

# Hackstat-SQI: Modeling Different Development Processes

Aaron Kagawa  
Philip Johnson  
Collaborative Software Development Laboratory  
Department of Information and Computer Sciences  
University of Hawaii  
[kagawaa@hawaii.edu](mailto:kagawaa@hawaii.edu)  
[johnson@hawaii.edu](mailto:johnson@hawaii.edu)

Last update: 07/21/2004 11:00:21 AM

## Introduction

This paper is a design document for the Hackstat-SQI system, which will support multiple projects at the Jet Propulsion Laboratory. The fundamental difference between this Hackstat system and other Hackstat systems is the differences in the development processes of the projects we will be supporting.

The JPL projects that we will be support all use Harvest for version control. However, that is where the similarity begins and ends because Harvest can be configured differently for each project. There are two main areas of differences in the configurations of Harvest: (1) Harvest Package Model and (2) Harvest State Change Lifecycle.

- The **Harvest Package Model** represents a way to categorize the different types of "work" that is done on the system. The different types of work are represented with Packages with in the model. For example, a defect can be represented with package type "A" and a new functionality can be represented with package type "B". See the Harvest Package Model section for more detailed information.
- The **Harvest State Change Lifecycle** represents how these packages move through a development cycle. The lifecycle contains a set of Harvest States that represents exactly where in the development cycle the packages is currently in. For example, a package that is being currently developed can be in the "Dev" Harvest State. See the Harvest State Change Lifecycle section for more detailed information.

There has been an ongoing discussion between Rich Hug and I whether we should "normalize" the data before sending it off to Hackstat. (By normalized we mean fit specific package models and state change lifecycles into a generic mapping). However, we feel that the differences between the model and lifecycles should not be hidden, because in fact these differences can affect quality! Therefore, we have decided that we shall send all data to Hackstat in their true form and allow Hackstat to process the differences.

The following sections address problems and solutions that Hackstat-SQI must address to be able to correctly interpret data from different projects.

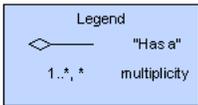
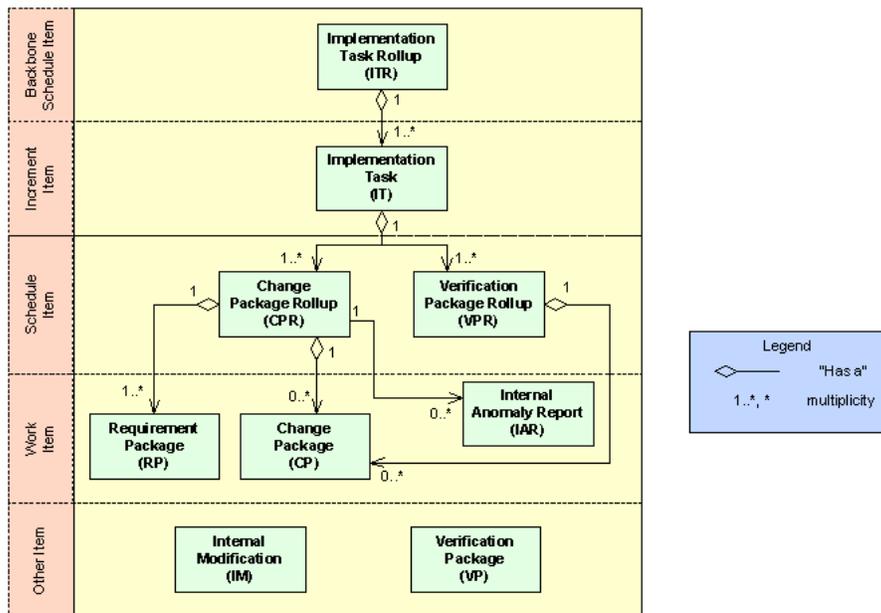
## Harvest Package Model

As I explained in the introduction, different projects can have different Harvest Package Models. While this is perfectly reasonable, the differences in the package models presents a problem in Hackstat. Basically, Hackstat must be able to interpret the meaning of the different packages and be able to calculate relevant information for these packages. It has been proven in the Hackstat-MDS system, where we constructed a Hackstat system for a single JPL Project that "package types" and their meanings are an important piece of the puzzle. In Hackstat-MDS we were able to do this by "hard coding" in the significance of the Package Types and implemented specific code that tailored to the MDS Harvest Package Model. However, in Hackstat-SQI "hard coding" the Harvest Package Model is not possible.

