Project Number. GFP0904

WPI Suite Metrics Module

A Major Qualifying Project Report:

submitted to the faculty of the

WORCESTER POLYTECHNIC INSTITUTE

in partial fulfillment of the requirements for the

Degree of Bachelor of Science

by:

_____

_____

Donald Gaxho

Chaoran Xie

Date: April 28, 2010

Approved:

_____

Professor Gary F. Pollice, Major Advisor

1. Metric
2. Software Engineering
3. Requirement Management

Abstract

The WPI Suite Requirement Metrics project seeks to expand WPI Suite's capabilities as a software project management tool by creating a framework for defining requirement measurements and metrics that provide a customizable form of project health assessment. The module uses data gathered from the existing WPI Suite requirement metrics module, but is designed to be extensible enough to handle future requirement definitions that conform to our Artifact model. The customizable creation of new metrics and measurements can be done by developers by writing simplified code that harnesses our framework, or through a code-free GUI based creation process.

Acknowledgements

We would like to acknowledge Professor Gary Pollice for his consistent guidance and support throughout the development of this project.

We would also like to thank the original WPI Suite development team for giving us the ability to build upon a project management tool that will be very valuable to future WPI software engineering students.

Finally, we would also like to recognize Paul Kehrer and Anthony Azersky, who developed a similar WPI Suite module dealing with project health assessment, for their cooperation and assistance in the development of our initial framework design.

Table of Contents

List of Illustrations

List of Tables

1. Introduction

Software development projects require constant, up to date monitoring of their progress to ensure success. The quality and stability of software requirements can be one such useful indicator of a project's health. According to Shroff, "good requirements management practices help improve customer satisfaction, lower the system development costs, and increase the chance of having a successful project." (2001) When used in combination with other software metrics, requirements management metrics can be a critical component of project management as a whole. However, there is some debate as to what requirement metrics are most useful, and many metrics have varied definitions.

WPI Suite started as a WPI Software Engineering project in 2009, and aims to provide an integrated environment for project management. This tool gives software development teams the ability to create and view projects, constituent members, and several indicators of project performance, such as traceability and project velocity. WPI Suite's goal is to streamline project development, increase productivity and provide the ability to detect the health of WPI computer science projects.

This project's goal is to improve WPI Suite's ability to assess project health through the use of data gathered from project requirements. Our team provides a module that will give WPI Suite users the power to create the requirement measurements and metrics that best address their individual project needs. Our framework is at the same time flexible enough to work with any future changes to the requirement types that are currently defined in WPI Suite.

2. Background

2.1 Measurement and Metrics

The concepts of measurements and metrics were central in addressing our goal to gather requirement data that could be used to assess project health. Fenton describes measurement as “a process that assigns numbers or symbols to attributes of entities in the real world according to clearly defined rules” (1998). Thus a measurement is something that ultimately evaluates to a number or symbol which represents some value associated with a property of a certain entity. This simple definition drove our framework design, as we could see how the entities in the definition could represent WPI Suite requirement artifacts, and this idea made it obvious that the concept of requirement attributes had to also be introduced to identify individual artifact properties to be measured.

The concept of a metric directly builds upon that of a measurement. Whereas a measurement is a single number representing a certain property of an entity, a metric additionally defines the procedures for interpreting and assessing the measurement. One of the ways that the idea of interpretation is attached to a measurement is represented by a distribution metric. Distribution metrics are sets of measurements which differ on a certain attribute. By outputting a measurement for each entity on the differing entity property, the individual measurement ceases to be simply a number, but can be interpreted in the context of the other related measurements represented by the metric.

We realized that the creation of metrics would be the ultimate goal of our project, as their support for measurement interpretation would actually allow users to assess their meaning as indicators of project health. Of course before we could design the concept of

metrics in our framework, we decided to first define measurements that we could ultimately construct metrics from.

2.2 Examples of Standard Measurements and Metrics

Due to the importance of requirements in scheduling, cost estimation, and resource allocation, it is important for project managers to quantify "a set of measures that can be analyzed to determine the effectiveness of the processes set in place to control, manage, and trace the requirements" (Persse, 2009). The following is a set of measurements and metrics that would likely be considered by most project managers.

