Hackystat Documentation

Hackystat Quality Engineering Log

Aaron Kagawa
Collaborative Software Development Laboratory
University of Hawaii

$Id: README.html,v 1.1.1.1 2004/09/09 02:39:46 kagawaa Exp $

Overview

This document is an engineering log, which provides detailed information about the inspections recommended using hackyQuality analyses. For more details, please see my thesis proposal.

Engineering Log

This table represents the decisions that we made to recommend an inspection and the results of that inspection.

Inspection Number: 1	
Date:	9.21.04
hackystat-dev-l message:	[HACKYSTAT-DEV-L:82] Review Request 9/21/04
Module and Review Id:	hackyStdExt - JavaMap
Package:	org.hackystat.stdext.workspace.mava.javamap
Stats:	Quality Analysis Info: * active time: 0.5h * last active time: July 27, 2004 * number of contributers (active time): 2 (hongbing and johnson) * expert: hongbing (0.33h) * review: 0 * last review: N/A * file metric: loc=196, classes=1, methods=12 * test filemetric: classes=0, methods=0 * unit test: 0

	* coverage: 90% Other Info * Code initially created on May 7, 2003 - fairly old code * 7 other classes has least one reference to JavaClassWorkspaceMapManager
Quality Level:	Low
Discussion:	This package was not ranked with the lowest possible quality level. Yet, this package seemed like a very good package to pick because of the zero unit tests and 90% coverage. This value indicated that although we have no unit tests a lot of code must use this package for some sort of basic processing. This is evident by the 7 references to the JavaClassWorkspaceMapManager. The references metric would be a great metric to have. Currently, our static analysis tool LOCC does not collect this sort of information (and it is not known at this time whether this would be hard to do or not). In addition, the package has had a few developers contributing to it, therefore one would think that is a problem. Another issue is that the code is fairly old code, and because of the substantial changes to the project caches one would think that there should be some changes to the code. The another critical factor here is that the code has never been reviewed. Yet, it is so vital to the way hackystat works. One would also think that because it is an underlying data structure that is used by many analyses that this makes it a good canidate for inspection. The most critical factor for its selection for inspection was my own subjective feeling about the code. I can't seem to put into words why i thought this was a good package for inspection but, something made me feel that way. Perhaps it was because I knew that this code provided critical functionality to the hackystat project yet for some reason it was so low my quality determination.
Number of Issues:	37 review issues: - critical: 3 - major: 7

	- normal: 12 - minor: 7 - trivial: 5 - unset: 3
Retrospective:	This package was an excellent selection for inspection. Most of the issues that were generated where of upper severity (normal and above). The issues that were examined in the team phase proved that this was a good package to inspect. However, we didn't find any mission critical defects in the code that would cause system failures. However, most of the issues that were generated where valid issues that are severe enought to warrant change.

Inspection Number: 2	
Date:	9.29.04
hackystat-dev-l message:	[HACKYSTAT-DEV-L:115] Review Request 9/29/04
Module and Review Id:	hackyJira - IssueSdt
Package:	org.hackystat.stdext.issue.sdt
Stats:	* Burt is the primary developer * Zero unit tests, 26% coverage * Has never been reviewed
Quality Level:	Low
Discussion:	This package was created by a brand new Hackystat developer. In my subjective feeling, was that new developers will have more defects than more experienced developers. Hence, there should be a ranking of developers based on some quanitifiable measurement of experience. Maybe just a configurable ranking would suffice. In addition to the developer issue, the package contained no unit tests which is also a major quality problem. It seems that any time a package has zero unit tests, the package should immediately inspected. Actually, there are a couple of alternatives;

	identify that a package should implement unit tests (1) if unit tests are implemented, then see how that affects its quailty ranking. If the package is still low in quailty then we should inspect it. Or identify that a package should implement unit tests, (2) if the developer responds that it is either too hard or impossible to do so, then most definitely we should conduct inspections. However, it seems that the bottom line for this specific selection was the inexperience of the developer. Hopefully, this inspection will provide an educational value for the developer.
Number of Issues:	19 review issues: - critical: 1 - major: 2 - normal: 8 - minor: 5 - trivial: 1 - unset: 2
Retrospective:	Although, this inspection was cancelled and only two members actually finished the individual phase many issues were generated. Many if not all of the issues were valid and required changes to the code at some level. I would claim that this package was definitely proven to be low quality and deserved the most in need of inspection destinction. I believe the primary indicator was the unexperience of the developer, in this case Burt. Also, note that we didn't find any mission critical issues that would cause the code to fail under normal circumstances.

