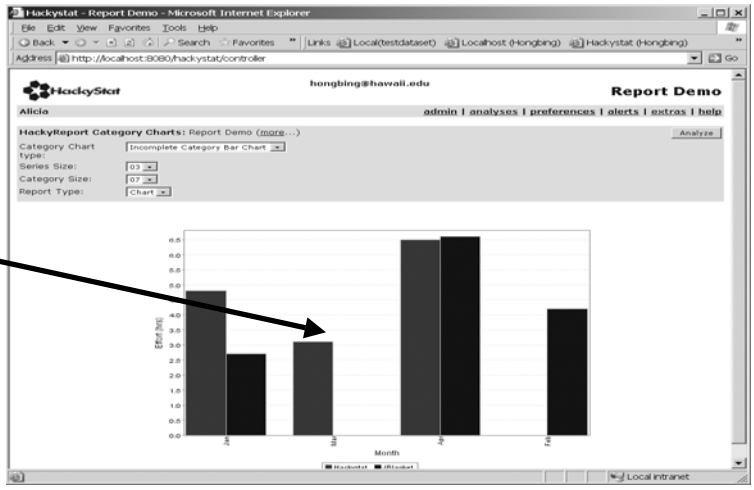
Incomplete Line



Series2 has no value to category c1 and c3

(20)

Incomplete Histogram



JBlanket has no value on Mar

(21)

Are above charts enough?

Probably not
- Histogram with average line
- Gannt chart for planning

Normal category charts
- Category is primitive object type (day, month, email, project)
- Value is numeric Number (int, float, double)

How to add new normal category chart?
- Define a new CategoryChartType
- Create new plot in CategoryChartView.java

(22)

Extraordinary Charts

Box Whisker
- Each category has a bunch of values in some range

Gantt chart
- Value is day period not primitive. Ex, From May 1 to May 5 is Hackystat conference

(23)

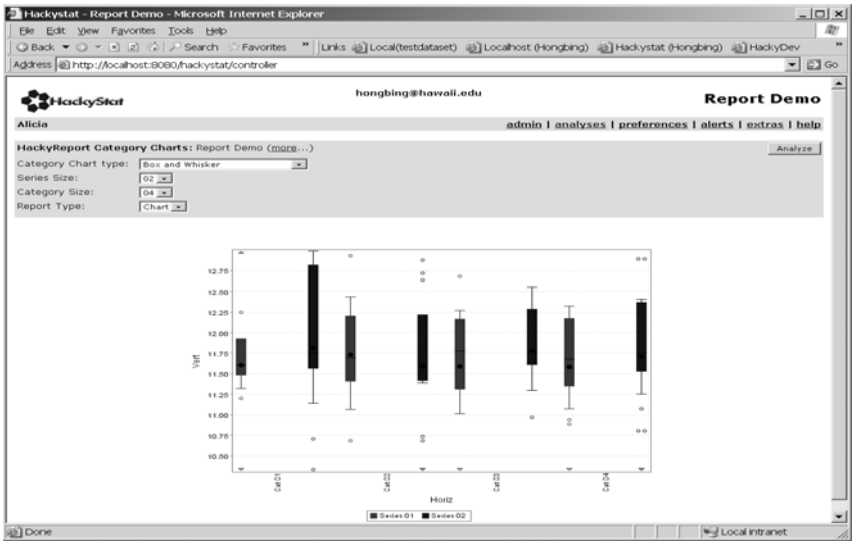
How they are implemented?

New category type
- BoxAndWhiskerCategory.java
  - Each category contains a list of Number values
- GanttCategory.java
  - Each category has start and end day

New views
- Reimplement Chart, Table, CSV, and XML views for them.

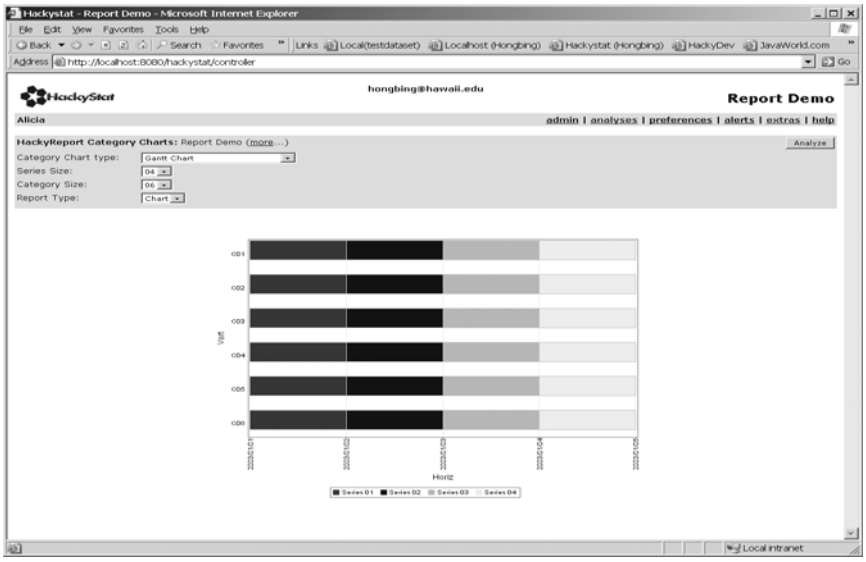
(24)

Box And Whisker Chart



(25)

Gantt Chart



(26)

New Extraordinary Chart?

Does JFreeChart support it?
- Because our charts depend on JFreeChart module

Extends Category.java
- Customize your category

Adds view support
- Chart
- Table
- XML
- CSV

Adds new CategoryChartType

Examples
- org.hackystat.stdext.report.category.gantt
- org.hackystat.stdext.report.category.boxwhisker

(27)

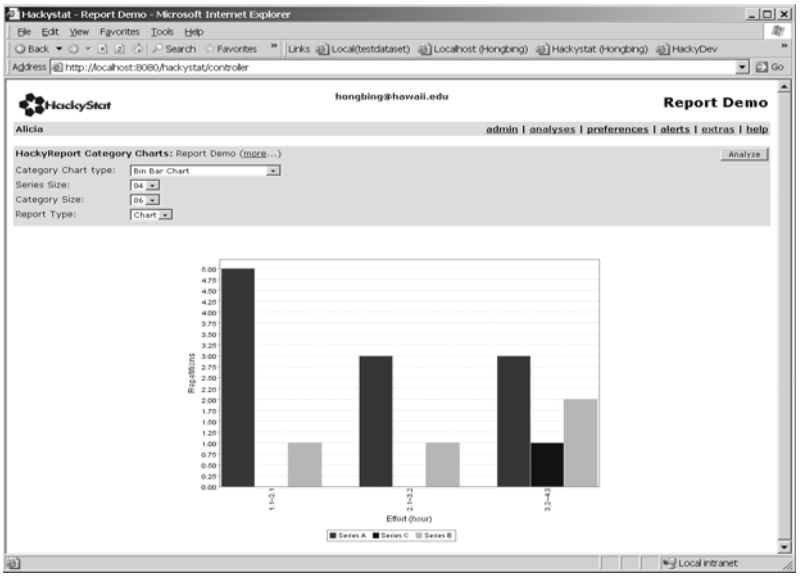
Charts Requires Pre-processing or Post-processing

Bin Chart
- Assembly category to bins to do classification

Combo Chart
- One chart is not enough, I want to compare two charts in the same analysis

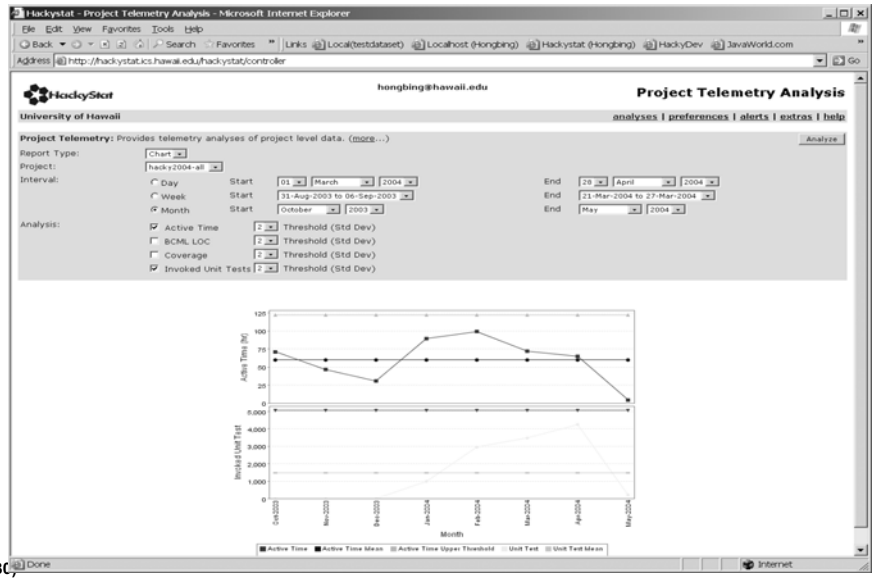
(28)

Bin Chart



(29)

Combo Chart



(30)

XY Charts

Not versatile as category chart

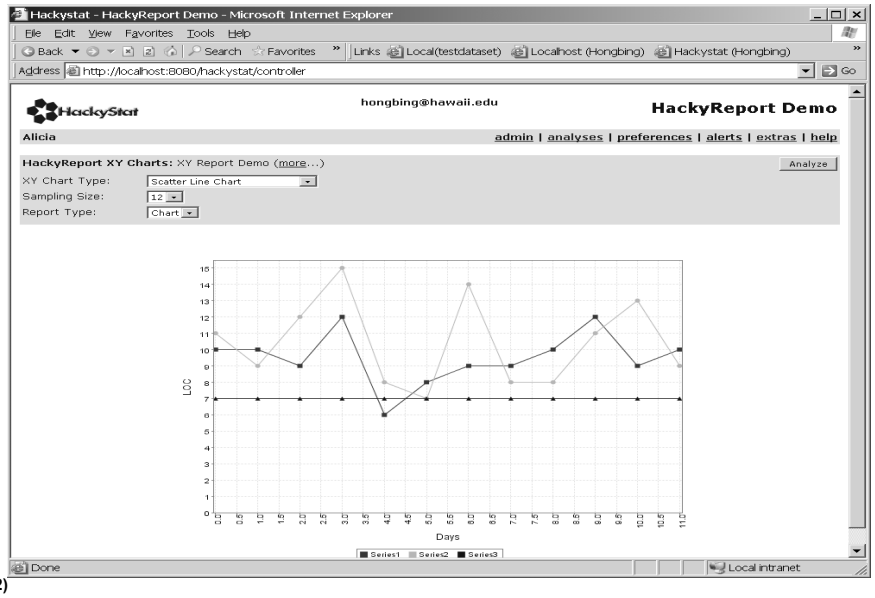
Lines can be
- Scatter (lot of value dots)
- Line with scatter
- Histogram

Model
- Point, PointSeries, XYDataModel

Views
- Chart
- Table
- XML
- CSV

(31)

Demo of Line Chart



(32)

New Chart Type

Updates XYChartView.java

(33)

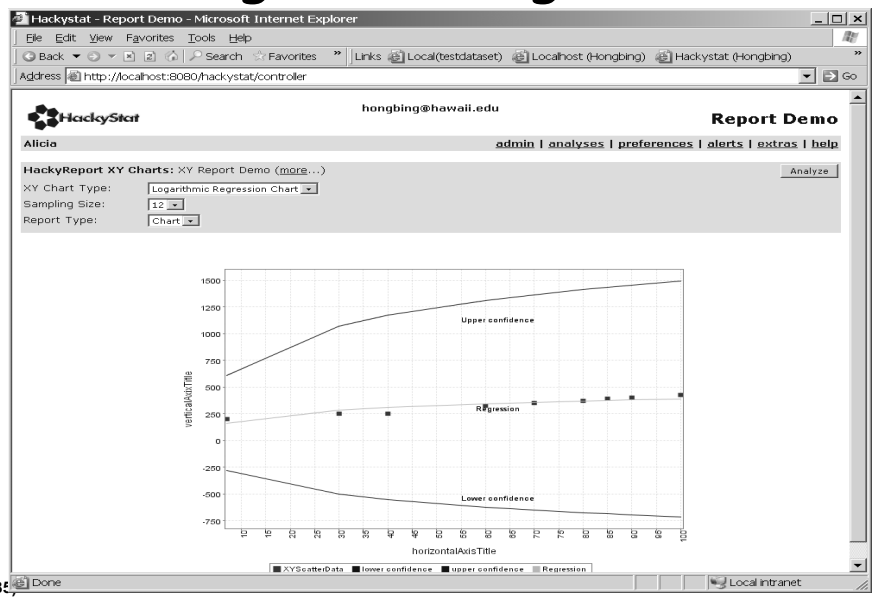
Regression Line Support

Package
- org.hackystat.stdext.report.xy.regression

Present Regressions
- Linear
- Logarithm
- Power
- Exponential

(34)

Logarithm Regression



(35)

New Regression

Extends XYRegression.java

Implement the abstract methods for regression

(36)

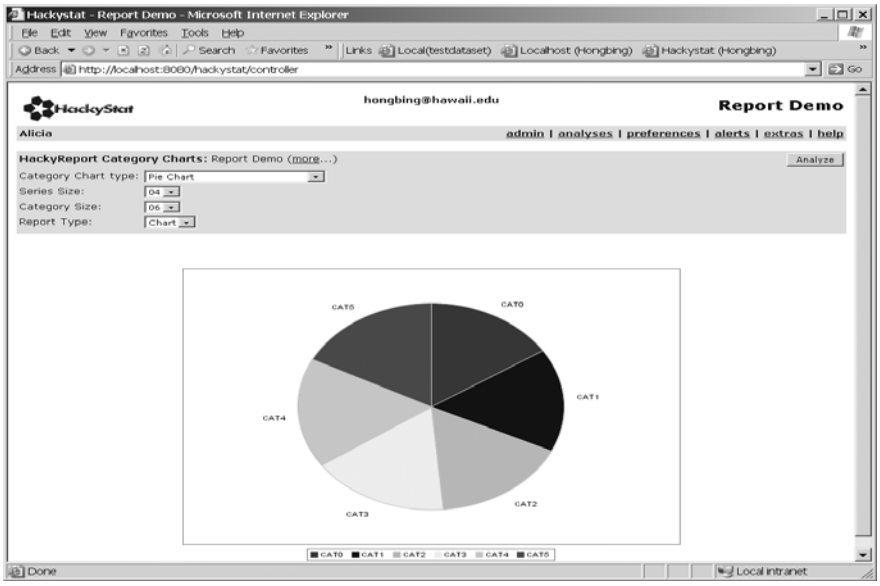
Pie Chart

Model
- Category, CategorySeries, PieDataModel
- Same as category data model except that there is only one series

View
- Chart
- Table
- XML
- CSV

(37)

Pie Chart Demo



(38)

Hacky Report and JFreeChart

Hackyreport is a wrapper and enhancement of JFreeChart for Hackystat analyses use.

- All charts need to be supported by JFreeChart. (Customized version 0.9.13)
- Views are designed for Hackystat analyses
  - Chart, XML and CSV are rendered to files
  - Table data is in two dimensional tables so that JSP page can iterate through it and display data in HTML table

(39)

Three Steps to Hackystat Chart

Step 1. Create data model

```
CategoryDataModel data = new CategoryDataModel();
   for (int i = 1; i <= seriesSize; i++) {
     CategorySeries s = new CategorySeries("Series " + i);
     if (i < 10) {
       s = new CategorySeries("Series 0" + i);
     }
     for (int j = 1; j <= categorySize; j++) {
       String category = "C" + j;
       if (j < 10) {
         category = "C0" + j;
       }

       s.addCategory(new Category(category, new Integer(i +
                     j), null,
"drilldown.jsp?example=CategoryData&info=" + category));
     }
     data.addSeries(s);
   }
```

(40)

Step 2 Creates Chart View

```
CategoryChartView view = new CategoryChartView(
                categoryData, chartType, user,
                xAxisLabel, yAxisLabel);
view.setJspChartAttribute(request, "Chart");
```

(41)

Step 3. Display Chart

```
<c:when test="${!empty Chart}">
      <c:out value="${Chart}"
        escapeXml="false"/>
</c:when>
```

(42)

How chart is drawn?

DataModel is mapped to JFreeChart dataset objects

Plot is created according to chart types

Chart is rendered out to PNG files upon request

(43)

Part 3. Report Example

Module name:

- hakcyReportExample

It's a working documentation how to create data model and use the view to have chart, table, xml or csv file for both category, xy and pie reports.

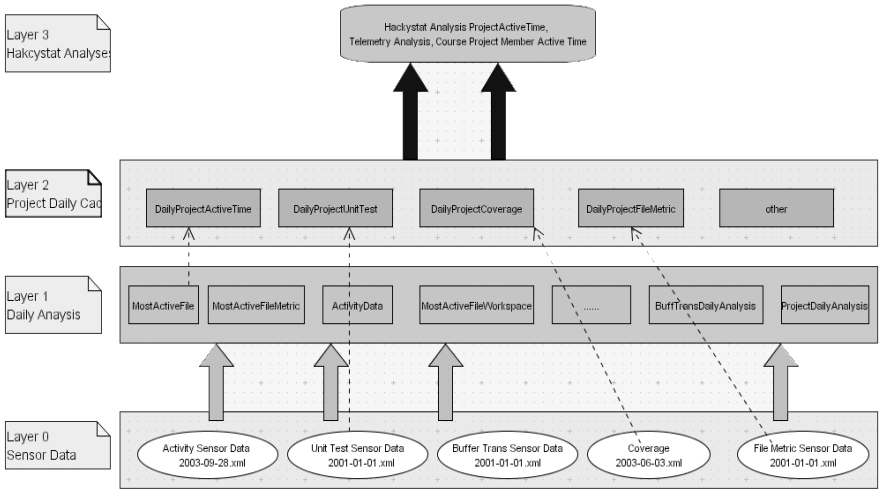
(44)

Key Standard Extension Concepts

Hongbing Kou
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Data Layers in Standard Extension



(2)

Overview

Hackystat Data Upstream Layers

Most Active File

Daily Diary

Daily Analysis

Workspace
- Workspace root
- Workspace Management

Project
- Project concept
- Project Setup
- Project Management

(3)

1. Most Active Files

The Origin
- Hackystat v1
  - JBuilder and Emacs Sensors
  - Activity data like "Open File", "Close File", "Save File" etc.
  - Cacoon was used to view sensor data
- Hackystat v2
  - Added "state change" activity to detect developer is active or leaves IDE unattended
  - "Open File", "Close File" were annoying because IDE can open 10 or more files when starts up and close them all when it shuts down

(4)

State Change Activity

State Change Activity data

- Thu Jan 08 09:36:22 HST 2004
- JBuilder
- State Change
- C:/work/hackyStdExt/src/org/hackystat/stdext/test/HackystatCommandUnitTest.java

What can cause state change?