Therefore, Hackstat-SQI needs a way to interpret different Harvest Package Models, in a sense mapping the packages to generic meanings. Here are two Harvest Package Models from the MDS and TCDM projects

### MDS Harvest Package Model:

MDS Package Model has a complex structure that groups these packages into a hierarchical structure. The following figure provides this structure:



The following table proves some explanation of the significance of the packages.

MDS Harvest Package	Significance
Implementation Task Rollup (ITR)	
Implementation Task (IT)	
Change Package Rollup (CPR)	
Verification Package Rollup (VPR)	
Requirement Package (RP)	
Change Package (CP)	represents new "work" or functionality
Internal Anomaly Report (IAR)	represents a defect
Internal Modification (IM)	represents a small change
Verification Package (VP)	

**TCDM Harvest Package Model:**

Unlike the MDS Harvest Package Model, the TCDM Harvest Package Model does not have a complicated structure. It simply contains the following 4 package types:

TCDM Harvest Package	Significance
New Request (NR)	represents new work
Change Request (CR)	
External Anomaly Report (AR)	a defect in released software
Internal Anomaly Report (IAR)	
Internal Modification *	this package is generated from meta data provided in IAR packages

\* The TCDM Internal Modification Package Type is a special package type because it is generated from meta data provided in IAR packages.

- (Philip's Thoughts) My thinking about the TCDM IAR package is that this is mostly a bug in their representation; they really should have two separate packages. But, that's not something we can dictate. Also, I don't think we want to represent the TCDM IAR package as having both the MinorChangePackage and the InternalAnomalyType, because any individual instance is either one or the other but never both.

The way to resolve this really depends upon what will be done with the data. The simplest thing is to just make the TCDM IAR a subtype of the InternalAnomaly type. Then the people on the project see the analyses the way they think about things in Harvest; its going to be 'natural' to them. It might be that they voluntarily decide they want an additional package type (IM) once they see the benefits that would accrue from having it. In other words, let them figure out that they have a bug in their representation and see if they become motivated to fix it themselves because they think fixing it will have value for them.

On the other hand, if the immediate goal is cross-project comparison, then it seems like we have to manually categorize each of their IARs as either a true IAR or as an IM. That's going to be a hassle. But I don't think having the IAR be both an IAR and IM at the same time is the right way to go.

- (Rich Hug's Thoughts) The only problem we have is when a certain type of package can be used for 2 different types of changes (TCDM IAR - internal bug and small change) (MDS - IAR - internal and external problem).

I think this problem should be handled by the Harvest extraction script. Based on meta data within the package, it is possible to decide which type of package it is. I would propose that the extraction script used for a particular project to be customized to identify and rename the package as appropriate (e.g., IAR-00010 becomes IM-IAR-00010 or AR-IAR-00010).