Inspection Number: 3	
Date:	10.6.04
hackystat-dev-l message:	[HACKYSTAT-DEV-L:142] Review Request 10/06/04
Module and Review Id:	hackyKernel - UserMap, hackyJira - IssueSensor

Package:	org.hackystat.kernel.sensor.usermap, cm.atlassian.jira.event.listener in the hackyJira/src_listener directory
Stats:	- It is extremely fresh code and is already on the public server. - There were some junit failures in the latest build - And we haven't had issues sent to the server yet
Quality Level:	Low
Discussion:	This inspection was not triggered by using the current quality levels. Rather, it was selected because the JIRA sensor was not working correctly. However, there were some indicators on why this was low quality code.
Number of Issues:	Issues Sensor - 50 review issues: - critical: 5 - major: 9 - normal: 27 - minor: 5 - trivial: 0 - unset: 2 UserMap - 20 review issues: - critical: 0 - major: 5 - normal: 8 - minor: 5 - trivial: 1 - unset: 1
Retrospective:	This inspection generated numerous issues. This time some of them were mission critical, meaning that the issue could cause system failure. This is obvious because the JIRA sensor wasn't working. Many issues were generated about coding style and formatting. Obvisously, the number of generated issues we consider formatting and coding style as very important. Again, unexperienced developers played a major factor.

Inspection Number: 4

Date:	10.13.04
hackystat-dev-l message:	[HACKYSTAT-DEV-L:179] Fwd: Review Request 10/06/04 (typo: should read 10/13/2004)
Module and Review Id:	hackyKernel - UserMap2, hackyJira - IssueSensor2
Package:	org.hackystat.kernel.sensor.usermap, org.hackystat.stdext.sensor.jira
Stats:	- It is extremely fresh code and is already on the public server. - And we haven't had issues sent to the server yet
Quality Level:	Low
Discussion:	Again, this inspection was not triggered on quality levels. Instead, it was triggered because there were so many issues generated in the last inspection session that we decided to put the same code up for re-inspection.
Number of Issues:	IssuesSensor2 - 55 review issues: - critical: 1 - major: 11 - normal: 30 - minor: 8 - trivial: 2 - unset: 3 UserMap2 - 28 review issues: - critical: 0 - major: 4 - normal: 11 - minor: 11 - trivial: 2 - unset: 0
Retrospective:	Again, there were many issues generated. And again there were some mission critical issues generated that would cause system failure under normal conditions. The documentation that was provided lead to a lot of confusion and many issues were generated because of that. Again, unexperienced developers were the leading factor. In addition, I don't believe that the coverage or unit test attributes would have provided effective information in the level of quality.

Inspection Number: 5	
Date:	10.20.04
hackystat-dev-l message:	[HACKYSTAT-DEV-L:205] Review Request 10/20/04
Module and Review Id:	hackyKernel - UserMap3, hackyJira - IssueSensor3
Package:	org.hackystat.kernel.sensor.usermap, org.hackystat.stdext.sensor.jira
Stats:	
Quality Level:	Low
Discussion:	Again, this inspection was not triggered on quality levels. Instead, it was triggered because there were so many issues generated in the last inspection session that we decided to put the same code up for re-inspection.
Number of Issues:	IssuesSensor3 - 22 review issues: - critical: 0 - major: 5 - normal: 12 - minor: 4 - trivial: 0 - unset: 1 UserMap3 - 20 review issues: - critical: 1 - major: 1 - normal: 11 - minor: 5 - trivial: 2 - unset: 0
Retrospective:	No mission critical issues were identified. However, the code was substantially improved since the last inspection such that we were able to identify some higher-level design issues that weren't apparent before. Again, unexperienced developers were the leading indicator.

Inspection Number: 6	
Date:	
hackystat-dev-l message:	
Module and Review Id:	hackyTelemetry
Package:	
Stats:	
Quality Level:	Unsure
Discussion:	This inspection was proposed by Cedric with the knowledge that his code has never been inspected and the subjective claim that it is complicated code. This brings up an important point, the quality analysis that I provide is no better than ones own subjective feeling about the code. In other words, a inspection process should never give up the right to inspect code that is being volunteered and hackyQuality will never relace that. However, that does not mean that hackyQuality is inferior to the subjective feelings (volunteering), I would claim that the quality analysis will identify code that should be inspected that no one has ever thought about or have forgotten. It seems that the quality analysis is just a tool that inspection managers can use to identify potential areas of interest. We can't inspect everything; volunteering code will work to a certain extent, but we are likely to miss some code that isn't voluteered. I think hackyQuality will help find the code that seeps through the cracks.
Number of Issues:	
Retrospective:	

Inspection Number: 7

Date:	
hackystat-dev-l message:	
Module and Review Id:	hackyStdExt
Package:	org.hackystat.stdext.unittest.dailyproject, org.hackystat.stdext.coverage.dailyproject
Stats:	
Quality Level:	Unsure
Discussion:	These two packages should be similar in nature in that they process Sensor Data that contain only a Java class name and the associated data. In other words, the Sensor Data does not contain a workspace. Therefore, these two daily project data implementations must use the JavaClassWorkspaceMapManager to associate a classname with a workspace. This is where the similarities end. The two daily project data implementations are quite different and make design decisions that are probably functionally correct, but are hard to understand when looking at both of them. I believe that the two daily project data implementations should be similar; therefore it will be easier to understand.
Number of Issues:	
Retrospective:	