- Active Buffer is changed because of developers typed in some words
- JBuilder sensor wakes up every 30 seconds to check whether buffer is changed. If so it sends out "state change" data.

(5)

Is the developer working?

If there is "state change" activity developer must be working.

Five minutes was proposed by Philip as inspection period because it is neither small nor big.

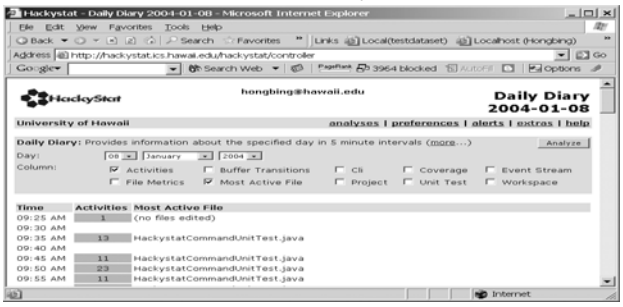
- It's a reasonable magic number
- Finer grain (1 min) is not very much superb than 5 minutes interval. http://csdl.ics.hawaii.edu/techreports/02-09/02-09.pdf

(6)

Most Active File

Representation of a five minutes period

- A list of "state change" activities
- One single file or multiple files
- The file with most number of "state change" activities is nominated as the representation of this five minutes period.



(7)

Most Active File Made its Fame

Abstract developers' work successfully

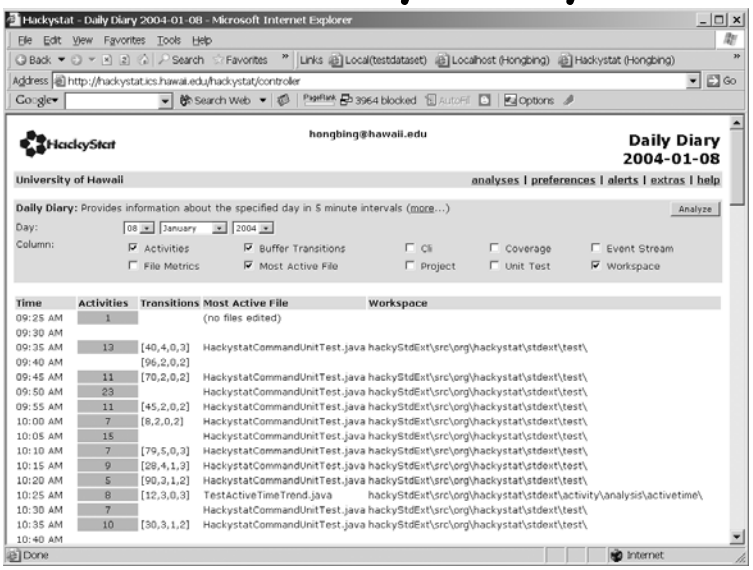
- Simple
- A notation that makes us be able to think above stream of IDE activities
- Extensible
  - Most active workspace
  - Active project
- Represents "Effort" or "Active Time" (priceless value)

(8)

Activity Sensor Data

(9)

2. Daily Diary



(10)

Inside Daily Diary

Inspired from five minutes interval
- A day has 288 cells using five minutes as unit
- (24hr * 60min/hr) / 5min = 288
- DayArray.java

Daily analysis
- Each column represents one kind of analysis Activities, Buffer Trans, Most active file etc.

(11)

DayArray.java

Summary
- Manipulates array with 288 elements in a day

Package
- org.hackystat.stdext.dailyanalysis.dayarray

APIs
- Object get()
- boolean hasData(int index)
- void set(int index, Object obj)
- void set(Date time, Object obj)
- static int timeToIndex(Date time)
- static String indexToTimeString(int index)
- DayArrayIterator iterator()

(12)

Extension of DayArray

DayArrayList
- Each element contains a list structure

DayArrayMap
- Each element contains a map

DayArrayIterator
- Extended `iterator()` implementation
- `hasNext(), hasNextNonempty()`
- `next(), nextNonempty()`
- `nonemptyIndex()`

(13)

Design of Daily Analysis

Package
- org.hackystat.stdext.dailyanalysis

Classes
- `DailyAnalysisManager.java`
  - Loads dailys analysis configuration files
  - Folder `hackystat/web-inf/dailyanalysis`
- `DailyAnalysis.java`
  - Provides common methods for daily dairy
  - Super class of all daily analyses
- `DailyAnalysisCache.java`
  - Provides cache function for daily analysis and access point

(14)

Instantiating Daily Analysis?

Daily Analyses

- ActivityData, MostActiveFile, File Metrics BufferTrans etc

Instantiating daily analysis objects

```
ActivityData activityData = (ActivityData)
    DailyAnalysisCache.getInstance().get(user,
                          day, ActivityData.class);
```

(15)

New Daily Analysis

1. Creates your class which extends

   `DailyAnalysis.java`

2. Create daily analysis configuration file dailyanalysis.XXXX.xml in format

```
<dailyanalysis>
 <analysis name="Project"
     class="org.hackystat.stdext.project.
      dailyanalysis.ProjectDailyAnalysis"
     enable="true"/>
</dailyanalysis>
```

3. Put your configuration file in folder hackystat/web-inf/dailyanalysis

(16)

Use Daily Analysis or not?

Up to your analysis requirement
- Activity Data
- Most Active Time
- Most Active File Workspace
- Most Active File Metric
- Buffer Trans

Does DailyAnalysis loss fine-grained information for your analysis?

New Daily Analysis if necessary

(17)

3. Workspace

Def.
- A canonical representation of a directory in file system which is cross-platform independent and case-sensitive.

Examples
- Windows workspace
  - `C:\work\hackyKernel\src\org\hackystat\kernel\user`
- Unix-like workspace
  - `\export\home\hongbing\hackyKernel\src\org\hackystat\kernel\user`

(18)

Workspace Root

Workspace root

- A workspace can have none or one root.
- Workspace root can be an ancestor, descendant or just itself.
- To workspace `C:\work\hackyReport\`
  - Ancestor `C:\, C:\work\`
  - Itself `C:\work\hackyReport\`
  - Its descendent `C:\work\hackyReport\src\`

(19)

Motivation for Workspace Root

A developer can work in different platform on the same project which has different directory

- `C:\work\hackyReport\`
- `\usr\hongbing\cvs\hackyReport\`

If `C:\work\ is the root` for the first workspace and `\usr\hongbing\cvs` is the root to second workspace we can get same workspace hackyReport

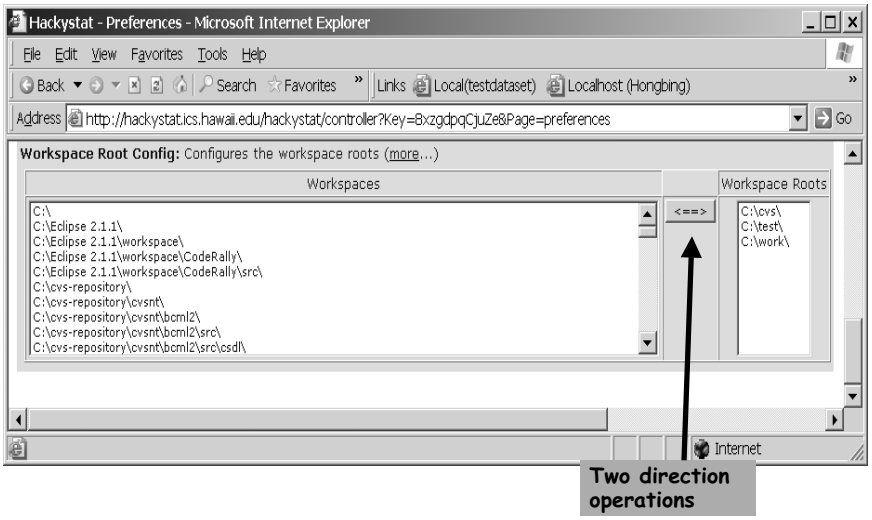
Work on different platform can be merged together

Other cases

- Same project in different directory
- Two or more people's data on same project can be combined together if workspaces are same after trimming off roots.

(20)

Workspace Root Configuration



(21)

Workspace Master

WorkspaceManager.java

- Singleton
- Loads Activity, File Metric, and CVS commit data to construct workspace. Ex. to file `C:\work\hackyReport\build.xml` it will generate workspace
  - `C:\`
  - `C:\work\`
  - `C:\work\hackyReport\`
- Listens to new sensor data and updates workspace repository
- Set workspace root appropriately as your request
- Hides workspaces as your request

(22)

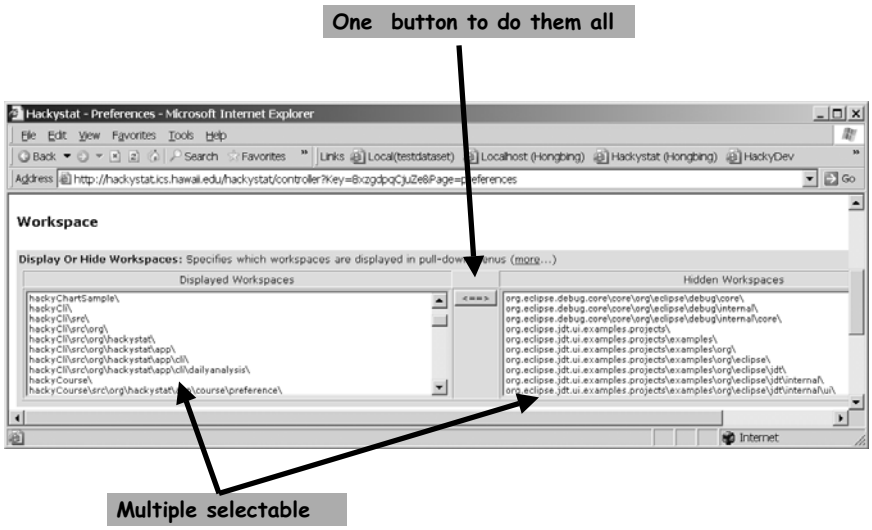
Hide Workspace

Why we want to hide workspace?

- Workspace list became huge. Ex. I have 1812 workspace entries.
- Not all workspaces are interesting
  - `C:\work\org.eclipse.debug.core\`
  - `C:\work\hackyStdExt\src\org\hackystat\stdext\activity\`

(23)

Display or Hide Workspace



(24)

Something to know about workspace

It's self managed

- You don't need to worry about when and how workspace is created.

Workspace roots cannot be nested

- `C:\work\` and `C:\work\test\` can NOT be workspace roots at the same time.

Workspaces are cached

- It only loads sensor data once when the first time you ask for workspace

(25)

Workspace APIs

`WorkspaceCache.java`

- You have file or folder but you want workspace instance
  - `Workspace getWorkspace(User user, String rawPath)`
  - `WorkspaceFile getWorkspaceFile(User user, String rawFileNamePath)`

`WorkspaceManager.java`

- You want all displayable workspace
  - `Set getDisplayWorkspaceSet()`

(26)

Workspace API (cont.)

Workspace.java

- `String getTrimmedPath()` Workspaces excluding root
- `boolean isSameWorkspace(Workspace workspace)` Two workspaces are equal or not
- `boolean isSubWorkspace(Workspace workspace)` Is the workspace given workspace's ancestor or not?

(27)

Individual to Collaboration

Most active file and workspace are both personal information

You can only see your data and analysis is personal

- Daily Diary – View your sensor data
- Workspace Effort/ActiveTime (obsolete)
  - Effort/ActiveTime on a single folder

Project was proposed for collaboration

(28)

4. Project

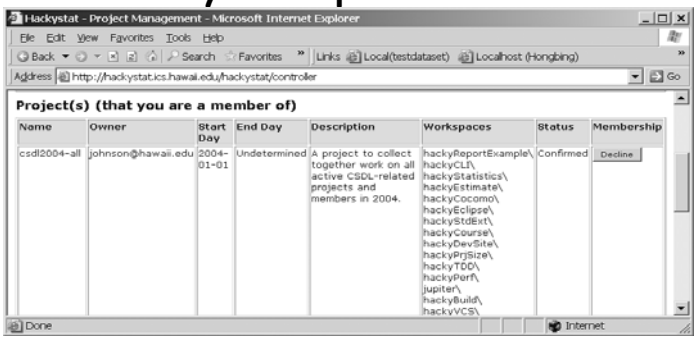
Elements of project
- Name
- Owner (leader, manager)
- Start day
- End day (Optional)
- Developers (Hackystat Key? emails?)
- CVS Modules (hackyKernel, hackyStdExt)
- Description

(29)

Project in Hackystat

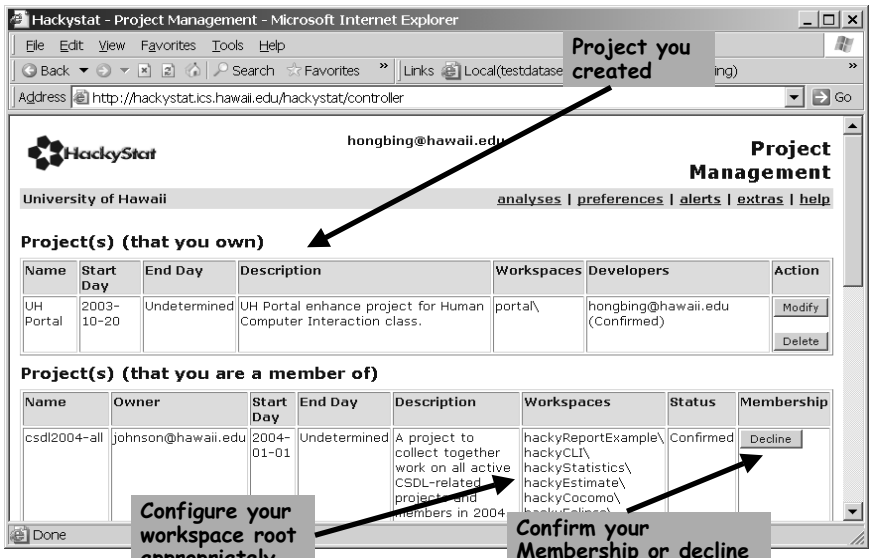
Developers by emails
- Hackystat key is confidential
- Invitation-acknowledge model setup
- Email notification

Coordinated by workspace



(30)

Manage Projects



(31)

Project APIs

ProjectManager.java

- `Project getProject(String name)`
- `List getProjectsOwned(User owner)`
- `List getWorkingOnProjects(User user)`
- `Iterator iterator()`

Project.java

- `User getOwner()`
- `Set getMembers()`
- `Day getStartDay()`
- `Set getWorkspaceSet()`

(32)

Project Daily Cache

Daily based

Foundation for project analyses

Cached for Better Performance

Two project daily cache types

- State
- Aggregated

Defined Daily Cache

- DailyProjectCoverage.java
- DailyProjectFileMetric.java
- DailyProjectActiveTime.java
- DailyProjectUnitTest.java

(33)

Project Daily Cache (cont)

Instantiation

- DailyProjectActiveTime.getInstance(project, day)
- DailyProjectUnitTest.getInstance(project, day)
- ...

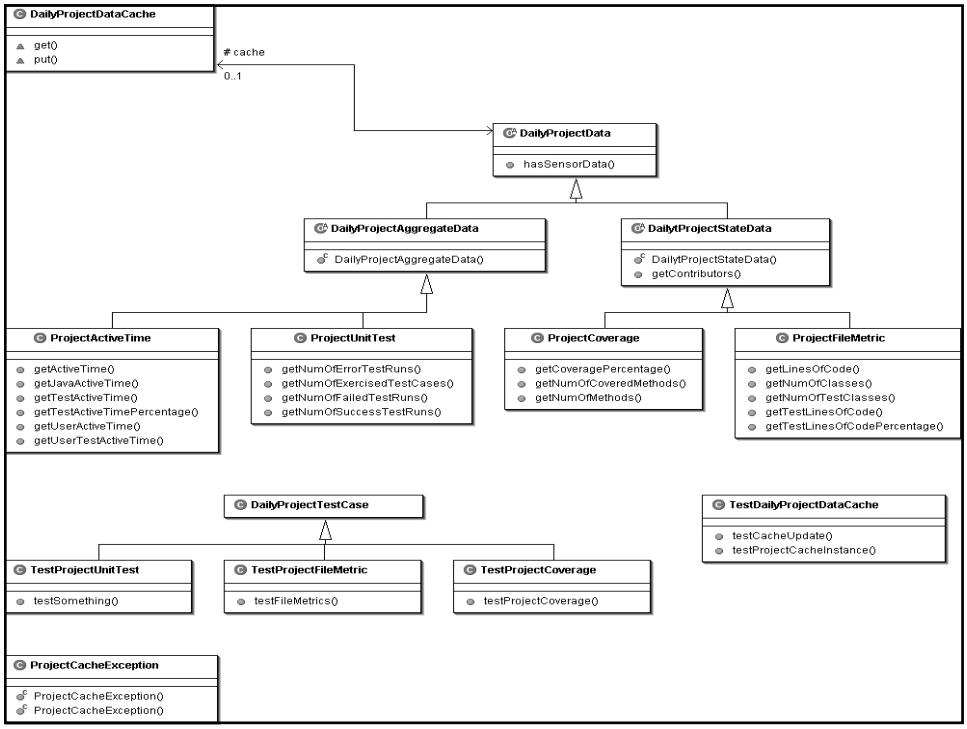
Code Snippet from Telemetry Analysis

```
for (Iterator i = interval.iterator(); i.hasNext();) {
  Day day = (Day) i.next();
  Category category = new Category(day,
        new Double(DailyProjectActiveTime.getInstance(
              project, day).getActiveTime()));
  categorySeries.addCategory(category);
}
```

Reference

- http://csdl.ics.hawaii.edu/~hongbing/hackystat/design/ProjectCache.jpg

(34)



More Stuff?

(36)

Anatomy of hackyVisualStudio

Qin Zhang
Collaborative Software Development Laboratory
Communication & Information Sciences
University of Hawaii, Manoa

May 2004

(1)

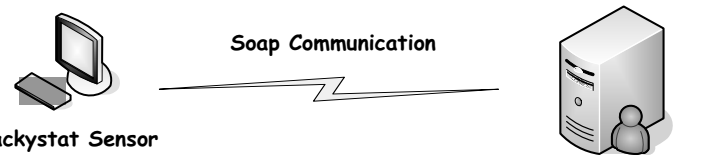
Visual Studio Sensor Requirements

Collecting activity data

- Open file, Close file, State change, etc.