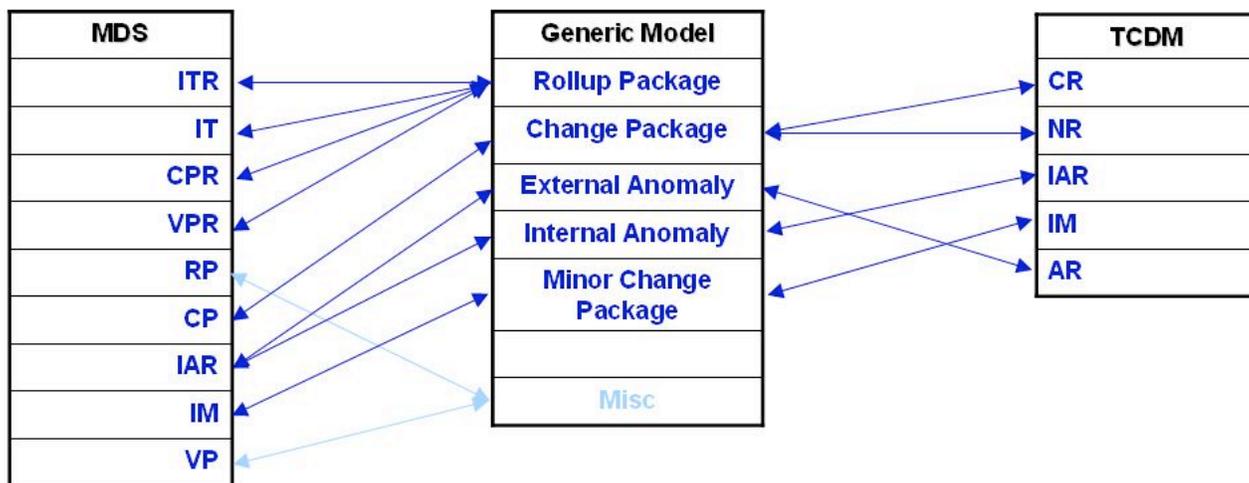
We have concluded that we will "normalize" the data, thus creating an IM package from IAR meta data.

**The Problem**

The fundamental problem that Hackstat-SQI faces is how to calculate the Package Level information without having to "hard code" package type information for each Harvest Package Model. For example, one could think of an analysis that calculates the average age of a package that represents new "work" or functionality. To implement a generic analysis that works for all projects the code needs to be able to determine what the package type within a package model represents new functionality.

**The Solution**

The solutions to the problem is to create a Mapping system that allows a specific Harvest Package Model to map to a Generic Model. The following graphic is a mapping of MDS and TCDM to a generic model.



This mapping has the following properties:

- All MDS and TCDM package types maps to a package type in the Generic Model
- All Generic Model package types need NOT map to a MDS or TCDM package type
- Mappings can be either a 'one to one' mapping or a 'one to many' mapping. For example, the MDS IAR package type represents both Internal Anomalies and External Anomalies.

**Harvest Package Model Mapping in Relational Database Table form**

To make this a little easier to visualize I've converted the previous graphical representation of the Mapping into Relational Database Tables. Basically, all we need are two tables for each model; (1) a table for the Package Types and (2) an intersect table that maps the Package Types to the generic model.

Generic Harvest Package Model Table	MDS Harvest Package Model Table	TCDM Harvest Package Table
Rollup Package	Implementation Task Rollup (ITR)	New Request (NR)
Change Package	Implementation Task (IT)	Change Request (CR)
External Anomaly	Change Package Rollup (CPR)	External Anomaly Report (AR)
Internal Anomaly	Verification Package Rollup (VPR)	Internal Anomaly Report (IAR)
Minor Change Package	Requirement Package (RP)	Internal Modification (IM)
Misc Package	Change Package (CP)	
	Internal Anomaly Report (IAR)	
	Internal Modification (IM)	
	Verification Package (VP)	

Generic-MDS Harvest Package Model Mapping Table (Intersect Table)	
MDS	Generic
Implementation Task Rollup (ITR)	Rollup Package
Implementation Task (IT)	Rollup Package
Change Package Rollup (CPR)	Rollup Package
Verification Package Rollup (VPR)	Rollup Package
Requirement Package (RP)	Change Package
Change Package (CP)	Change Package
Internal Anomaly Report (IAR)	Internal Anomaly
Internal Modification (IM)	Minor Change Package
Verification Package (VP)	Misc Package