Table 1: Common Requirement Measurements & Metrics

Measurement	Metric
Total number of original requirements	Percentage of completed requirement by given user
Number of requirements in priority HIGH/LOW/MEDIUM	Average requirement priority for all requirements
Number of requirements complete/incomplete	Average number of revisions per requirement
Number of requirements that are functional/non-functional	Number of revisions per requirement
Length to implement the requirement	Number of requirements per iteration
Number of requirements that contain word of choice	Number of requirements owned by each user

For each individual requirement, software development teams might be interested in the total number of revisions, since requirements that change often tend to be less stable and more difficult to implement. Managers might also be interested in particular characteristics of a requirement attribute. For example, they might want to know which requirements are defined as functional or non-functional, and may also be interested in their priority.

Measurements are generally defined as the number of requirements that satisfy a given condition. For example, one simple measurement might be the total number of complete or incomplete requirements. We can also look at the number of requirements that satisfy more than one condition, such as the number of complete requirements owned by a given user.

Very often, the measurement might not be very valuable when out of context. This can be seen in a measurement like the number of user stories owned by a given user. Without knowing the total number of user stories, there is no way to know the significance of the measurement. Creating more complex metrics puts these measurements into context, such as with the percentage of user stories owned by a given user.

A further analysis of the percentage metrics shows that they can be defined as the division of two measurements. This can be seen in a metric that is defined by a solution space divided by a domain space, which are both provided by measurements. A simple example to illustrate this idea would be the calculation of the percentage of complete tasks done by a given user over all user stories. The domain space in this case would be

the number of tasks completed by all the users while the solution space is the number of task completed by the user of interest. In all cases, the solution space is stricter than the domain space.

Average metrics additionally provide information about the overall requirements. For each requirement there might be either a number of revisions associated with it or a priority such as one on a scale of 1 to 5. Software development teams would be interested in the average number of revisions, and average priority value for requirements in a given domain of requirements.

Finally, metrics based on a given time period are also good indicators of project health. Time is often represented by iterations or other similar time units. The trend analysis of these metrics gives insight into the future success or failure of the project. Knowing the percentage of requirement completed over multiple iterations can reveal far deeper insight than simply viewing the percentage for any given iteration.

2.3 Limitations of Requirement Management in WPI Suite

Many of the standard requirement metrics and measurements previously defined can only be gathered if the entity properties that they measure are defined within an existing requirement management system. In the case of WPI Suite, some of these properties were not defined within existing requirement types. For example, in the model we were working with, the concept of a functional or non-functional requirement was not defined as a user story artifact attribute. Likewise, the concept of time is not explicitly

associated with each requirement artifact, which does not allow us to track its development and modification over time.

While WPI Suite's requirement artifact model provides much flexibility in defining requirements that include these needed attributes, the existing artifact definitions did not provide all the information required by many common measurements and metrics. Because this project's main goal was to focus on requirement metrics generation and not requirement management, and additionally because of the wide variety of existing standard metrics and measurements, we decided to focus our efforts on creating a framework that would give future developers the flexibility to create any metric that they might determine useful to their project health assessment efforts.

Rather than working to create more requirement artifacts that would be structured to support the vast number of standard measurements and metrics available, we decided instead to work within the limitations of the existing WPI Suite requirement management interfaces, knowing that future development could rework the requirement artifacts which our creation process would support.

2.4 Requirement Management Tools

Research into existing requirement management tools has revealed several common functionalities and user interface practices. Some of the major requirements management tools that have been analyzed are TechExcel DevSpace™, IBM Rational DOORS™, Seapine TestTrack RM™, *Borland* CaliberRM™, Accompa's Accompa RM™, the open source OSRMT (Open Source Requirements Management Tool),

Gatherspace.com's GatherSpace™, Microsoft Visual Studio Team System 2010™, and Serlio CaseComplete™. Four key functionalities emerged. These are the hierarchal view, filter view, multiple artifact types and traceability.

The hierarchal view functionality describes the ability of requirement tools to categorize the requirements into subgroups. This can be seen in almost all the tools we researched. The filter view's functionality additionally allows users to further filter requirements within a section. Many of these tools allow the user to utilize multiple artifact types, such as test cases, user stories, features and more. This feature allows for traceability between all of the various artifact types.

2.5 Limitations of Metrics

Thought metrics are helpful in understanding the health of the project, they are not without their limitations. Shroff describes several common metric limitations :

1. Many metrics do not use the same units. For example, iteration time might be in days or months.
2. Many metrics may not have the same context. For example, we consider the case where a requirement changes 5 times and another changes 10 times. This does not necessarily mean that the first requirement is more stable because the first requirement could change 5 times in a day while the second requirement changes 10 times in a year.
3. It is difficult to interpret the raw metrics data. For example one requirement might have 10 simple test cases associated with it while another requirement

might have only one complex test case. It is hard to determine how well a requirement is tested by just looking at the number of test cases associated with the requirement.