(2)

General Hackystat Sensor Design

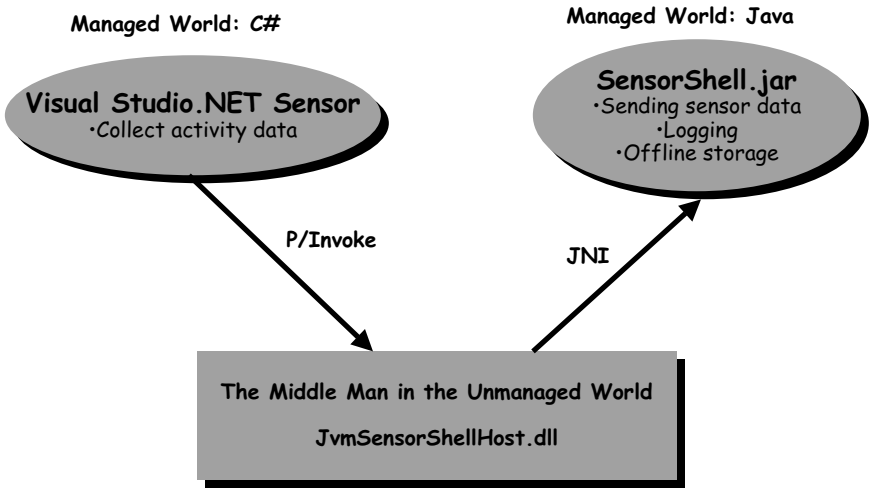


- Collecting data
- Sending data via Soap
- Logging
- Offline Storage

SensorShell.jar

(3)

Invoking Java Functions in .NET



(4)

Interop Implementation 1
The Middle Man

C++ definition for JvmSensorShellHost.dll

```
extern "C"
{
    __declspec(dllexport) bool CreateJVM(const char* jvm, const char* class);
    __declspec(dllexport) bool DestroyJVM();
    __declspec(dllexport) bool Send();
    __declspec(dllexport) bool DoCommand(…);
}
```

(5)

Interop Implementation 2
How the middle man calls Java Code?

Sample: to invoke java method *send()* in java class

org.hackystat.stdext.sensor.visualstudio.SensorShellWrapper

C++ and JNI (Java Native Interface)

```
static char* javaClass =
"org/hackystat/stdext/sensor/visualstudio/SensorShellWrapper"
//Find the java class
jclass theClass = env->FileClass(javaClass);
//Find the method in the class
jmethodID theMethod = env->GetStaticMethodID(theClass, "send", "()Z");
//Invoke the method
jboolean result = env->CallStaticBooleanMethod(theClass, theMethod);
```

(6)

Interop Implementation 3
How C# code calls the middle man?

Sample: to invoke native function

Send() exported in JvmSensorShellHost.dll

C# and P/Invoke

```
class SensorShellAdapter
{
   [DllImport("JvmSensorShellHost.dll",
              EntryPoint="Send",
              CallingConvention=CallingConvention.Cdecl)]
   public static extern bool Send();

   //…  other functions

}
```

(7)

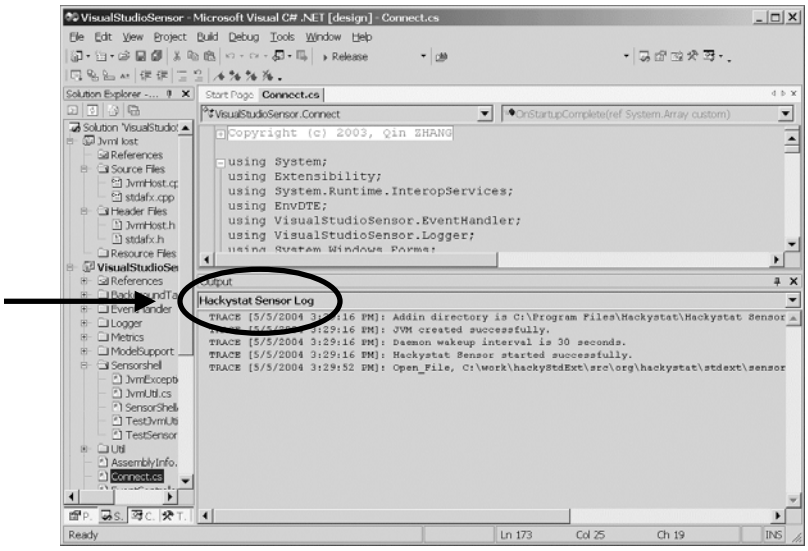
VS.NET Sensor Implementation

The sensor is implemented as a Visual Studio .NET add-in.

```
public class Connect : IDTExtensibility2
{
   public void OnConnect(…)
   {
     //register IDE events we are interested in,
     //and supply call-back functions.
     //
     //IDE will call your function
   }
 }
```

(8)

VS.NET Sensor at Work



(9)

CVS Repository

:pserver:anonymous@hackydev.ics.hawaii.edu:/cvsnt

Check out:

hackyStdExt/src/org/hackystat/stdext/sensor/visualstudio

(10)

Thank you!

(11)

The Anatomy Of hackyOffice

The requirements, design, and implementation of the Microsoft Office sensor

Burt Leung
Collaborative Software Development Laboratory
Communication & Information Sciences
University of Hawaii, Manoa

May 2004

(1)

What Is hackyOffice?

A Hackystat sensor that collects activity data from Microsoft Office applications. E.g.

- Opening a file
- Closing a file
- File state changes

(2)

Why Is It Important?

Exploration/Analysis of documentation efforts

Analysis of non-coding development efforts

(3)

Which Applications Are We Talking About?

All original Office applications from Office 2000

- Microsoft Word
- Microsoft Excel
- Microsoft FrontPage
- Microsoft PowerPoint
- Microsoft Access
- Microsoft Publisher
- Microsoft Project

(4)

Great! What Can't This Sensor Do?

Newer MS Office applications aren't supported

- OneNote (API not available in initial release)
- InfoPath

(5)

Have I Seen This Before?

hackyOffice is based on hackyVisualStudio (by Cedric Zhang)

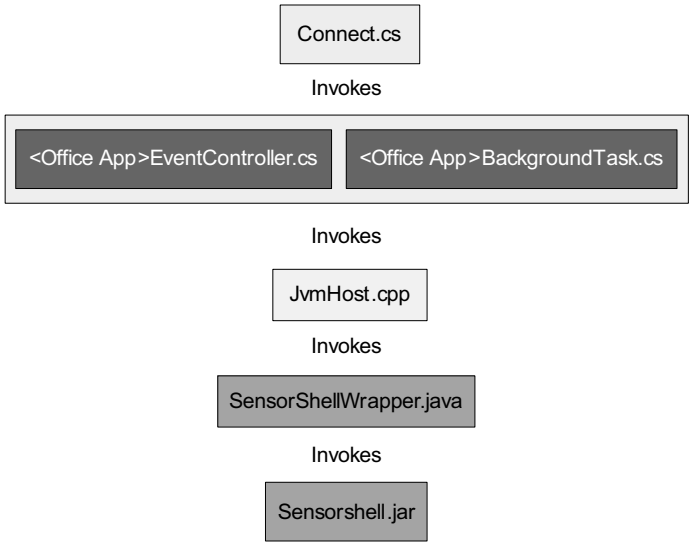
- Uses its Java-to-.Net communication code
- Uses its basic code structure, E.g.
  - Central controller
  - Event handler hooks

(6)

The Master Design Of hackyOffice

(7)

Basic Class Relationships



(8)

PIAs and GAC? What?!?

PIA: Primary Interop Assembly:
- Office Application library as metadata
- A PIA can "wrap" more than one version of the same type library [ref]
- hackyOffice uses the individual Office app's PIA to work with "any" version of Office

GAC: Global Assembly Cache:
- This is like a global library for .Net based apps
- hackyOffice's reusable components are placed in the GAC for easy reference

(9)

The MS Word Sensor Component

Classes
- WordEventController.cs
- WordBackgroundTask.cs

Detects statechange by temporal changes in
- Character count

(10)

The Excel Sensor Component

Classes
- ExcelEventController.cs
- ExcelBackgroundTask.cs

Detects statechange by temporal changes in
- Count of cells
- Count of total characters

(11)

The PowerPoint Sensor Component

Classes
- PowerPointEventController.cs
- PowerPointBackgroundTask.cs

Detects statechange by temporal changes in
- Total "Shapes" of all slides
  - A "Shape" is an object that can hold text and/or multimedia objects
- Total character count of all Shapes

(12)

The FrontPage Sensor Component

Classes:
- FrontPageEventController.cs
- FrontPageBackgroundTask.cs

Detects statechange by temporal changes in
- File size

Notes
- Pages must be initially saved first

(13)

The Access Sensor Component

Classes
- AccessEventController.cs
- AccessBackgroundTask.cs

Detects statechange by temporal changes in
- Count of controls on forms and reports

(14)

The Publisher Sensor Component

Classes:
- PublisherEventController.cs
- PublisherBackgroundTask.cs

Detection of statechange same as for PowerPoint sensor (by analyzing Shapes)

(15)

The MS Project Sensor Component

Classes
- ProjectEventController.cs
- ProjectBackgroundTask.cs

Detects statechange by temporal changes in
- Count of "Tasks"
- Count of "Resources"

(16)

The MS Visio Sensor Component

Classes
- VisioEventHandler.cs
- VisioBackgroundTask.cs

Detects statechange in a manner similar to the PowerPoint component (via Shapes)

Notes
- Visio 2000 does not support Add-Ins

(17)

Some Cons

Sensor cannot detect reorganization of items in Office applications

Some activity detection is not supported in certain Office applications

(18)

Some Pros

Sensor for all Office applications can be globally disabled/disable
- Via ENABLE_OFFICE_SENSOR property in sensor.properties file

Install the sensor once and all Office applications are supported
- Even if application is installed later

(19)

Developer Tips and Information

(20)

You Mean I Can't Go Solo?!?

Before using or developing hackyOffice you need:

- The .Net framework installed
- A Java runtime (same version as required by hackyStat)

(21)

Microsoft Virtual PC Is A Lifesaver!

Testing deployment should be done on a clean machine

- hackyOffice development requires changes to the registry
  - As time passes the registry may get clogged with conflicting entries
  - Testing installs to dev machine may become unreliable

A Virtual machine program allows for easy testing, E.g.

- MS Virtual PC
- VMware

(22)

Anatomy of the Ant/LOCC sensor

Michael Paulding
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Introduction

Ant is a build tool similar to make, but Java-based.

LOCC is an extensible system for producing hierarchical, incremental measurements of work product size (primary metric being lines of code).

They are friends.

(2)

LOCC – Lines of Code Counter

LOCC provides the user with the ability to gather various metrics from source code.

For Java code, this includes total lines (excluding comments and blank lines) and number of methods per class.

LOCC is grammar based, meaning it actually parses source code and counts LOC only for true code. (In words of one user, it is "deadly accurate")

(3)

LOCC – Lines of Code Counter

LOCC is currently able to parse 3 grammars: Java, C++ and text. (Fortran 90/95 – Summer 2004)

LOCC provides a set of output formats for data presentation: text, CSV, Leap and XML.

All parameters can be passed via command line in console, via drop-down menus in the GUI and via parameters in the Ant build.xml.

(4)

Collaboration between Ant/LOCC

LOCC is packaged as a binary jar file for invocation through command line or Swing GUI.

Ant provides capability to execute jar files during build process.

This integration provides developer with ability to gather size metrics (LOC, methods per class) automatically during build.

Following is an example of the integration…

(5)

Integration of Ant/LOCC in Build Process

```
<target name="locc" description="Runs LOCC to generate source
    code metrics, then sends results to Hackystat.">
    <taskdef name="locc"
    classname="csdl.locc.tools.ant.LOCCTaskdef"/>

    <taskdef name="hacky-locc"
    classname="org.hackystat.stdext.sensor.ant.locc.LoccSensor"/>

    <mkdir dir="${build.dir}/locc"/>

    <locc sizetype="javaline" outfile="locc-all.xml"
    outdir="${build.dir}/locc">

    <fileset dir="${src.dir}"> <include name="**/*.java"/>
    </fileset> </locc>

    <hacky-locc verbose="on">
    <fileset dir="${build.dir}/locc"> <include name="locc-
    all.xml"/> </fileset> </hacky-locc> </target>
```

(6)

Integration of Ant/LOCC/Hackystat

In previous slide, there was a taskdef named hacky-locc.

```
 <taskdef name="hacky-locc"
classname="org.hackystat.stdext.sensor.ant.locc.LoccSensor"/>
```

The LoccSensor class reads XML data produced by the locc task and interprets it into relevant FileMetric data for LOC and methods per class.

Remember FileMetrics from Hongbing's presentation on Tuesday?

FileMetric data is available for display in several Hackystat analyses (e.g. Daily Diary)

(7)

Anatomy of hackyEclipse

Takuya Yamashita
Collaborative Software Development Laboratory
Information & Computer Sciences
University of Hawaii at Manoa

(1)

hackyEclipse project

Goals:
- Implements sensor for Eclipse platform.

Approach:
- Create sensor as a Plug-in
  - Follow plug-in development environment
- Startup sensor on startup
  - Hook startup extension point
  - Instantiate sensor shell in earlyStartup()

(2)

Plug-in Installation

Installation
- **Update Manager**
  http://hackystat.ics.hawaii.edu/hackystat/download/eclipse
- **Manual installation**
  http://hackystat.ics.hawaii.edu/hackystat/controller?Key=&Page=help&Subpage=install&Sensor=Eclipse

Installation Directories
- **ECLIPSE_HOME\plugins\ org.hackystat.stdext.sensor_5.5.402.3x**
- **ECLIPSE_HOME\features\ org.hackystat.stdext.sensor_5.5.402.3x**

(3)

Plug-in Structure

org.hackystat.stdext.sensor_5.5.428
- **plugin.xml – configuration file**
- **sensorshell.jar – hackystat sensor shell**
  - **Common packages for all sensors**
- **sensor.eclipse.3x.jar**
  - **Eclipse-specific package**
- **Logo, license info, etc**

(4)

Plug-in Structure cont.

- **Plugin.xml file**

```
<plugin id="org.hackystat.stdext.sensor "
 class="org.hackystat.stdext.sensor.eclipse.EclipseSen
 sorPlugin">
  <runtime>
   <library name="sensorshell.jar">
      <export name="*"/>
    </library>
    <library name="sensor.eclipse.jar">
      <export name="*"/>
    </library>
  </runtime>
  <requires >
   <import plugin="org.eclipse.ui"/>
    …
 </requrires>
  <extension point="org.eclipse.ui.startup" />
```

(5)

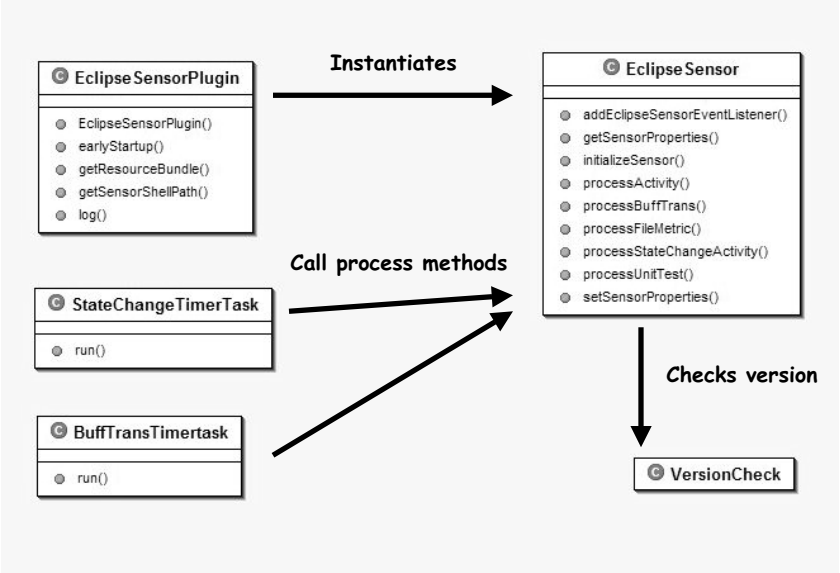
Plug-in Structure cont.

- **public void earlyStartup()**
  - Implements **IStartup - Early instantiation**

```
public class EclipseSensorPlugin
   extends AbstractUIPlugin
    implements IStartup {
    …
    public void earlyStartup() {
      // Instantiates Hacky Eclipse
 sensor
      EclipseSensor.getInstance();
    }
    …
}
```

(6)

Plug-in Structure cont.



(7)

Sensor Data Collected

Sensor Data Type (SDT)

- Activity data
  - Open file / Close file
  - Open project / Close project
  - Save file
  - State change
  - Build error
  - Breakpoint sets
- FileMetric data
  - WMC, DIT, NOC, RFC, Size, Last modified, (LOC)
- UnitTest data
  - Success, Failure, Error
- Buffer Transitions data

(8)

Sensor Data Collected cont.

Listener class
- IDocumentListener (buffer)
- IResourceChangeListener (file save, etc)
- IPartListener (file open/close, activavtion)
- IWindowListener (window activation)
- ITestRunListener (JUnit invocation)

(9)

Sensor Data Collected cont.

IWindowListener (Window Activation)

```
InitializeListeners() {
  EclipseSensorPlugin plugin = EclipseSensorPlugin.getInstance();
  IWorkbench workbench = plugin.getWorkbench();
   workbench.addWindowListener(new WindowListenerAdapter());
  …
```

IPartListener (Open / Close / File Activation)

```
…
 IWorkbenchWindow activeWindow = workbench.getWorkbenchWindows()[0];
IWorkbenchPage activePage = activeWindow.getActivePage();
activePage.addPartListener(new PartListenerAdapter());
```

(10)

Sensor Data Collected cont.

IDocumentListener (buffer)

```
IEditorPart activeEditorPart = activePage.getActiveEditor();
ITextEditor activeTextEditor = (ITextEditor) activeEditorPart;
IDocumentProvider provider = activeTextEditor.getDocumentProvider();
IDocument document =
provider.getDocument(activeEditorPart.getEditorInput());
document.addDocumentListener(new DocumentListenerAdapter())
```

(11)

Sensor Data Collected cont.