### Harvest Package Model Mapping in XML form

Because Hackstat currently does not use a Relational Database, we need to store the mapping information in XML files. The advantage of using XML files in this situation is that when a new Harvest Package Model is introduced the Hackstat administrator would just have to create a XML file that provides the mapping and restart the server. Hackstat-SQI would read in the XML file and dynamically have the mapping. This approach is much more flexible than using a type-safe enumeration approach, because we would then have to compile and rebuild the Hackstat-SQI system. The drawback to using XML files is that it is difficult to ensure the correct values (schema is perfect for this, however, currently Hackstat does not utilize this in any of our XML files).

It seems that the Generic Model can be hard coded into the system, because this list of packages probably will not change with the introduction of a new project.

```
<harvestpackagemodel type="mds">
  <harvestpackage name="Implementation Task Rollup" type="ITR" supertype="Rollup Package" />
  <harvestpackage name="Implementation Task" type="IT" supertype="Rollup Package" />
  <harvestpackage name="Change Package Rollup" type="CPR" supertype="Rollup Package" />
  <harvestpackage name="Verification Package Rollup" type="VPR" supertype="Rollup Package" />
  <harvestpackage name="Requirement Package" type="RP" supertype="Change Package" />
  <harvestpackage name="Change Package" type="CP" supertype="Change Package" />
  <harvestpackage name="Internal Anomaly Report" type="IAR" supertype="Internal Anomaly" />
  <harvestpackage name="Internal Modification" type="IM" supertype="Minor Change Package" />
  <harvestpackage name="Verification Package" type="VP" supertype="Unmatched" />
</harvestpackagemodel>
```

This XML implementation works well for the MDS Harvest Package Model. However, the TCDM model presents a problem.

```
<harvestpackagemodel type="tcdm">
  ...
  <harvestpackage name="Change Request" type="CR" mapping="Work Package" />
  <harvestpackage name="New Request" type="NR" mapping="Work Package" />
  <harvestpackage name="Internal Anomaly Report" type="IAR" mapping="Internal Anomaly" />
  <harvestpackage name="Anomaly Report" type="AR" mapping="External Anomaly" />
  ...
</harvestpackagemodel>
```

Rich Hug has explained that the IAR type in the TCDM model seems to be a "catch all" package type, in that IARs have different meanings. For example, some IARs are really IMs. The problem is whether the XML should look like the following.

```
<harvestpackagemodel type="tcdm">
  ...
  <harvestpackage name="Internal Anomaly Report" type="IAR" mapping="Internal Anomaly, Minor Change Package" />
  ...
</harvestpackagemodel>
```

The problem with this representation is that we will not know when a IAR is actually a Internal Anomaly or a Minor Change Package. There are a couple solutions to this problem; (1) we let Rich "normalize" the data on the Harvest side and send "IM"s instead of IARs when appropriate or (2) have another attribute in the State Change data so on the Hackstat side we can make that determination.

It has been determined that we will "normalize" the package data to generate IM's based on the IAR's meta data. Therefore, this gives the following mapping:

```
<harvestpackagemodel type="tcdm">
  <harvestpackage name="Change Request" type="CR" supertype="Change Package" />
  <harvestpackage name="New Request" type="NR" supertype="Change Package" />
  <harvestpackage name="Internal Anomaly Report" type="IAR" supertype="Internal Anomaly" />
  <harvestpackage name="Internal Modification" type="IM" supertype="Internal Modification" />
  <harvestpackage name="Anomaly Report" type="AR" supertype="External Anomaly" />
</harvestpackagemodel>
```

### How to use the Harvest Package Model Mapping in Java

The Java implementation to manage the different Harvest Package Models includes representations for the individual models, mappings, and reading the xml files. In addition

to the Harvest Package Models there must also be representations for Harvest Packages. These representations calculate the properties of Harvest Package. For example, a Work Package representation will include the number of files, builds, etc.