4. The most important question to answer after gathering this raw data is what to do with the results. The results often need to be further analyzed in context before being compared and provide useful meaning.

Because of these limitations, it is often difficult and not very valuable for a requirements metrics tool to define all the metrics that a specific project manager will need to monitor the progress of their project. Instead, it is much more effective to provide the user with a way to create their own metrics based on their specific project needs.

3. Methodology

3.1 The Need for a Framework

Our research into requirement measurements and metrics revealed two important considerations for our module.

1. There is no fixed/limited number of measurements or metrics that all software development teams would like to know.
2. The schema of the existing is like to change, both addition and modification of existing tables is very likely to happen.

Knowing these facts, we realized that we could not hard code each of the measurements or metrics. One reason is because we have no way of being sure what each project team would be interested in for their particular project. Additionally, some of the underlying database tables that are required for certain metrics do not currently exist. Because of this, we recognized the need for a framework for measurement and metric creation.

3.2 Requirements of the Framework

We defined two primary requirements for the framework itself.

- The framework needs to allow easy creation of measurements and metrics through code, and if time permits through a GUI based creation process.

- The framework needs to be database independent. This requires both DBMS independence (i.e. Oracle® vs. MySQL®), and independence from the chosen database schema.

3.3 Technology Required

To satisfy the requirement of database independence, we researched into several Object Relational Mapping libraries and eventually settled on the ActiveObjects open source library. ActiveObjects was chosen for three reasons. It gives the ability to access most major database systems, including Oracle and MySQL. Additionally, it has a low learning curve. Although ORMs such as Hibernate are much more powerful, the high learning curve would prevent us from finishing the project on time. ActiveObjects was also a natural choice because it was already being used in the WPI Suite Requirement Management Module.

3.4 Limitations of the Technology

As we found out later into our project, ActiveObjects came with some limitations. These limitations did impact the design of our framework.

3.4.1 Lack of Support for Complex Where Statements

By default, ActiveObjects does not support the building of queries with compound ‘where’ statements. Any compound ‘where’ conditions need to be constructed ahead of time and passed to ActiveObjects as a single string. In order to adjust for this, we realized that we needed to create a query builder component into our framework.

3.4.2 Lack of Support for Foreign Key Conditions

ActiveObjects does not support complex foreign key ‘where’ conditions that specify relations to attributes of multiple tables. To make up for this, we created query building components into our framework that handle foreign key relations.

3.4.3 Lack of Support for Listing of the Attributes in a Given Table

Because we wanted to provide a GUI creation process on top of our framework, we needed an easy way to find the attribute names and types of a given database table. This is because a GUI would need to give the user a selection of attributes to do operations on. AO currently does not support this feature, but it does require each database table to have an associated interface that defines the object mapping, with interface methods corresponding to table attributes. We got around our problem by utilizing Java reflection to look up the methods in these interfaces and obtaining the attributes that they were associated with. The following shows our method for obtaining artifact attributes through reflection.

```
private Object[] getAttributesFromClass(Class<? extends Entity> class1)
{
	ArrayList<Method> getMethods = new ArrayList<Method>();
	Object [] attributeNames;
	attributeNames = new Object[]{};
	Class c = class1;
	allMethods = c.getMethods();


	// Get all non foreign key getter methods of the artifact class
	for (Method m : allMethods) {
		if(isGetter(m) && !isForeignKey(m))
		getMethods.add(m);
	}
	attributeNames = new Object[getMethods.size()];
	attributeMap.clear();

	// Extract the attribute name from the getter methods
	for(int i=0; i <getMethods.size(); i++ ){
		String name = getMethods.get(i).getName().substring(3);
		attributeNames[i]= name;
		attributeMap.put(name, getMethods.get(i));
	}
	return attributeNames;
}
```

3.5 The Framework

The first major component of our framework is the artifact model. We realized that in order to obtain measurements from the requirement data available we needed to first target the specific rows in a database table. The following Figures illustrate the terminology used in our artifact model component.

Artifact: UserStory

UserStoryID	Project	Author	Owner	Title	Description
1	Project 1	dgaxho	dgaxho	ATM withdrawals	Using new software system, allow for account withdrawals.

Figure 1: Artifact Model Table Example

We call the information which the table represents the artifact. In this case, a user story is one example of such an artifact. We call the column attribute of the database table the attribute of our artifact model. In this case, a user story's description is given as an example of an attribute.