IResourceChangeListener (File save, etc)

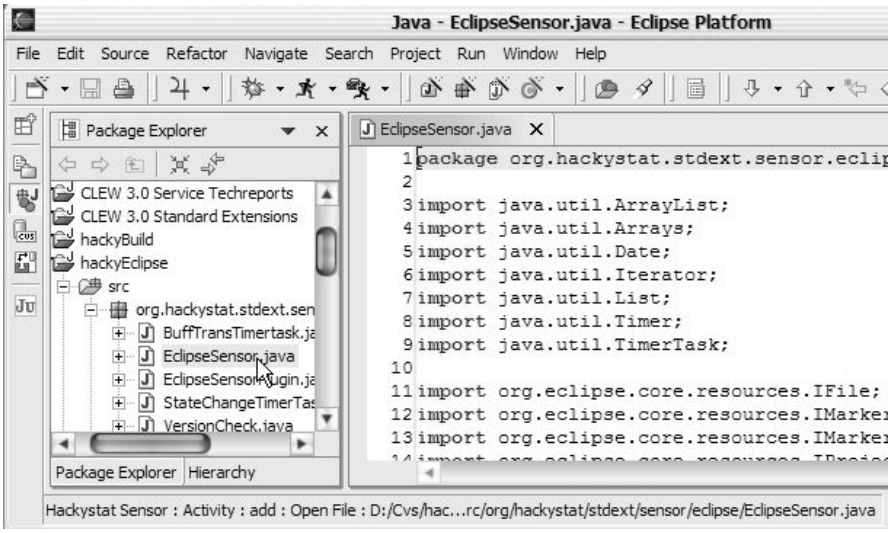
```
IWorkspace workspace = ResourcesPlugin.getWorkspace();
workspace.addResourceChangeListener(new ResourceChangeAdapter(),
                                    IResourceChangeEvent.POST_CHANGE);
```

ITestRunnerListener (Success / Failure / Error)

```
JUnitPlugin plugin = JUnitPlugin.getDefault();
plugin.addTestRunListener(EclipseJUnitListener(this);
```

(12)

Sensor Data Collected cont.



(13)

Gathering Sensor Data cont.

```
SensorShell started at: 05/06/2004 15:34:57
Type 'help' for a list of commands.
Host: http://hackystat.ics.hawaii.edu/ is available and key is valid.
Defined SDT:     ActivityDefined SDT:     BadData (no shell command)Defined SDT:     BuffTransDefined SDT:
Defined shell command: Activity
Defined shell command: Coverage
Defined shell command: UnitTest
Defined shell command: Commit
Defined shell command: BuffTrans
Defined shell command: FileMetric
#> AutoSend [10]
AutoSend OK (set to 10 minutes)
AutoSend enabled every 10 minutes.
Checking for offline data to recover.
No offline data found.
#> Activity [setTool, Eclipse]
setTool OK
#> Activity [add, Open File, D:/Cvs/hackyEclipse/src/org/hackystat/stdext/sensor/eclipse/EclipseSensor.java]
Activity add OK (1 total)
#> FileMetric [addJava, D:/Cvs/hackyEclipse/src/org/hackystat/stdext/sensor/eclipse/EclipseSensor.java, org.h
FileMetric add OK (1 total)
[BCML org.hackystat.stdext.sensor.eclipse.EclipseSensor cbo:51 dit:1 loc:291 noc:0 rfc:63 size:17292 wmc:39 ]
#> Activity [statechange, D:/Cvs/hackyEclipse/src/org/hackystat/stdext/sensor/eclipse/EclipseSensor.java, 50
Activity statechange OK (initial file name and buffer size)
#> Activity [add, Close File, D:/Cvs/hackyEclipse/src/org/hackystat/stdext/sensor/eclipse/EclipseSensor.java]
Activity add OK (2 total)
#> send
Sending sensor data (05/06 15:37:35)
  Activity: Send OK (2 entries)
  Cli: Send OK (No entries to send.)
  Ping: Ping OK (contacted server http://hackystat.ics.hawaii.edu/ with valid key.)
  Coverage: Send OK (No entries to send.)
  AutoSend: AutoSend OK ('send' command ignored)
  BuffTrans: Send OK (No entries to send.)
  Commit: Send OK (No entries to send.)
  UnitTest: Send OK (No entries to send.)
  FileMetric: Send OK (1 entries)
```

(14)

Resources

API
- Platform API
- Java Development Tool (JDT) API
- API Doc Support (HELP menu on Eclipse)

CVS
- Source code available
- :pserver:anonymous@dev.eclipse.org:/home/eclipse

Technical Articles
- more than 30 very useful articles
- http://eclipse.org/articles/index.html

News Groups
- 23 news groups
- news.eclipse.org
- http://eclipse.org/newsgroups/index.html

Mailing Lists
- 56 mailing lists
- http://eclipse.org/mail/index.html

(15)

How to start on your plug-in

Technical Articles
- http://www.eclipse.org/articles/index.html
  - Your First Plug-in (Revised for 2.0)

Platform API
- (workbench ui, jface ui, swt, etc)
- Help | Help Contents | Platform Plug-in Developer Guide | Reference | API Reference
- <eclipse_home>\plugins\org.eclipse.platform.doc.isv_2.1.0\doc.zip –reference\api\index.html

JDT API
- (compile, test, debug, and edit programs written in the Java programming language)
- Help | Help Contents | JDT Plug-in Developer Guide | Reference | API Reference
- <eclipse_home>\plugins\org.eclipse.jdt.doc.isv_2.1.0\doc.zip –reference\api\index.html

(16)

Anatomy of hackyPerf

Aaron Kagawa
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Talk Outline

What is hackyPerf
Motivation for Performance Analysis
Performance Questions
Design and Implementation
Current Performance Results

(2)

Motivations

How many users can Hackystat support? 10? 50? 100?

We have all these great frameworks for caching but do they really work?

When changes are made to performance sensitive areas how do they affect performance?

What are the performance sensitive areas?

In Fall 2003, we had sudden outbreak of the OutOfMemory Exceptions. What was the problem?

(3)

Why don't we use JMeter

JMeter does load testing BUT,

- Doesn't know anything about Hackystat
- Can't send Sensor Data to server
- Can't integrated with Daily Build (no Ant support)
- Harder to register Hackystat users and Hackystat projects

(4)

Goals

Research Goals
- Assess the scalability of Hackystat
- Assess the thread-safety of Hackystat
- Assess the memory utilization of Hackystat
- Assess the impact of changes to the Hackystat server on performance

hackyPerf Solutions
- Reproducible Results
- Automation
- Emulate different "loads" of use

(5)

What is hackyPerf?

It is not part of the Hackystat Framework

We rolled our own Performance Analysis Tool

- Configurable performance load tests that can model different "usage loads"
- Sends Sensor Data
  - Activity, Coverage, File Metric, JUnit
- Sends Http Requests
- Collects Server Response Time and Determines Correctness
- Generates Reports

(6)

Design and Implementation

Command Line Interface

```
java –jar hackyPerf.jar [loadtest-config.xml]
```

Apache Ant

```
<!-- ================================================= -->
 <target name="perf" depends="prepare" description="Invoke the set
of
     preformance tests.">

   <hackyPerfTest todir="${hackyperf.report.dir}"
       verbose="${config.verbose}">
    <fileset dir="${hackyperf.build.production.dir}">
      <include name="${config.prefix}-*${config.file}*.xml" />
    </fileset>
   </hackyPerfTest>

   <hackyPerfReport todir="${hackyperf.report.dir}">
    <fileset dir="${hackyperf.report.dir}">
      <include name="*.xml" />
    </fileset>
   </hackyPerfReport>
 </target>
```

(7)

```
<LoadTest Label="ActivitySensorDataBaseline"
     Description="Tests sending activity sensor data under baseline
     conditions." Server="@config.server@">
  <ActivitySensorData Label="ActivitySensorDataBaseline"
        TargetRequestsPerMinutePerUser="30"
        TotalTestTimeInSeconds="60"
        SensorDataBatchSize="10"
        SensorDataTimeStampStartTime="2004-01-01T00:00:00"
        SensorDataTimeStampIncrementTime="00:10:00"
        ActivitySensorDataAction="statechange"
        ActivitySensorDataFileNameTemplate="c:\foo.java">
    <User Name="perf-activitysensordatauserbaseline1"
       AutoRegistration="true"/>
  </ActivitySensorData>
</LoadTest>
```

(8)

```
<LoadTest Label="ActivitySensorDataModerate "
     Description="Tests sending activity sensor data under
     moderate conditions." Server="@config.server@">
  <ActivitySensorData Label="ActivitySensorDataModerate"
        TargetRequestsPerMinutePerUser="100"
        TotalTestTimeInSeconds="60"
        SensorDataBatchSize="10"
        SensorDataTimeStampStartTime="2004-01-01T00:00:00"
        SensorDataTimeStampIncrementTime="00:10:00"
        ActivitySensorDataAction="statechange"
        ActivitySensorDataFileNameTemplate="c:\foo.java">
    <User Name="perf-activitysdtmod0" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod1" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod2" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod3" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod4" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod5" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod6" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod7" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod8" AutoRegistration="true"/>
    <User Name="perf-activitysdtmod9" AutoRegistration="true"/>
  </ActivitySensorData>
</LoadTest>
```

(9)

Performance Results

Problems Solved

- StringListCodec
- Server Deadlock
- ConcurrentModificationException

(10)

Performance Results

Sensor Data Results

- No significant differences between the different SDTs
- Duh, more users equals longer response time
- Hackystat server can handle:
  - 100-120 SD requests/minute from one user
  - 30 SD requests/minute from 10 users
  - 12 SD requests/minute from 20 users

(11)

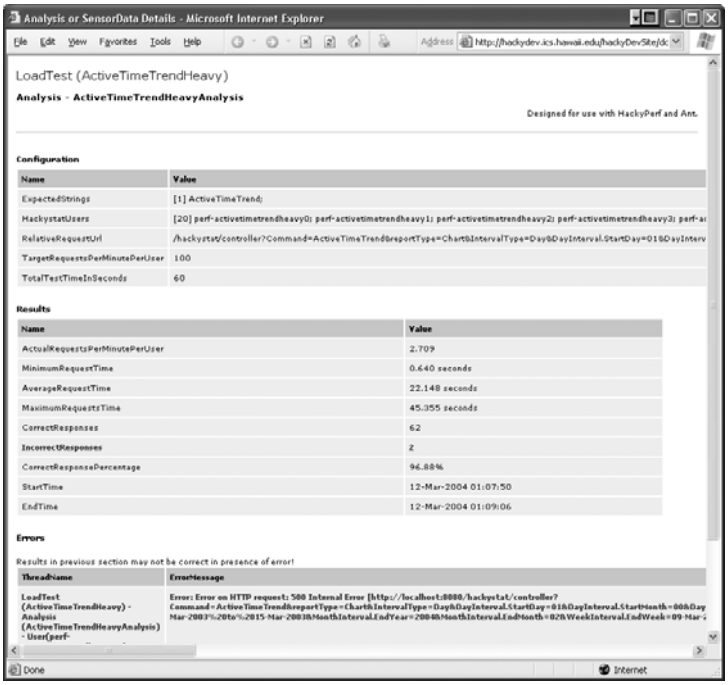
(12)

Performance Results

Analysis Results

- Tomcat has its own limitations
- No problems with DailyDiary (single day)
- Problems with Active Time Trend (multi-day, charts, caching)

(13)



(14)

Future Plans

- Improve Automation so hackyPerf can be executed during the daily build
- Further analyze differences between one user and many users
- Performance of creating Charts
- Project and Course Performance
- Performance Sensor Data Type

(15)

Research Directions: Improving HPC Development

Michael Paulding
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Motivation

Ever wonder how software engineering varies between development on 1 processor to development on 100 processors?

As you could imagine, communication between processors is a critical factor.

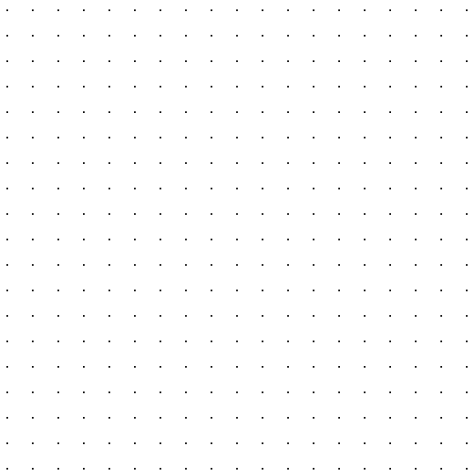
For example, what if a data array you need resides on another machine? Is it worth it to deal with network traffic to get the array or should it be recomputed locally?

These issues as well as many others make it interesting to study the difference between traditional software engineering (SE) and HPC development.

(2)

Motivation (cont)

Imagine having this many nodes at your disposal. Could you manage them effectively? Configure the communication topology? Handle the electricity bill?



(3)

Introduction

Development on a high performance computing system vastly different from traditional software engineering (SE).

Environments for HPC development often lack integration of modern tools.

Philosophy of HPC development is opposite that of traditional SE: optimization as first priority, rather than afterthought.

(4)

Introduction (cont)

HPC development introduces new set of software/developer metrics.

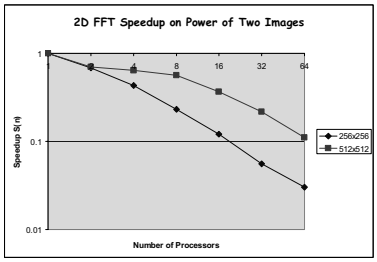
Five universal metrics of HPC development:
1) Speedup 2) Efficiency 3) Redundancy
4) Utilization 5) Quality of Parallelism

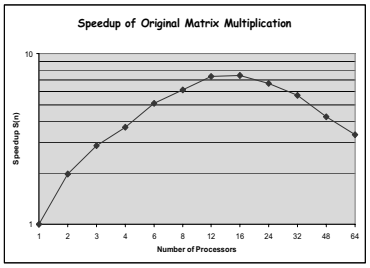
A primary goal of HackyHPC is to automatically collect and analyze metrics related to HPC development to improve software quality and developer productivity.

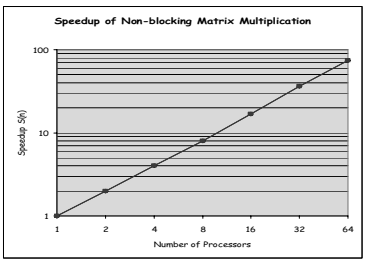
Sun Microsystems, Inc. has provided $50K year 1 grant to realize this goal.

(5)

Speedup Curves – Not always Linear







(6)

Related Work

Pittsburgh Supercomputing Center (PSC)
http://www.psc.edu

- Received $800K grant from IBM to build system similar to Hackystat to measure HPC specific development.
- Researchers at PSC building system essentially from scratch.
- Good resource for collaboration, possibly incorporation of Hackystat system.

(7)

The System (brainstorming)

Create a software package to capture and analyze 5 performance indicators, etc* utilizing the Hackystat system as its backbone.

Extension to Message Passing Interface (MPI) to isolate HPC specific events during compilation and execution.

Tweak existing sensors to record HPC specific events: shared/local mode broadcasting, processor configuration, etc.

(8)

Experimental Evaluation Spring-Summer 2004

Currently implementing software to optimize design of planar truss using sequential and parallel techniques.

Self-reflection process during development to identify and isolate issues specific to HPC.

Using Hackystat system to gather metrics from Emacs tool and command line invocations.

(9)

Data Collected

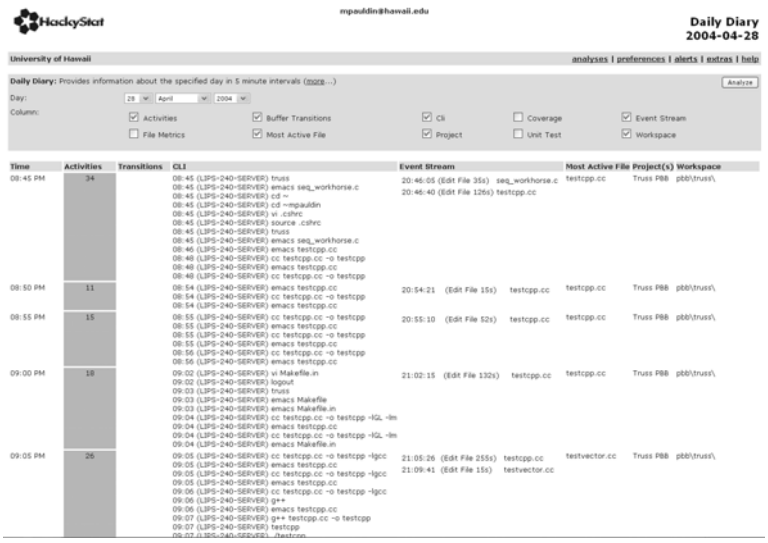
Daily diary of programming events from both Emacs and command line invocations.

Daily journal entry from developer, commenting on issues/insights from the session.

Catalog of defects encountered during development, categorized and indexed.

(10)

Data Collected – example Hackystat diary



(11)

Data Collected – example Developer journal

HPC Research Initiatives – Engineering Log
Collaborative Software Development Laboratory
Sun Microsystems, Inc.

General Information:

Date	Project	Researcher
26-Apr-2004	Truss Optimization	Mike Paulding

Session Information:

Active Time	Start Time	End Time	File(s) Modified
4.5 hr(s)	2:30pm	8:00pm	seq_workhorse.c, force_balance.h, truss.h

Tasks Completed:

In this session, the objective was to continue work on coding the sequential workhorse portion of the optimal truss design problem. The sequential workhorse is defined as the module that computes the mass of a truss and verifies whether the structure is in equilibrium, given a specific topology and geometry of the truss.

Specifically, this session was devoted to the implementation of a force calculator, given a support joint, a load and a maximum of 3 members.

Defects Encountered:

In this session, more issues were uncovered in the translation of problem description to source code. Specifically, it was noticed that the C language does not provide a mechanism for structure creation and initialization, similar to a constructor. Therefore, whenever a new struct is defined, it requires several additional lines to initialize it before

(12)

Contributions

DARPA has announced national objective to boost the power of supercomputers one-thousand fold by 2010.

In order to achieve this, HPC developers must be able to utilize potential of hardware through efficient use of software.

HackyHPC intends to bridge this gap by analyzing how HPC development is done and how it can be done better.

(13)

Timeline

April-May 2004: Working on optimal truss design parallelization. Deliverables include working software, daily journals of developer insights and daily diaries of Hackystat data.

June-July 2004: Completion of truss benchmark. Analysis of data gathered during development to identify issues specific to HPC.

July-August 2004: Internship at Sun Microsystems HPC group. Analysis of Sun's processes and products in relation to personal findings. Refinement of HPC related metrics, sensors, sensor data.