HarvestPackageModel

## Harvest State Lifecycle

As I explained in the introduction, different projects can have different Harvest State Lifecycles. While this is perfectly reasonable, the differences in the state change lifecycles presents a problem in Hackstat. Basically, Hackstat must be able to interpret the meaning of the different Harvest States and be able to calculate relevant information. It has been proven in the Hackstat-MDS system, where we constructed a Hackstat system for a single JPL Project that "harvest states" and their meanings are an important piece of the puzzle. In Hackstat-MDS we were able to do this by "hard coding" in the significance of the Harvest States and implemented specific code that tailored to the MDS Harvest State Change Lifecycle. However, in Hackstat-SQI "hard coding" is not possible.

Therefore, Hackstat-SQI needs a way to interpret different Harvest State Change Lifecycles, in a sense mapping the states to generic meanings. Here are two Harvest State Change Lifecycles from the MDS and TCDM projects

### MDS Harvest State Change Lifecycle:

The following table proves some explanation of the significance of the states .

MDS Harvest States	Significance
Created	
Dev Waiting	
Dev	
Dev Null	
Dev Complete	
Build Queue	
CM Build	
Integration Test	
Test Complete	
Release	
Release-5.1	
Release-5.2	
Release-<version>	

### TCDM Harvest State Change Lifecycle:

The following :

TCDM Harvest States	Significance
Created	
AR/CR Rejected	
AR/CR Postponed	
AR/CR Waiting	
AR/CR Review	
Coding	
Dev Null	
Dev Complete	
CM Build	
Subsystem Test	
Test Complete	
Test Hold	
Release	
Release_V28.0	
Release_V28.1	
Release_V<version>	