Fall 2004: Development of software package to gather HPC metrics and performance analysis, including extensions to existing Hackystat sensors .

(14)

Test-Driven Development Visualization and Recognition

Hongbing Kou

(1)

Introduction

Stop Light Pattern of TDD

1. Start. (Green light!)
2. Write a test.
3. Try to run the test. *It fails to compile, because the called routine hasn't been written yet.* (Yellow light!)
4. Write a stub for the new routine.
5. Try to run the test. *It fails, because the stub doesn't* do *anything yet.* (Red light!)
6. Write the body of the stubbed routine.
7. Try to run the test. *It passes.* (Green light again!)
8. Start the cycle again.

(2)

Introduction cont'd

Two basic rules of TDD:

1. Write new code only if an automated test has failed
2. Eliminate duplication (Refactoring)

(3)

Introduction Cont'd

Refactoring

1. Start. (Green light.)
2. Apply the Refactoring.
3. Compile and run the test. (Green light again!)

(4)

What's cool of TDD?

Yield high quality code
- Reliable and comprehensive test code as well as application code

Robust Design
- Test forces clear interface design
- Code is better structured and testable

High Flexibility
- Changes can be made with confidence because of unit tests

(5)

Related Work

1. Boby George concluded that both students and professional developers yielded higher quality code
2. Michael Maximilien and Laurie Williams found defect rate of a project developed at IBM was reduced by 50% and TDD developers passed 18% more black box test than non-TDD developer in comparison study
3. Hakan etc. studied TDD cycle pattern with the zipped Hackystat data

(6)

Limitation of TDD Practice

Turns Development of Software Development Upside Down

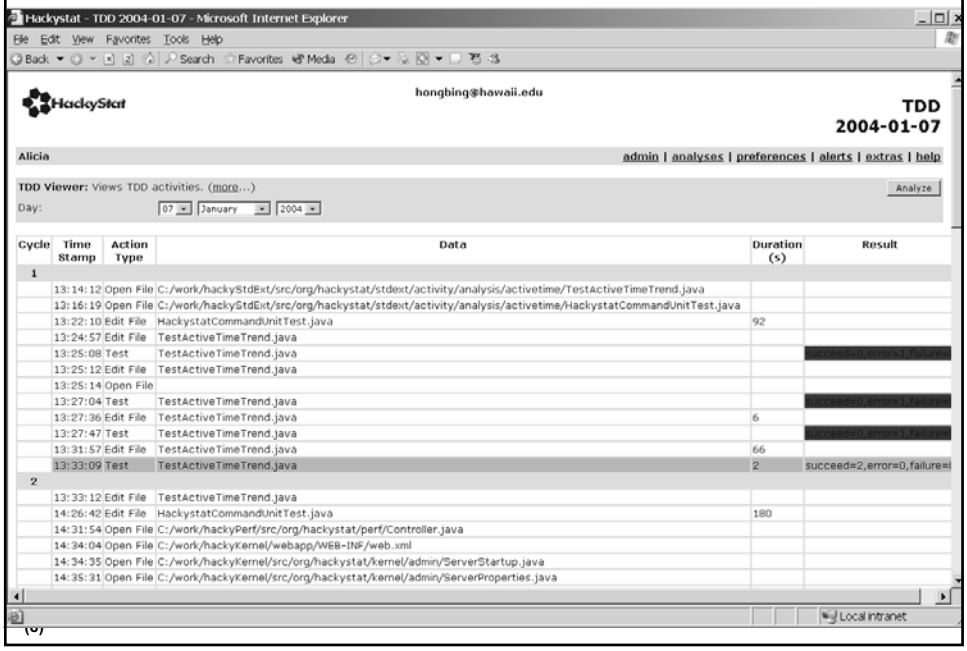
TDD usage depends on discipline to write test first before code, which contradicts with usual software development

Relies on test technique

- xUnit
- Mock complex or external operations

(7)

TDD Viewer



Problems and Solution

Challenges:

- How to distinguish TDD with ad-hoc, and mix of TDD with ad-doc (test last) ?
- How to point out the violation of TDD ?

Solution:

- Assuming TDD process in default and make TDD cycles with activity data.
- Study cycle attribute and the event sequence in the cycle.

(9)

What system can and will do?

1. Visualize development activities
2. Answers question "Do developers do Unit Test?"
3. Visualizes TDD process using activity data and unit test data
4. TDD Pattern identification and pattern study (To be added)

(10)

Thesis Statement

TDD process can be identified with development data collected by Hackystat sensor

Visualization and evaluation on TDD process can improve the discipline of TDD practice

Frequent violation of TDD rules can reduce the effectiveness of development to TDD practitioners

(11)

Evaluation

1. Comparison study between TDD and ad-hoc development in CSDL. Can TDD viewer tell difference between these two?

2. Solicit adoption of TDD viewer and analyses from TDD practitioners

3. Conduct TDD experiment on students' project to see how well they can do TDD with the visualization tool support.

(12)

Data Collected

1. Development activity data collected by Hackystat IDE sensor and unit test data by test sensor includes
   - IDE activities such as Open File, Close File, Edit File, Save File, State Change, Debug(Break Pointer), Buffer Pointer Unit Test, Build Error etc.
   - Eclipse is not event-driven system so it is not possible to collect all activities such as renaming, pull up, push down and so on menu-driven operation.
2. Qualitative data on adoption of TDD because of the visualization tool. (Empirical Study)

(13)

Contribution

1. Provides a measurement and inspection tool for Extreme Programming

2. Helps educating TDD

3. Understands software development process with activity theory.

(14)

Time Line

Jul 10, 2004
- Finish TDD view and analyses

Aug 1, 2004
- Release hackyTDD and ask for review from group and outside TDD practitioners.

Oct 15, 2004
- Enhance the TDD viewer and analyses to make it useful and ready for use

Dec 3, 2004
- Evaluate the data from students (and outside experiment data.)

(15)

Research Directions: Improving software development of MDS

Aaron Kagawa
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Talk Outline

Introduction
Related Work
The hackyMDS System
Thesis Statement
Experiment Evaluation
Contributions
Timeline

(2)

Introduction

- How hard would it be to determine the software quality of this very small system?
- How hard would it be to keep the software quality at a high level?

```
public static void main(String args[]) {
  System.out.println("Hello, World");
}
```

(3)

- How hard would it be to determine the software quality of a Software System with 5 million lines of code?
- How hard would it be to keep quality high?

(4)

All Software Systems Are Not Created Equal

How does a development process for LARGE systems differ from small systems?

What does that say about their Software Quality?

How does Software Quality Programs for LARGE systems differ from small systems?

Do LARGE systems require a different set of Software Metrics than small systems?

(5)

Large System Issues

Is it possible to review every line of code in a multi-million line system?

Is it possible to have 100% coverage for a multi-million line system?

Is it easier to have high software quality in a system with 5 developers than a system with 40 developers?

Building and Testing a LARGE software system takes MUCH longer than a small system. How does this effect development?

(6)

This Thesis In a Nutshell

This thesis isn't a comparison study

Case Study of Hackystat supporting software quality measurement and improvement of VERY LARGE Software System, the Mission Data System being developed at the Jet Propulsion Laboratory

(7)

Related Work

Large Software Projects

- Software Quality Programs/Process Management for Large Software Systems
- Software Metrics used in Large Software Systems

(8)

Mission Data System

A unified architecture for flight, ground, and test systems that enables missions requiring reliable, advanced software

- Build a highly reusable core software system for a wide variety of space mission applications.
- Promote modern, synergistic processes for systems and software engineering
- Establish an improved development life cycle for more reliable mission software
- Reduce development cycle time and cost
- Reduce operation cost with increased autonomy

"Mission Data System Architecture and Implementation Guidelines", George Rinker, March 2002.

(9)

MDS Development Process

I don't care about the implementation details

I will investigate the development process of MDS using the MDS CCC Harvest Build and Configuration Management Tool

hackyMDS uses data extracted from Harvest to analyze the Development Process

Using development process data we can measure software quality

(10)

hackyMDS Hackystat Extension

Sensor Data Types
- **Build**
- **State Change**

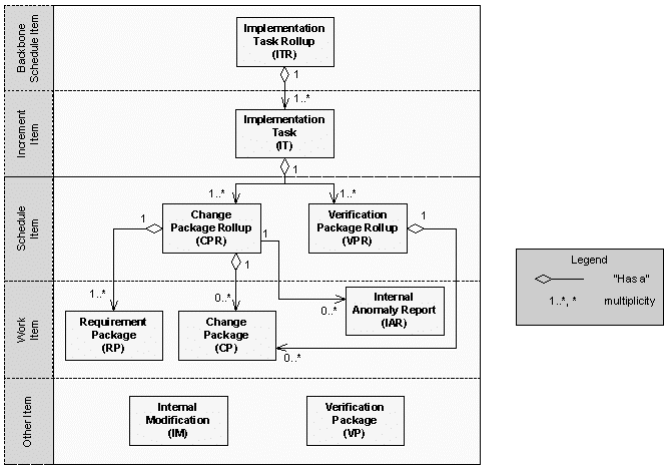
MDS Package Sensor Data

Cache
- **Summary Level**
- **MDS Package Type Level**

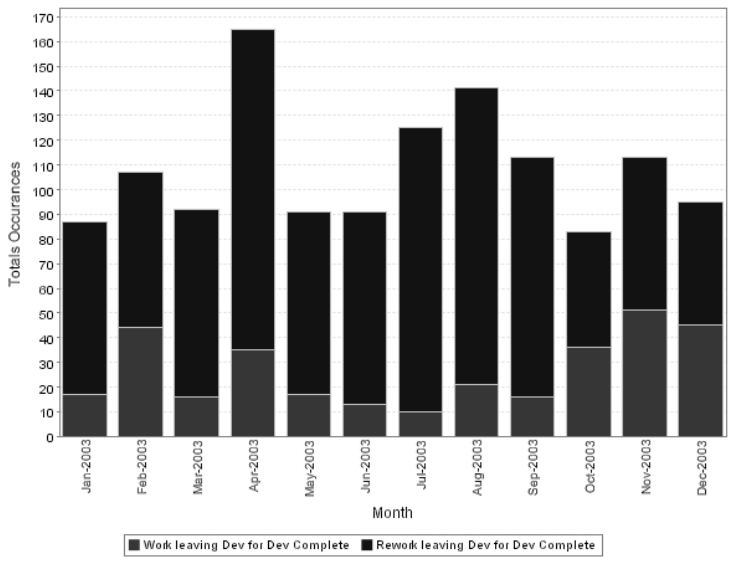
Analyses

(11)

MDS Package Representation

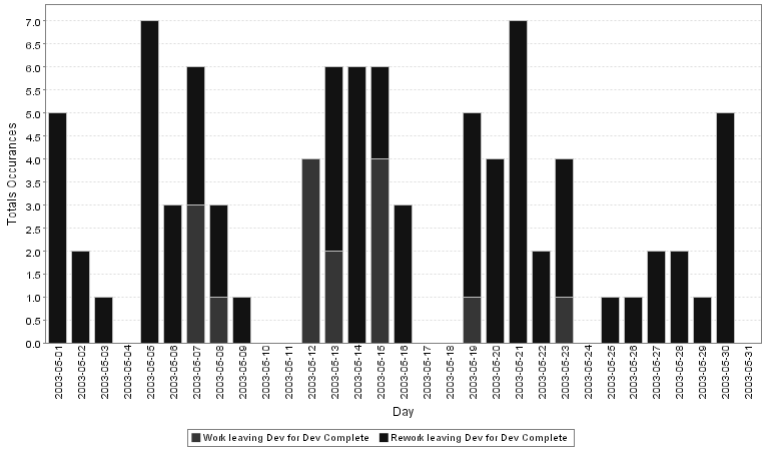


(12)



Work = CPs **Rework = IARs and IMs**

(13)



(14)

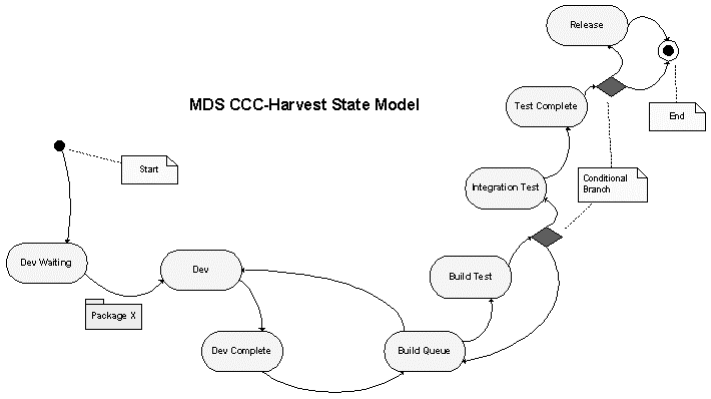
Why is Rework High?

What software development process factors influence such a high level of Rework

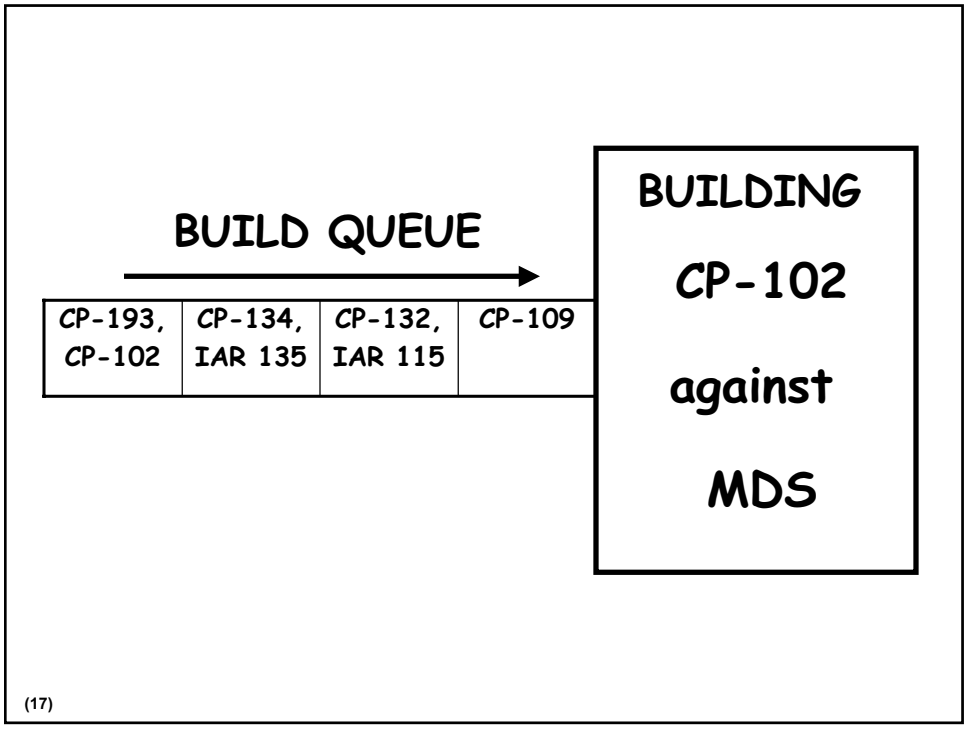
- hackyTelemetry is perfect for investigating this
  - Rework Telemetry Stream
  - CP Age in Dev Telemetry Stream
  - Developers Telemetry Stream

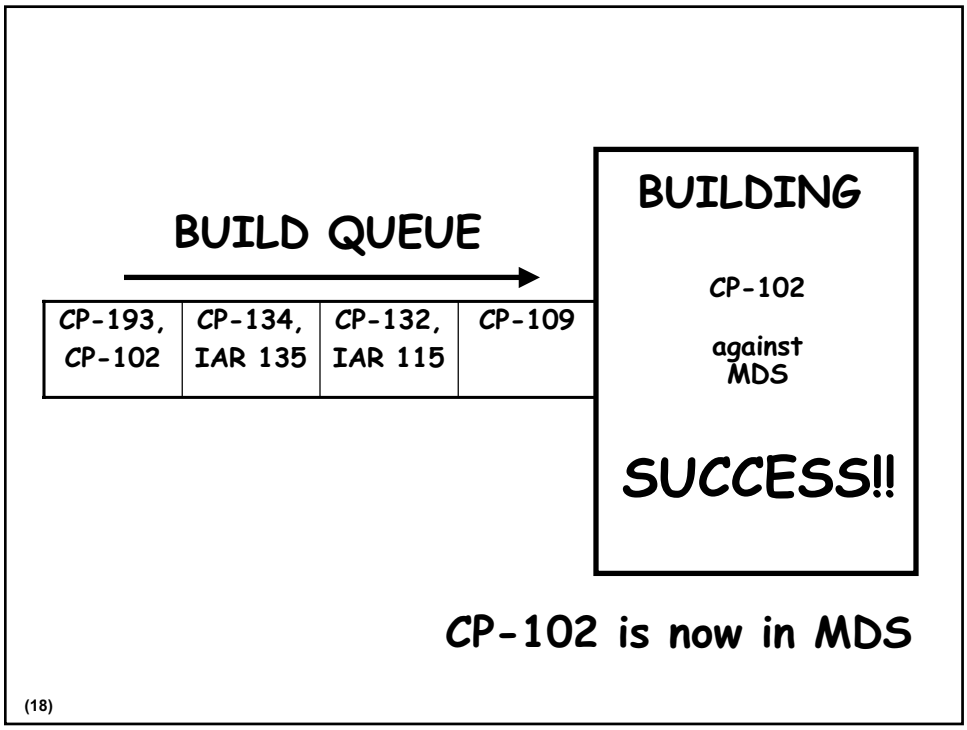
(15)

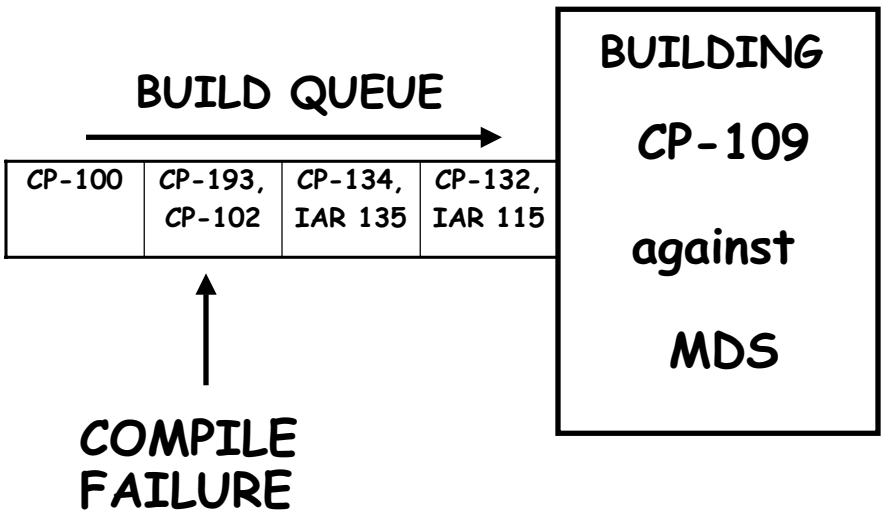
Mission Data System Harvest State Model



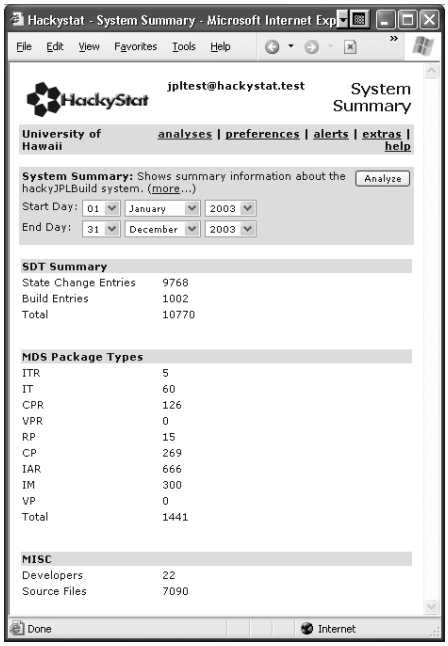
(16)


BUILD QUEUE
CP-193, CP-102
CP-134, IAR 135
CP-132, IAR 115
CP-109
BUILDING
CP-102
against
MDS
(17)

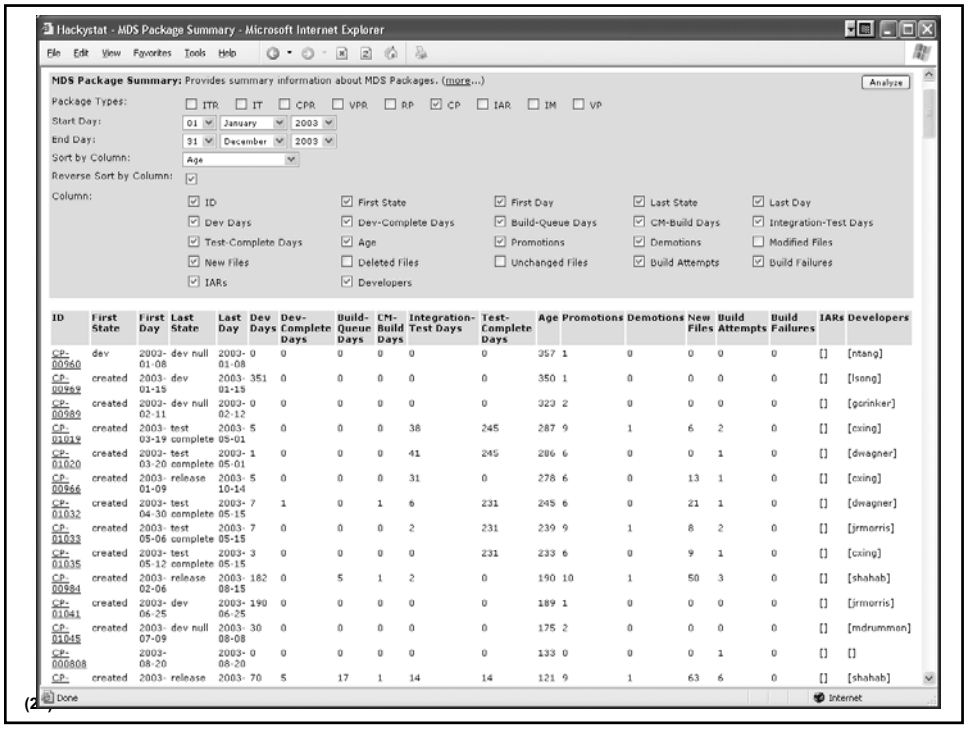

BUILD QUEUE
CP-193, CP-102
CP-134, IAR 135
CP-132, IAR 115
CP-109
BUILDING
CP-102
against MDS
SUCCESS!!
CP-102 is now in MDS
(18)


BUILD QUEUE
CP-100
CP-193, CP-102
CP-134, IAR 135
CP-132, IAR 115
BUILDING
CP-109
against
MDS
COMPILE FAILURE

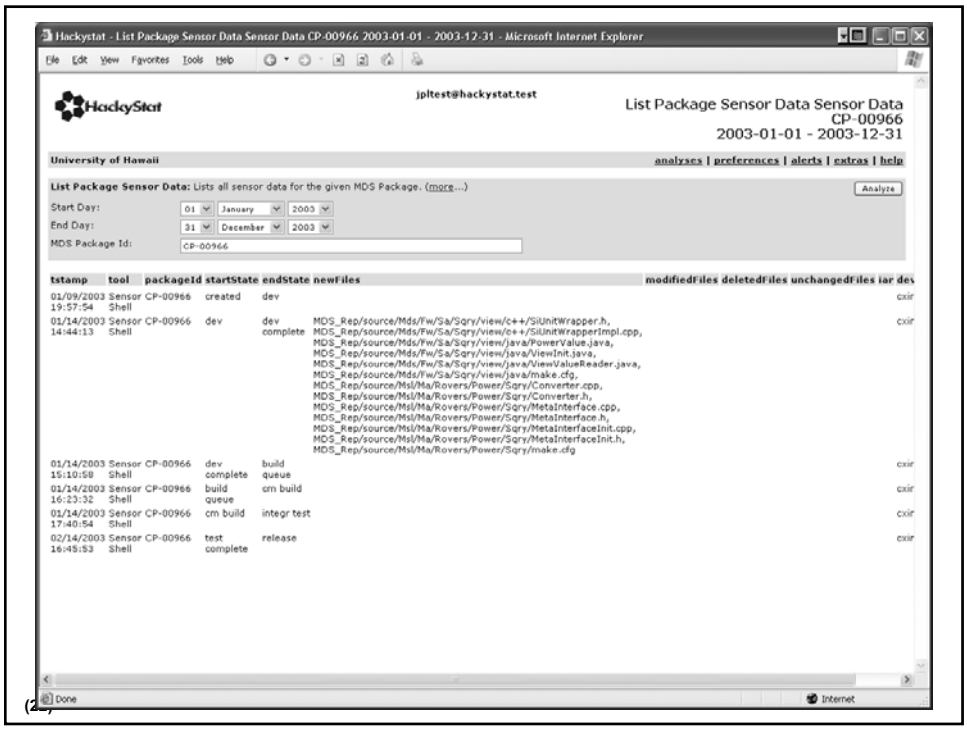
(19)


Hackystat - System Summary - Microsoft Internet Exp
File Edit View Favorites Tools Help
HackyStat
jpltest@hackystat.test
System Summary
University of Hawaii
analyses | preferences | alerts | extras | help
System Summary: Shows summary information about the hackyJPLBuild system. (more...)
Analyze
Start Day: 01 January 2003
End Day: 31 December 2003
SDT Summary
State Change Entries 9768
Build Entries 1002
Total 10770
MDS Package Types
ITR 5
IT 60
CPR 126
VPR 0
RP 15
CP 269
IAR 666
IM 300
VP 0
Total 1441
MISC
Developers 22
Source Files 7090
Done
Internet

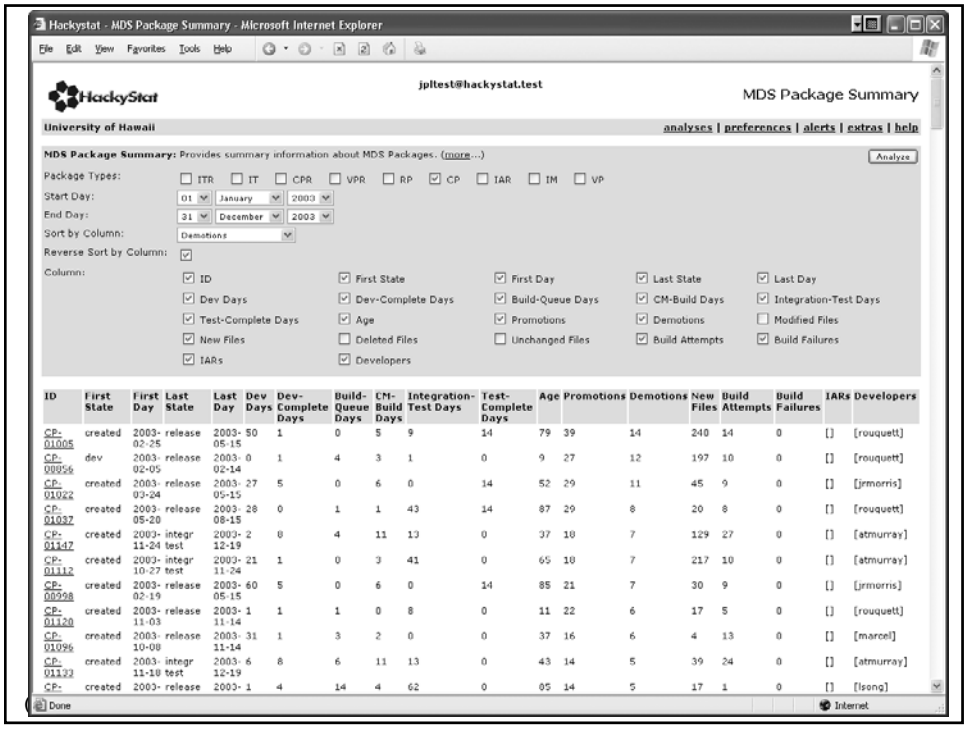
(20)

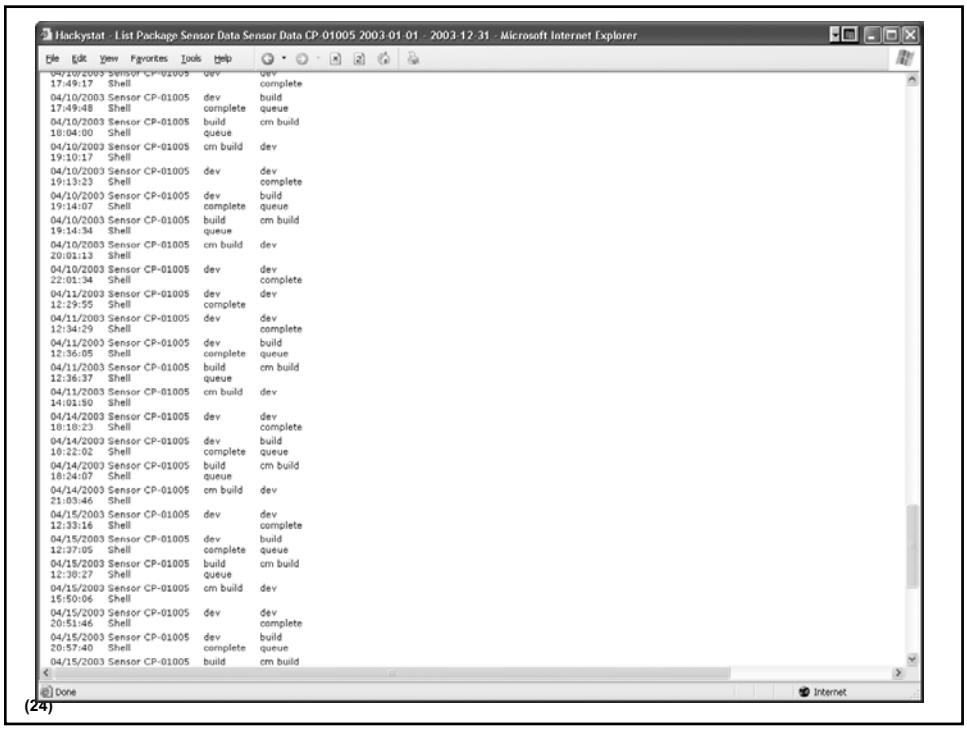

Hackystat - MDS Package Summary - Microsoft Internet Explorer
MDS Package Summary: Provides summary information about MDS Packages. (more...)
Analyze
Package Types: ITR IT CPR VPR RP CP IAR IM VP
Start Day: 01 January 2003
End Day: 31 December 2003
Sort by Column: Age
Reverse Sort by Column:
Column: ID First State First Day Last State Last Day Dev Days Dev-Complete Days Build-Queue Days CM-Build Days Integration-Test Days Test-Complete Days Age Promotions Demotions Modified Files New Files Deleted Files Unchanged Files Build Attempts Build Failures IARs Developers
ID First State First Day Last State Last Day Dev Days Dev-Complete Days Build-Queue Days CM-Build Days Integration-Test Days Test-Complete Days Age Promotions Demotions New Files Build Attempts Build Failures IARs Developers
CP-00960 dev 2003-01-08 dev null 2003-01-08 0 0 0 0 0 0 357 1 0 0 0 0 [] [ntang]
CP-00959 created 2003-01-15 dev 2003-01-15 351 0 0 0 0 0 350 1 0 0 0 0 [] [lsong]
CP-00949 created 2003-02-11 dev null 2003-02-12 0 0 0 0 0 0 323 2 0 0 0 0 [] [gcrinker]
CP-01019 created 2003-03-19 test complete 2003-05-01 5 0 0 0 38 245 287 9 1 6 2 0 [] [cxing]
CP-01020 created 2003-03-20 test complete 2003-05-01 1 0 0 0 41 245 286 6 0 0 1 0 [] [dwagner]
CP-00966 created 2003-01-09 release 2003-10-14 5 0 0 0 31 0 278 6 0 13 1 0 [] [cxing]
CP-01032 created 2003-04-30 test complete 2003-05-15 7 1 0 1 6 231 245 6 0 21 1 0 [] [dwagner]
CP-01033 created 2003-05-06 test complete 2003-05-15 7 0 0 0 2 231 239 9 1 8 2 0 [] [jrmorris]
CP-01035 created 2003-05-12 test complete 2003-05-15 3 0 0 0 0 231 233 6 0 9 1 0 [] [cxing]
CP-00984 created 2003-02-06 release 2003-08-15 182 0 5 1 2 0 190 10 1 50 3 0 [] [shahab]
CP-01041 created 2003-06-25 dev 2003-06-25 190 0 0 0 0 0 189 1 0 0 0 0 [] [jrmorris]
CP-01045 created 2003-07-09 dev null 2003-08-08 30 0 0 0 0 0 175 2 0 0 0 0 [] [mdrummon]
CP-000808 2003-08-20 2003-08-20 0 0 0 0 0 0 133 0 0 0 1 0 [] []
CP- created 2003- release 2003- 70 5 17 1 14 14 121 9 1 63 6 0 [] [shahab]
Done
Internet

(2)

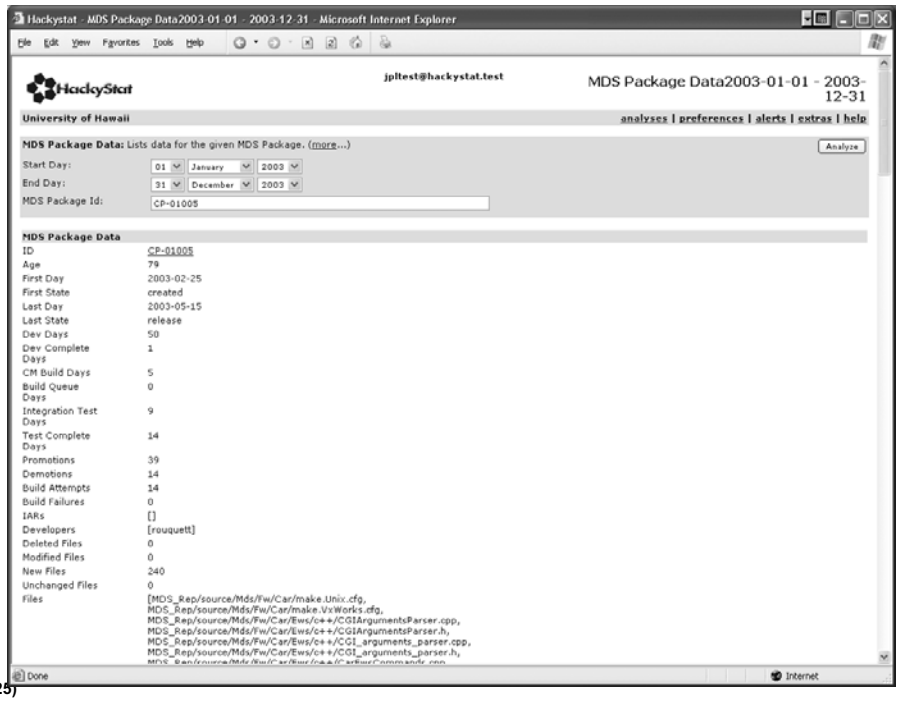

Hackystat - List Package Sensor Data Sensor Data CP-00966 2003-01-01 - 2003-12-31 - Microsoft Internet Explorer
HackyStat
jpltest@hackystat.test
List Package Sensor Data Sensor Data
CP-00966
2003-01-01 - 2003-12-31
University of Hawaii
analyses | preferences | alerts | extras | help
List Package Sensor Data: Lists all sensor data for the given MDS Package. (more...)
Analyze
Start Day: 01 January 2003
End Day: 31 December 2003
MDS Package Id: CP-00966
tstamp tool packageId startState endState newFiles modifiedFiles deletedFiles unchangedFiles iar dev
01/09/2003 19:57:54 Sensor Shell CP-00966 created dev cxir
01/14/2003 14:44:13 Sensor Shell CP-00966 dev dev complete MDS_Rep/source/Mds/Fw/Sa/Sqry/view/c++/SiUnitWrapper.h, MDS_Rep/source/Mds/Fw/Sa/Sqry/view/c++/SiUnitWrapperImpl.cpp, MDS_Rep/source/Mds/Fw/Sa/Sqry/view/java/PowerValue.java, MDS_Rep/source/Mds/Fw/Sa/Sqry/view/java/ViewInit.java, MDS_Rep/source/Mds/Fw/Sa/Sqry/view/java/ViewValueReader.java, MDS_Rep/source/Mds/Fw/Sa/Sqry/view/java/make.cfg, MDS_Rep/source/Msl/Ma/Rovers/Power/Sqry/Converter.cpp, MDS_Rep/source/Msl/Ma/Rovers/Power/Sqry/Converter.h, MDS_Rep/source/Msl/Ma/Rovers/Power/Sqry/MetaInterface.cpp, MDS_Rep/source/Msl/Ma/Rovers/Power/Sqry/MetaInterface.h, MDS_Rep/source/Msl/Ma/Rovers/Power/Sqry/MetaInterfaceInit.cpp, MDS_Rep/source/Msl/Ma/Rovers/Power/Sqry/MetaInterfaceInit.h, MDS_Rep/source/Msl/Ma/Rovers/Power/Sqry/make.cfg
01/14/2003 15:10:58 Sensor Shell CP-00966 dev complete build queue cxir
01/14/2003 16:23:32 Sensor Shell CP-00966 build queue cm build cxir
01/14/2003 17:40:54 Sensor Shell CP-00966 cm build integr test cxir
02/14/2003 16:45:53 Sensor Shell CP-00966 test complete release cxir
Done
Internet