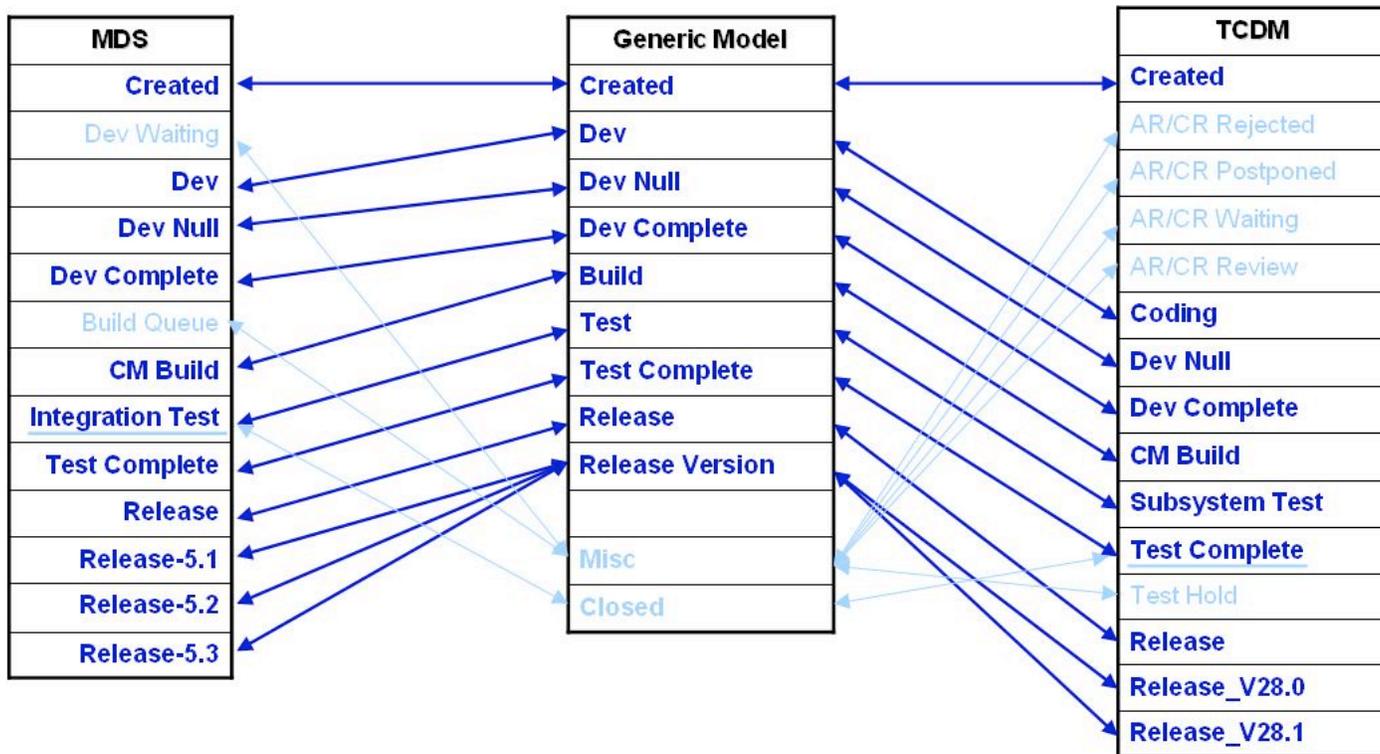
**The Problem**

The fundamental problem that Hackstat-SQI faces is how to interpret Harvest State information without having to "hard code" state information for each Harvest State Lifecycle. For example, one could think of an analysis that calculates the number of days a package spends in development. In MDS this state is "DEV" and in TCDM this state is "Coding". To implement a generic analysis that works for all projects the code needs to be able to determine what the harvest state within a state change lifecycle represents "development".

As more project enter the Hackstat-SQI it is unreasonable to assume that we can keep hard coding each of the Harvest State Change Lifecycles.

**The Solution**

The solutions to the problem is to create a Mapping system that allows a specific Harvest State Change Lifecycle to map to a Generic Lifecycle. The following graphic is a mapping of MDS and TCDM to a generic lifecycle.



As you can see, the mapping between the Harvest States is straightforward.

- Both MDS and TCDM harvest states, except the "Release Version" states, have a one to one relationship with the Generic Model.
- Both MDS and TCDM have a state that represents a Closed Package
- Both MDS and TCDM lifecycles have states that do not match the generic model and therefore Hackstat-SQI cannot have a generic analysis that involves any calculations with these unmatched. However, these states are very important to the project's specific Harvest State Change Lifecycle and project-specific analyses can be created to analyze these states when appropriate.

**Harvest State Change Lifecycle Mapping in Relational Database Table form**

To make this a little easier to visualize I've converted the previous graphical representation of the Mapping into Relational Database Tables. Basically, all we need are two tables for each lifecycle; (1) a table for the Harvest States and (2) an intersect table that maps the Harvest States to the generic lifecycle.

Generic Harvest State Change Lifecycle Table	MDS Harvest State Change Lifecycle Table	TCDM Harvest State Change Lifecycle Table
Created	Created	Created
Dev	Dev Waiting	AR/CR Rejected
Dev Null	Dev	AR/CR Postponed
Dev Complete	Dev Null	AR/CR Waiting
Build	Dev Complete	AR/CR Review
Test	Build Queue	Coding

Test Complete	CM Build	Dev Null
Release	Integration Test	Dev Complete
Release Version	Test Complete	CM Build
Misc	Release	Subsystem Test
Closed	Release-5.1	Test Complete
	Release-5.2	Test Hold
	Release-<version>	Release
		Release_V28.0
		Release_V28.1
		Release_V<version>

Generic-MDS Harvest State Change Lifecycle Mapping Table (Intersect Table)	
MDS	Generic
Created	Created
Dev Waiting	Misc
Dev	Dev
Dev Null	Dev Null
Dev Complete	Dev Complete
Build Queue	Misc
CM Build	Build
Integration Test	Test
Test Complete	Test Complete, Closed
Release	Release
Release-5.1	Release Version
Release-5.2	Release Version
Release-<version>	Release Version