(2)


Hackystat - MDS Package Summary - Microsoft Internet Explorer
File Edit View Favorites Tools Help
HackyStat
jpltest@hackystat.test
MDS Package Summary
University of Hawaii
analyses | preferences | alerts | extras | help
MDS Package Summary: Provides summary information about MDS Packages. (more...)
Analyze
Package Types: ITR IT CPR VPR RP CP IAR IM VP
Start Day: 01 January 2003
End Day: 31 December 2003
Sort by Column: Demotions
Reverse Sort by Column:
Column: ID First State First Day Last State Last Day Dev Days Dev-Complete Days Build-Queue Days CM-Build Days Integration-Test Days Test-Complete Days Age Promotions Demotions Modified Files New Files Deleted Files Unchanged Files Build Attempts Build Failures IARs Developers
ID First State First Day Last State Last Day Dev Days Dev-Complete Days Build-Queue Days CM-Build Days Integration-Test Days Test-Complete Days Age Promotions Demotions New Files Build Attempts Build Failures IARs Developers
CP-01005 created 2003-02-25 release 2003-05-15 50 1 0 5 9 14 79 39 14 240 14 0 [] [rouquett]
CP-00956 dev 2003-02-05 release 2003-02-14 0 1 4 3 1 0 9 27 12 197 10 0 [] [rouquett]
CP-01022 created 2003-03-24 release 2003-05-15 27 5 0 6 0 14 52 29 11 45 9 0 [] [jrmorris]
CP-01037 created 2003-05-20 release 2003-08-15 28 0 1 1 43 14 87 29 8 20 8 0 [] [rouquett]
CP-01147 created 2003-11-24 integr test 2003-12-19 2 0 4 11 13 0 37 10 7 129 27 0 [] [atmurray]
CP-01112 created 2003-10-27 integr test 2003-11-24 21 1 0 3 41 0 65 10 7 217 10 0 [] [atmurray]
CP-00998 created 2003-02-19 release 2003-05-15 60 5 0 6 0 14 85 21 7 30 9 0 [] [jrmorris]
CP-01120 created 2003-11-03 release 2003-11-14 1 1 1 0 8 0 11 22 6 17 5 0 [] [rouquett]
CP-01096 created 2003-10-08 release 2003-11-14 31 1 3 2 0 0 37 16 6 4 13 0 [] [marcel]
CP-01133 created 2003-11-10 integr test 2003-12-19 6 8 6 11 13 0 43 14 5 39 24 0 [] [atmurray]
CP- created 2003- release 2003- 1 4 14 4 62 0 85 14 5 17 1 0 [] [lsong]
Done
Internet


Hackystat - List Package Sensor Data Sensor Data CP-01005 2003-01-01 - 2003-12-31 - Microsoft Internet Explorer
File Edit View Favorites Tools Help
17:49:17 Shell complete
04/10/2003 Sensor CP-01005 dev build
17:49:48 Shell complete queue
04/10/2003 Sensor CP-01005 build cm build
18:04:00 Shell queue
04/10/2003 Sensor CP-01005 cm build dev
19:10:17 Shell
04/10/2003 Sensor CP-01005 dev dev
19:13:23 Shell complete
04/10/2003 Sensor CP-01005 dev build
19:14:07 Shell complete queue
04/10/2003 Sensor CP-01005 build cm build
19:14:34 Shell queue
04/10/2003 Sensor CP-01005 cm build dev
20:01:13 Shell
04/10/2003 Sensor CP-01005 dev dev
22:01:34 Shell complete
04/11/2003 Sensor CP-01005 dev dev
12:29:55 Shell complete
04/11/2003 Sensor CP-01005 dev dev
12:34:29 Shell complete
04/11/2003 Sensor CP-01005 dev build
12:36:05 Shell complete queue
04/11/2003 Sensor CP-01005 build cm build
12:36:37 Shell queue
04/11/2003 Sensor CP-01005 cm build dev
14:01:50 Shell
04/14/2003 Sensor CP-01005 dev dev
18:18:23 Shell complete
04/14/2003 Sensor CP-01005 dev build
18:22:02 Shell complete queue
04/14/2003 Sensor CP-01005 build cm build
18:24:07 Shell queue
04/14/2003 Sensor CP-01005 cm build dev
21:03:46 Shell
04/15/2003 Sensor CP-01005 dev dev
12:33:16 Shell complete
04/15/2003 Sensor CP-01005 dev build
12:37:05 Shell complete queue
04/15/2003 Sensor CP-01005 build cm build
12:38:27 Shell queue
04/15/2003 Sensor CP-01005 cm build dev
15:50:06 Shell
04/15/2003 Sensor CP-01005 dev dev
20:51:46 Shell complete
04/15/2003 Sensor CP-01005 dev build
20:57:40 Shell complete queue
04/15/2003 Sensor CP-01005 build cm build
Done
Internet

(24)



(25)

Thesis Statement

hackyMDS can support High Software Quality in MDS

- **hackyMDS accurately represents the development process of MDS and will improve the developers' and managers' understanding of MDS**
- **hackyMDS can identify "factors" that contribute to "problems" (ie build failures)**
- **hackyMDS can use these "factors" to predict and prevent "problems"**

(26)

Experiment Evaluation

hackyMDS accurately represents the development process of MDS and will improve the developers' and managers' understanding of MDS
- Verify and Validate hackyMDS's view of the world
- Questionnaire

hackyMDS can identify "factors" that contribute to "problems" (ie build failures)
- Using hackyTelemetry to identify threshold values for factors influencing "problems"
- Questionnaire + Statistical Results

hackyMDS can use these "factors" to predict and prevent "problems"
- Questionnaire + Development Process Changes

(27)

Contributions

A set of Software (or Development) Metrics that may or may not help determine the Software Quality of a Large System

(28)

Timeline

May: refactor hackyJPLBuild to hackyMDS

June – August: Internship at JPL
- Install hackyMDS: configure Hackystat system to MDS needs.
- Develop automated data sending mechanism
- Gain insights into MDS development
- Develop Telemetry Analyses

Fall 2004: Write Proposal and Conduct Experiment

Spring 2005: Evaluate Results and Write Thesis

Summer 2005: Submit Thesis to Committee

(29)

Research directions: Improving Hackystat Evaluation Quality

Melissa Rota
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Overview

- Introduction: the problem with current usability evaluation methods
- The benefits of quantitative usage data
- Relationships between quantitative and qualitative data
- Analysis Invocation Logger
- Current analysis for invocation log data
- Future analyses and improvements
- Contributions

(2)

Introduction

We want students to understand the process and products of software development

Hackystat-UH provides students with analyses related to process and products

Hackystat analyses need to be usable and useful to students

(3)

Previous Experience with Hackystat-UH

Software engineering students installed Hackystat sensors during the Fall 2003 semester

Students were provided with the following analyses:
- Project member active time
- Project member file active time
- Course Project

Students evaluated their experiences with Hackystat via a questionnaire
- Installation and configuration
- Overhead of use
- Usability and utility
- Future use

(4)

The Problem

Questionnaire only provides qualitative data

Are students running the analyses? If so, how frequently?

A student may say that an analysis is highly useful yet never run that analysis

A student may say that an analysis is not useful but run that analysis frequently

(5)

The Solution

Gather quantitative data about actual Hackystat usage

- Log all analysis invocation activity

Provide administrator-level analyses for quantitative log data

Thesis:
A mixture of quantitative and qualitative data regarding the usability of Hackystat will provide better insight into how the usefulness of the analyses can be improved.

(6)

The Benefits of Quantitative Data

Quantitative usage data can be used to answer these questions:

- Are students running analyses?
- Are there any observable trends in the frequency of analysis invocations?
- What is the average level of analysis invocations among students? What are the maximum and minimum values of invocations for a single user?
- Which analysis is invoked the most? Which analysis is invoked the least?
- Are there any relationships between active time and analysis invocation frequency?
- Are there any relationships between project data and analysis invocation frequency?

(7)

Relationships Between Quantitative and Qualitative Data

Usability level and usage

- Are analyses with high usability ratings invoked frequently?

Utility level and usage

- Are analyses with high utility ratings invoked frequently? Are there any analyses with high utility ratings that are rarely invoked, or analyses with low utility ratings that are invoked frequently?

Team products and usage

- Do teams with "better" products (i.e. higher test coverage) invoke analyses more frequently?

Inconsistencies between perceived usage and actual usage

(8)

Analysis Invocation Logger

Gathers metadata about how Hackystat is being used

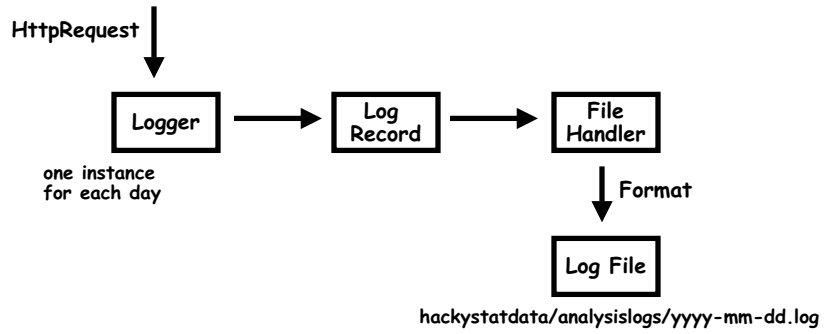
Information about each request is logged in a text file

- Timestamp
- User key
- Analysis name
- Parameters

(9)

Analysis Invocation Logger

Uses Java Logging API



(10)

Analysis Invocation Logger

Log file

```
1075862477903 Command=ListAnalysisLogData&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interv
1075865327110 Command=Login&Key=aYBUBmVf5SEu
1075865332728 Key=aYBUBmVf5SEu&Page=admin
1075865342352 Command=AnalysisLogSummary&DisplayDoc=more&Key=aYBUBmVf5SEu
1075865347920 Command=AnalysisLogSummary&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interva
1075865354319 Key=aYBUBmVf5SEu&Page=admin
1075865379225 Command=ListAnalysisLogData&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interv
1075884661522 Command=Login&Key=aYBUBmVf5SEu
1075884669514 Key=aYBUBmVf5SEu&Page=admin
1075884675993 Command=AnalysisLogSummary&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interva
1075884706487 Command=AnalysisLogSummary&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interva
1075884912123 Command=Login&Key=aYBUBmVf5SEu
1075884916259 Key=aYBUBmVf5SEu&Page=admin
1075884924330 Command=AnalysisLogSummary&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interva
1075885544372 Command=Login&Key=aYBUBmVf5SEu
1075885548327 Key=aYBUBmVf5SEu&Page=admin
1075885552914 Command=ListAnalysisLogData&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interv
1075885571941 Command=ListAnalysisLogData&DayInterval.EndMonth=01&IntervalType=Day&MonthInterval.Star
1075885571951 Key=aYBUBmVf5SEu&Page=admin
1075885583618 Key=aYBUBmVf5SEu&Page=admin
1075885588686 Command=AnalysisLogSummary&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interva
1075885618599 Command=Login&Key=testdataset
1075885629745 Command=ActiveTimeTrend&DayInterval.EndMonth=02&MonthInterval.StartYear=2000&IntervalTy
1075885676962 Command=Login&Key=aYBUBmVf5SEu
1075885679967 Key=aYBUBmVf5SEu&Page=admin
1075885684113 Command=AnalysisLogSummary&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interva
1075885699915 Command=AnalysisLogSummary&DayInterval.EndMonth=01&MonthInterval.StartYear=2000&Interva
```

Timestamp **Key-value pairs**

(11)

Analysis Invocation Logger

Hackystat based on MVC design pattern

- All analysis invocation requests pass through controller

Modified controller:

```
//Log the analysis invocation if command logging is enabled.
if (serverProperties.isAnalysisLoggingEnabled()) {
  AnalysisLogger.getInstance().log(request, Day.getInstance());
}
```

Enable/disable in hackystat.properties

(12)

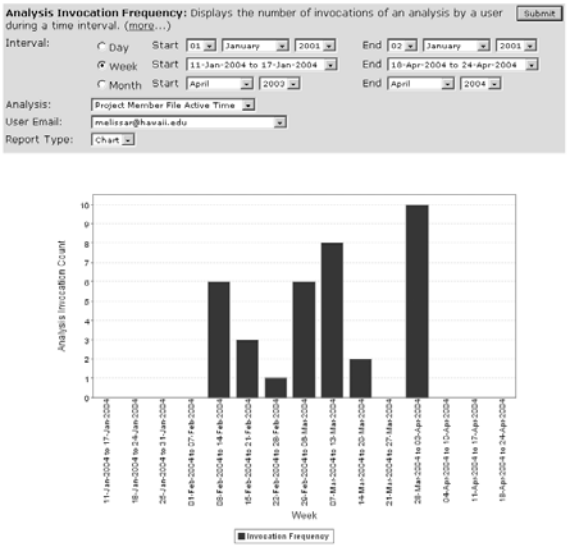
Current Analysis for Analysis Invocation Log Data

Analysis Invocation Frequency

- Produces a histogram of the number of invocations during the given interval by the selected user for the specified analysis
- Determines how often a given analysis is used by one or more users
- Determines how often a given user invokes one or more Hackystat analyses

(13)

Analysis Invocation Frequency



(14)

Future Analyses and Improvements

Analysis Invocation Summary
- Charts the invocation levels for each user and sums the values for each user for each interval

Analysis Invocation Telemetry
- Charts invocation frequency with other metrics like active time and test coverage

Expand email selector
- Course project selector
- Select individual emails or projects

Line chart representation for analysis invocation frequency

(15)

Contributions

Instrumentation for collecting data about real Hackystat usage

Quantitative data for improving Hackystat analyses

Experimental evidence that a mixture of quantitative and qualitative data can be used to improve usability evaluation

(16)

Thank you!

Any Questions?

(17)

Research Directions: Improving software review

Takuya Yamashita
Information & Computer Sciences
University of Hawaii at Manoa

(1)

Overview of the Talk

Introduction of Jupiter
Basic User Scenarios
Experimental Research Questions
Jupiter Sensor for Hackystat
Proposed Analyses

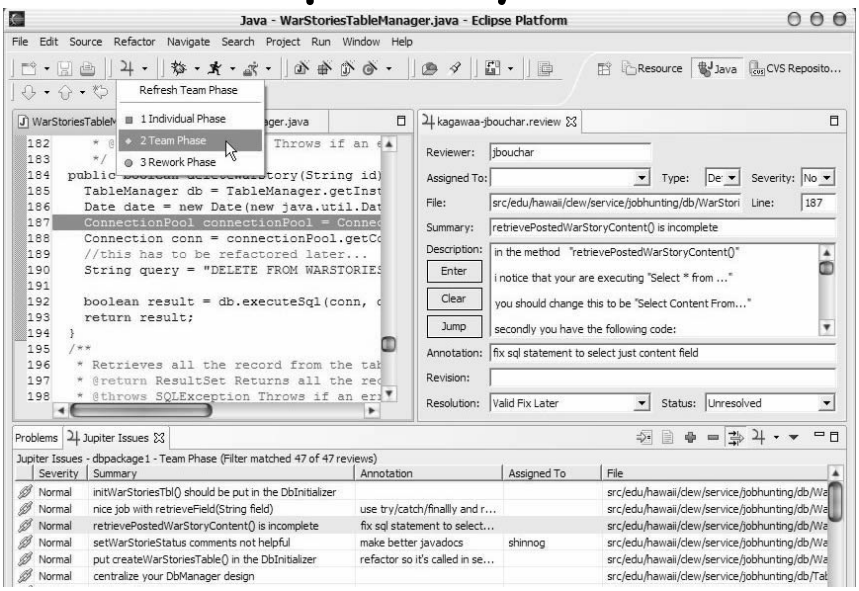
(2)

Jupiter (code review plug-in)

1. Eclipse plug-in (http://eclipse.org)
2. Open source – GNU license
3. Free – free of charge
4. Cross platform – Windows, Linux, MacOSX
5. XML data storage – XML based review file
6. Sorting and Filtering – sort and filter by categories.
7. File Integration – jump back and forth between review comments and source code

(3)

Jupiter System



(4)

Four Review Phase

1. Configuration – to configures review ID, reviewers, and category items.
2. Individual Review – to prepare individual review fast and find as much bugs as possible with in a time limitation.
3. Team Review – to review as much all brought-up review comments as possible and validate them with in a time limitation
4. Revision – to fix the problems efficiently

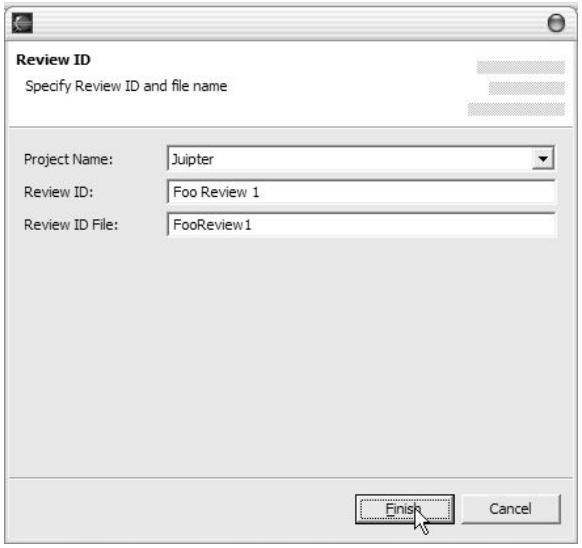
(5)

Configuration Phase

1. Specify Review ID –
   Distinguishes between review sessions
   Should be unique
   Used for the rest 3 phases (individual, team, and rework)
2. Specify Reviewers –
   Used for the assignedTo field
3. Specify Category Item – not implemented yet, supports this in the future
   Used for category elements -
   (e.g. ICS413 has Major, Normal, Minor in Severity. ICS414 has Critical, Major, Normal, Minor, Trivial)

(6)

Configuration Phase



(7)

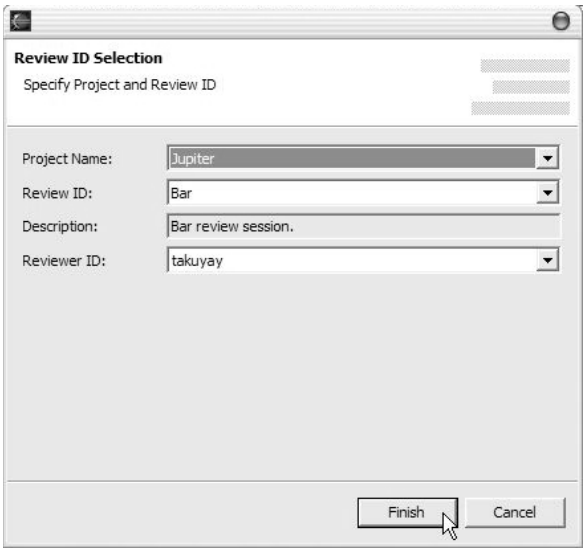
Individual Phase

1. Find an issue (e.g. bug) in a code
2. Select Jupiter menu - right click on it, then select "Add Jupiter Issue"
3. Fill out information for the issue –
   Type? – issue type (e.g. Defect)
   Severity? – importance (e.g. Critical)
   Summary? – summary of the issue
   Description? – detail description
4. Add the issue entry – click "Enter" button to add the issue to Jupiter view table.

** Don't forget to commit your review file!

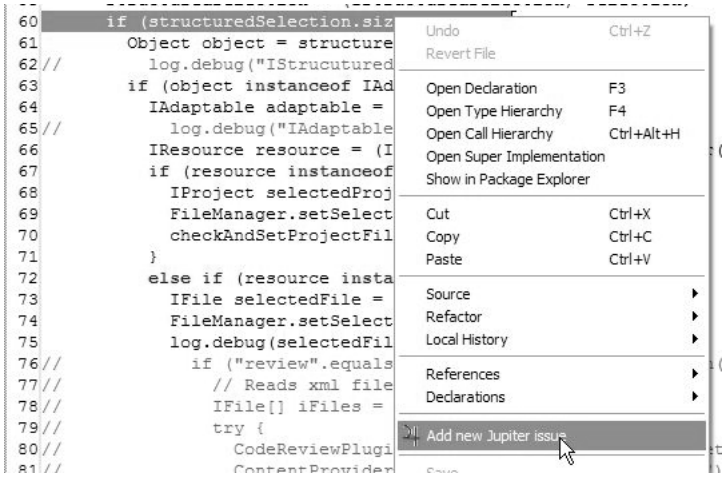
(8)

Individual Phase cont.



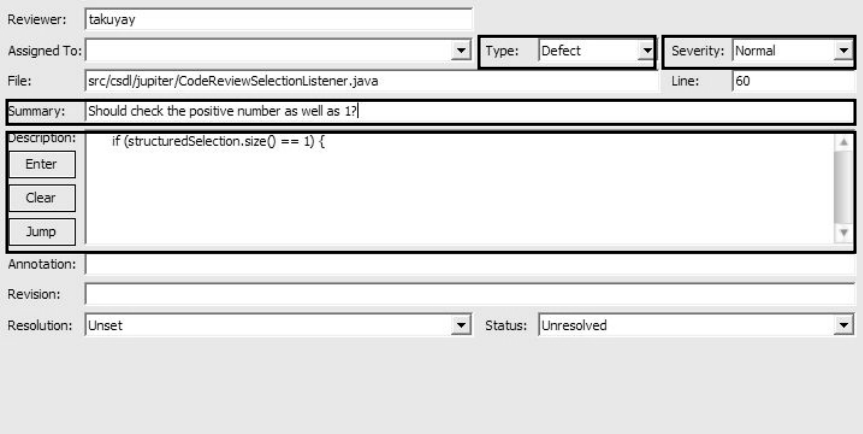
(9)

Individual Phase cont.



(10)

Individual Phase cont.



(11)

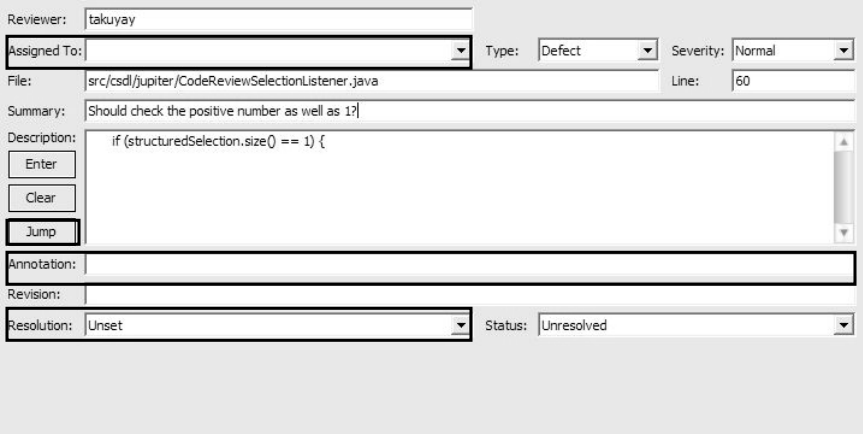
Team Review Phase

1. Set filter and sort – (e.g. filter Resolution-unset, sort by Severity)
2. Review an issue – single click on the issue
3. Jump to the target source – double click on the issue or click "Jump" button on the editor
4. Fill out information for the issue –
   Resolution? – validity of the issue (e.g.Valid-Needsfixing)
   Assign to? – the respondent to fix the issue
   Annotation? – as necessary
5. Enter the modified issue – click "Enter" button to update the issue

** Don't forget to commit reviewed files!

(12)

Team Phase cont.



(13)

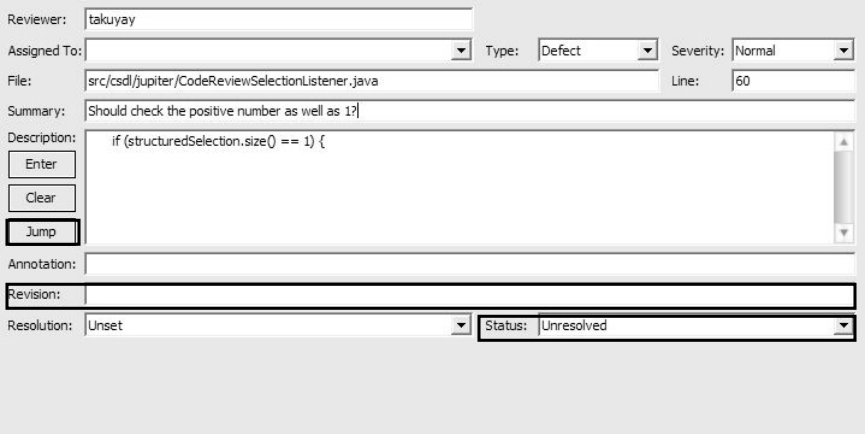
Rework Phase

1. Filter and sorting – (e.g. filter AssignedTo, Status-unresolved, Type-default, Resolution-valid-needsfixing)
2. Review an issue – single click on the issue
3. Jump to the target source – double click on the issue or click "Jump" button on the editor
4. Fix the issue
5. Fill out information for the issue –
   Revision? – comments or memo of the revision
   Status? – status of the issue (e.g. resolved)
6. Enter the modified issue – click "Enter" button to update the issue

** Don't forget to commit revision review file!

(14)

Rework Phase cont.



(15)

Research Questions

General:
- Does automated Jupiter system improve the development process?

Assumptions:
- Improving review process contributes to improving development process.
- High review coverage reduces defects and frequency of testing so as to improve development process.

Specific:
- Does Jupiter is useful enough to improve the review process?
  - Qualitative data (questionnaire)
- Does qualitative data (questionnaire) imply the improvement of the review process?
  - Quantitative data (invocation sensor)

(16)

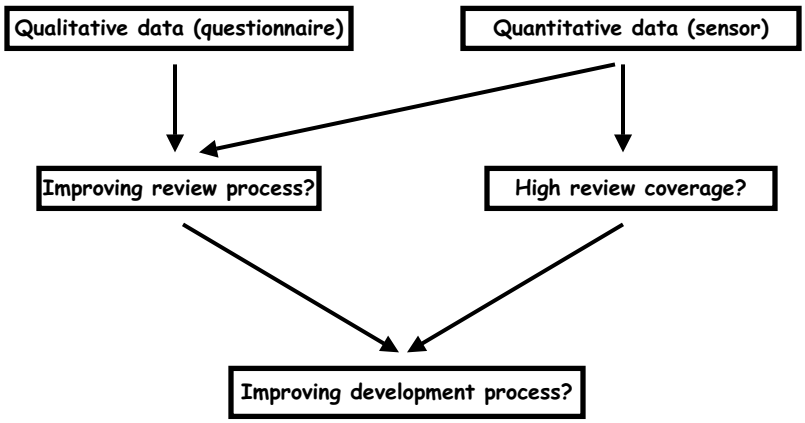
Research Questions cont.

Specific:

- Does Jupiter increase review coverage?
  - Quantitative data (sensor)

(17)

Research Questions cont.



(18)

Jupiter sensor for Hackystat

Goal:

- Understand the review process behavior and review coverage.

Approach:

- Collect review process data
  - Command invocation
- Collect review data
  - Review classes
  - Review time
  - Reviewers

(19)

Proposed Analysis

Time (x) – Review Coverage (y) graph

- Measures review coverage in a system over the time
  - High coverage improves development process

Review session (x) – Time (y) graph

- Measure time during a review session
  - Less time spent for a review session improves review process, as the results, it improves development process too.

(20)

Proposed Analysis cont.

Time (x) - The number of a defect type (y) graph

- Measure the number of a defect over the time
  - More qualified defect type found removes the number of tests for the potential critical defect. Thus it improves development process.

(21)

Research Direction

Software Telemetry

Qin Zhang
Collaborative Software Development Laboratory
Communication & Information Sciences
University of Hawaii, Manoa

May 2004

(1)

Introduction
What is Software Telemetry?

Software development data
- Effort (ActiveTime, etc.)
- File Metrics (size, complexity, etc.)
- Quality (test coverage, build failure, etc.)
- …

Telemetry
- Software development data collected are time-series data in nature.
- "Telemetry" refers to those time-series data.

(2)

Why is Software Telemetry useful?
Software project management using traditional approach

Traditional software planning requires historical software development database, which
- Most companies don't have one
- May be expensive and time consuming to build

Traditional software planning is based on past project data, which
- May not be applicable in the current project

(3)

Why is Software Telemetry useful?
Software project management using telemetry data

Telemetry Data
- Extremely low cost to collect.
- Always up-to-date.
- Project management is based on the data from the current project.

At least, it's good complement to traditional software management techniques.

(4)

My Thesis Topic

Goal:

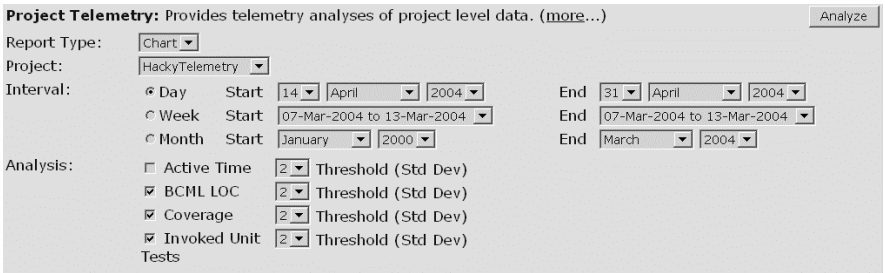
Investigate the utility and applicability of telemetry data in the context of software development management.

Tasks:

- **Find out uniform way to generate telemetry streams from Hackystat sensor data.**
- **Find out how telemetry streams can be useful in software project management.**

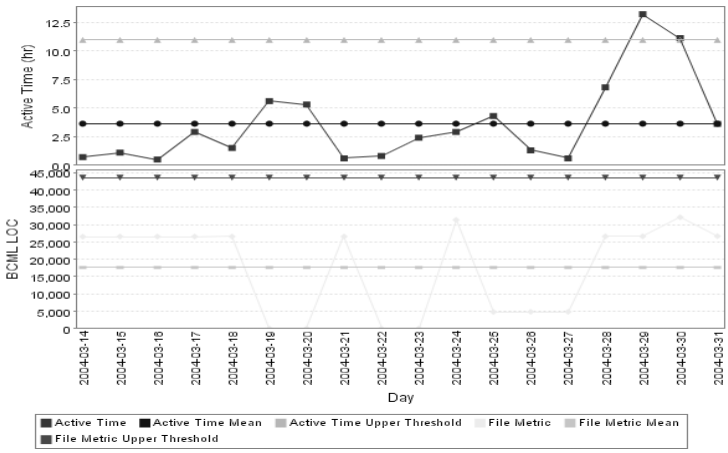
(5)

Telemetry Stream in Current Implementation



(6)

Telemetry Stream in Current Implementation



(7)

Telemetry Stream in Current Implementation

The problem

- Telemetry streams are hard-wired:
  - We have ActiveTime, LOC, Coverage, TestInvocation streams.
  - We have other Sensor Data (e.g. Commit, CLI) but no telemetry streams.
  - Other unknown types of sensor data will be added to Hackystat in the future.
- Not flexible for research need:
  - What if I am only interested in LOC of my test cases?

(8)

Current Research
Flexible Telemetry Stream Generation

Requirements:

Need a flexible and uniform way to generate different telemetry streams from Hackystat sensor data.

- Adding new types of Hackystat sensor data should not require change in Telemetry stream generation module.
- There should be enough flexibility to allow the generation different kinds of stream from the same type of Hackystat sensor data (e.g. LOC of testing code, LOC of java code, LOC of all files)

(9)

Current Research
Flexible Telemetry Stream Generation

Solution: HackyTelemetry Query Language

HackyTelemetryQL → Reduction (, Reduction)*

Reduction:

- mapping between telemetry stream and the underlying Hackystat sensor data.

(10)

Current Research
Flexible Telemetry Stream Generation

Reduction by Example:

```
reduction
➔"CumulativeChurn"
  <Commit(LinesAdded, **/*.java, cumulative) +
  Commit(LinesDeleted, **/*.java, cumulative)>
```

- **"Commit" is sensor data aggregator, there would be one sensor data aggregator for each type of sensor data.**
- **Each sensor data aggregator can take optional arguments.**
- **The reduction allows simple mathematical computation (ADD, SUB, MUL, DIV, NEG)**

Note: We sometimes call "Sensor data aggregator" "reducer".

(11)

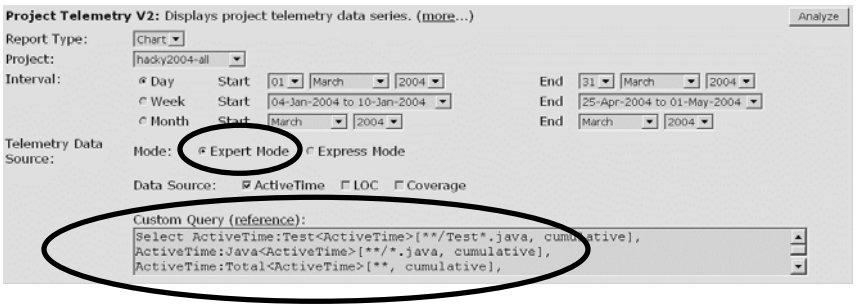
Telemetry QL Prototype

Note:

1. **This prototype was built before we began to think seriously about the telemetry query language.**
2. **The main purpose is to demonstrate the concept of telemetry query language is a feasible concept.**
3. **There is gap between what we want and what the implementation does.**

(12)

Telemetry QL Prototype



(13)

Telemetry QL Prototype

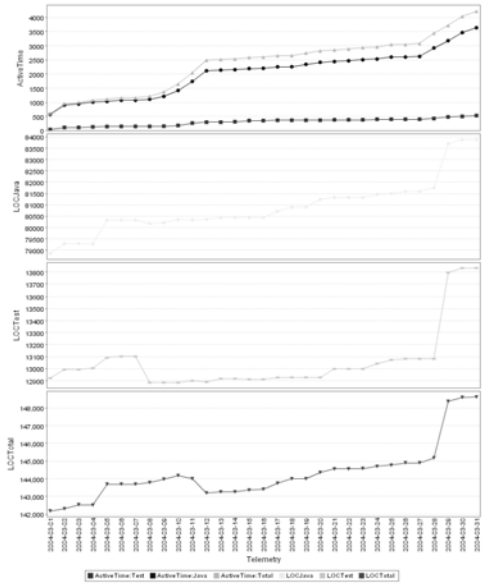
Custom Query:

```
ActiveTime:Test<ActiveTime>[**/Test*.java, cumulative],
ActiveTime:Java<ActiveTime>[**/*.java, cumulative],
ActiveTime:Total<ActiveTime>[**, cumulative],
LOCTest<TotalLines2>[**/Test*.java],
LOCJava<TotalLines2>[**/*.java],
LOCTotal<TotalLines2>
```

Note: Prototype implementation differs slightly from the spec.

(14)

Telemetry QL Prototype



(15)

The Research – Future Direction

1. Finish Telemetry QL implementation.

2. Research how telemetry stream can be useful in software project management.
   - How to apply SPC (statistical process control) theory in software engineering?
   - How to apply statistical control bounds?
   - How to find useful streams (DM might be useful)?

(16)

Thank you!

(17)

Commercialization and Technology Transfer of Hackystat

Philip Johnson
Collaborative Software Development Laboratory
Information and Computer Sciences
University of Hawaii

(1)

Current status

Hackystat is a maturing framework for automated metrics collection and analysis.

Apparently unique combination of:
- Current and emergent functionality
- Development process sophistication
- R&D funding
- Velocity and quality of ongoing development
- Open source, transparent development

(2)

So what's the problem?

CSDL has:
- Finite resources
- Research focus
- Limited capability for technology transfer

Substantial technology transfer of Hackystat will require additional organizational entities that focus on:
- Applications to specific organizational needs.
- Tech transfer and adoption
- Correct interpretation of metrics
- Training and documentation

(3)

Potential markets

Traditional software development companies:
- They need lower-cost approaches to metrics collection and analysis.

New "outsourcing" software development:
- They need a way to gain visibility into development processes occurring in Russia or India.

(4)

A structure

UH/CSDL:

- Continues to develop and publish open source version of technology.
- Performs research and evaluation of technology.

Commercial organization(s):

- Provides enhancements, extensions, maintenance, and services to commercial entities.

Open source lowers risk to buyers, makes product more attractive.

(5)