### Harvest State Change Lifecycle Mapping in XML form

Because Hackstat currently does not use a Relational Database, we need to store the mapping information in XML files. The advantage of using XML files in this situation is that when a new Harvest State Change Lifecycle is introduced the Hackstat administrator would just have to create a XML file that provides the mapping and restart the server. Hackstat-SQL would read in the XML file and dynamically have the mapping. This approach is much more flexible than using a type-safe enumeration approach, because we would then have to compile and rebuild the Hackstat-SQL system. The drawback to using XML files is that it is difficult to ensure the correct values (xml schema is perfect for this, however, currently Hackstat does not utilize this in any of our XML files).

It seems that the Generic Model can be hard coded into the system, because this list of packages probably will not change with the introduction of a new project.

```
<harveststatelifecycle type="mds">
  <harveststate name="created" order="0" supertype="created" />
  <harveststate name="dev waiting" order="1" supertype="misc" />
  <harveststate name="dev" order="2" supertype="dev" />
  <harveststate name="dev null" order="3" supertype="dev null" />
  <harveststate name="dev complete" order="4" supertype="dev complete" />
  <harveststate name="build queue" order="5" supertype="unmatched" />
  <harveststate name="cm build" order="6" supertype="build" />
  <harveststate name="integr test" order="7" supertype="test" />
  <harveststate name="test complete" order="8" supertype="test complete" closed="true" />
  <harveststate name="release" order="9" supertype="release" />
  <harveststate name="release-5.1" order="10" supertype="release version" />
  <harveststate name="release-5.2" order="11" supertype="release version" />
  <harveststate name="release-5.3" order="12" supertype="release version" />
</harveststatelifecycle>
```

Notice the "closed" attribute in the test complete entry, this attribute indicates what harvest state represents a closed package. (I don't know if this is the best way to do this).

### How to use the Harvest State Change Lifecycle Mapping in Java

The Java implementation to manage the different Harvest State Change Lifecycle includes representations for the individual lifecycles, mappings, and reading the xml files.

## Other Problems

There is one last twist to this issue. It is possible for different projects to use the same Harvest Package Model and/or Harvest State Change Lifecycle. Therefore, this means that we cannot associate one Model/Lifecycle with a Project. I propose that we add a Key-Value attribute to Project representations. This will enable Hackstat-SQI to determine what Model/Lifecycle a project is using.

```
Project project = ProjectSelector.getSelectedProject(user);  
  
HarvestStateChangeLifecycle projectLifecycle = project.getKeyValuePair(Project.Lifecycle).getValue();  
HarvestState testState = projectLifecycle.getMapping(GenericLifecycle.TEST);
```

This example code demonstrates the ability to look determine the specific Harvest State change Lifecycle associated with a project. And when the lifecycle is known we can then use the mapping to get a specific Harvest State associated with that lifecycle.

- (Philip's Thoughts) This is a good idea, but doesn't require us to extend the Project representation. Instead, I recommend that you implement a new package in SQI called `org.hackstat.app.jpisqi.lifecycle`. This package includes a Preference command (maybe at the admin level) that allows the user to associate a given Project with a given Lifecycle. Then, to find out the lifecycle, you would use something like:

```
Project mdsProject = ProjectManager.getInstance().getProject("MDS");  
ProjectLifecycle lifecycle = ProjectLifecycleManager.getInstance().getLifecycle(mdsProject);
```

Of course, ProjectLifecycleManager would persist these mappings using the conventional UserXML stuff.

Another problem associated with different Lifecycles and Models is the Hackstat HTML Interface. Because, projects can have different lifecycles and models with different types and states the interface should be dynamic depending on what project is selected. For example, if the MDS world wants to know how many packages entered the Build Queue state they should be able to select Build Queue without seeing the AR/CR Review state from the TCDM lifecycle.