The content of an attribute can be limited by a condition thus forming a filter on the overall measurement we want to define. In this case, the filter consists of the attribute named "description" with a condition specifying that it must contain the keyword "software".

Measurements are then fully defined by one or more filters. An example of a measurement constructed from the previously defined filter would be the number of user stories with description containing "software".

Metrics are defined by one or more measurements along with their interpretation. For example, for a measurement such as all the user stories completed by either Don or Chao in the project named project 1 and in the iteration called iteration 1, the interpretation is provided by virtue of the fact that it results in a set of measurements that can be compare with each other. A summary of this design is presented in the following diagram.

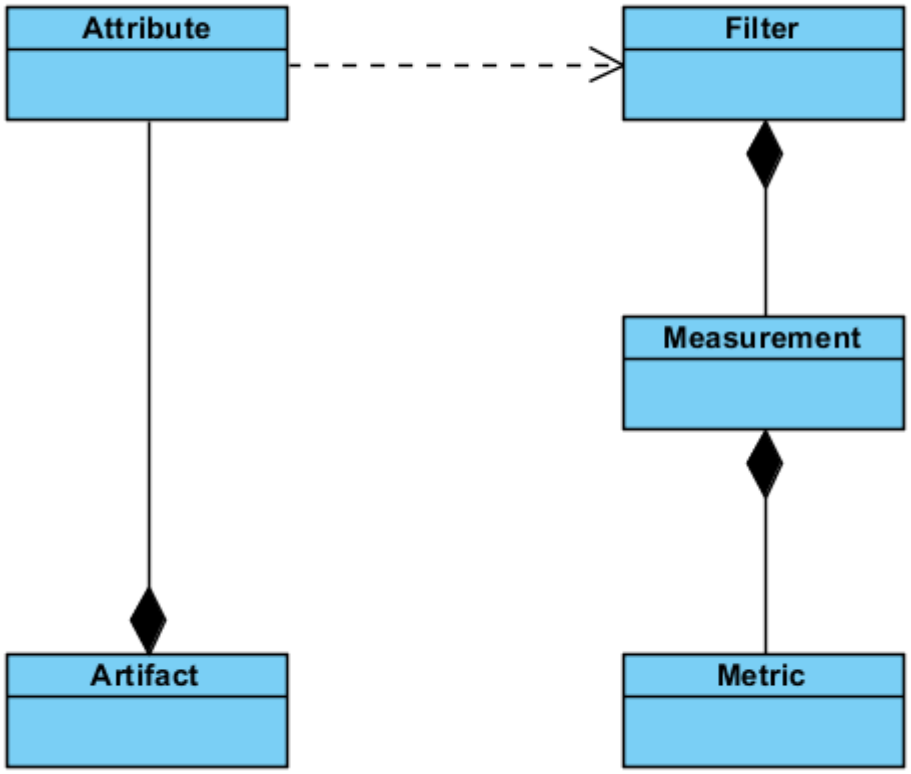

Figure 2: Framework UML Diagram

The next major component of our framework is the query builder which takes each of the filters on an attribute and creates a properly formatted SQL ‘where’ statement. This SQL statement is then passed to AO to be evaluated.

An additional component which builds on top of the query builder is the foreign key component, which allows user to create filters on attributes that are related by foreign keys.

4. Results and Analysis

The goal for this project was to design a framework that would allow software development teams to create any metric that they determine to be a good project health indicator for their project. To show that our framework was capable of this, we utilized it to implement several metrics. One metric that we feel best demonstrates the versatility of our framework is the following:

Distribution of all complete user stories by either “Don” or “Chao” in project "Test Project 1” in iteration "Iteration 1".

This metric is a good example because it has two key properties. It involves multiple filters:

1. userstory.author = Don OR userstory.author = Chao
2. userstory.status = complete
3. userstory.project.name = “Test Project 1”
4. userstory.iteration.name = "Iteration 1".

As shown in filters 3 and 4, this metric also involves relations between multiple artifacts. There is a relation between user story artifact and project artifact as well as between user story and iteration.

To construct a metric using our framework, one can either directly implement the metric by writing code that leverages the framework, or use our GUI prototype for metric creation. In both cases, the first step in defining a metric is to define the measurement that it is based on.

In our GUI creation process, when a user wants to create a new measurement, they are asked to give the measurement both a name, description and the artifact that it is based on. After this, they are able to create one or more filters to build up the measurement. The following Figure shows the creation of a filter using our GUI.

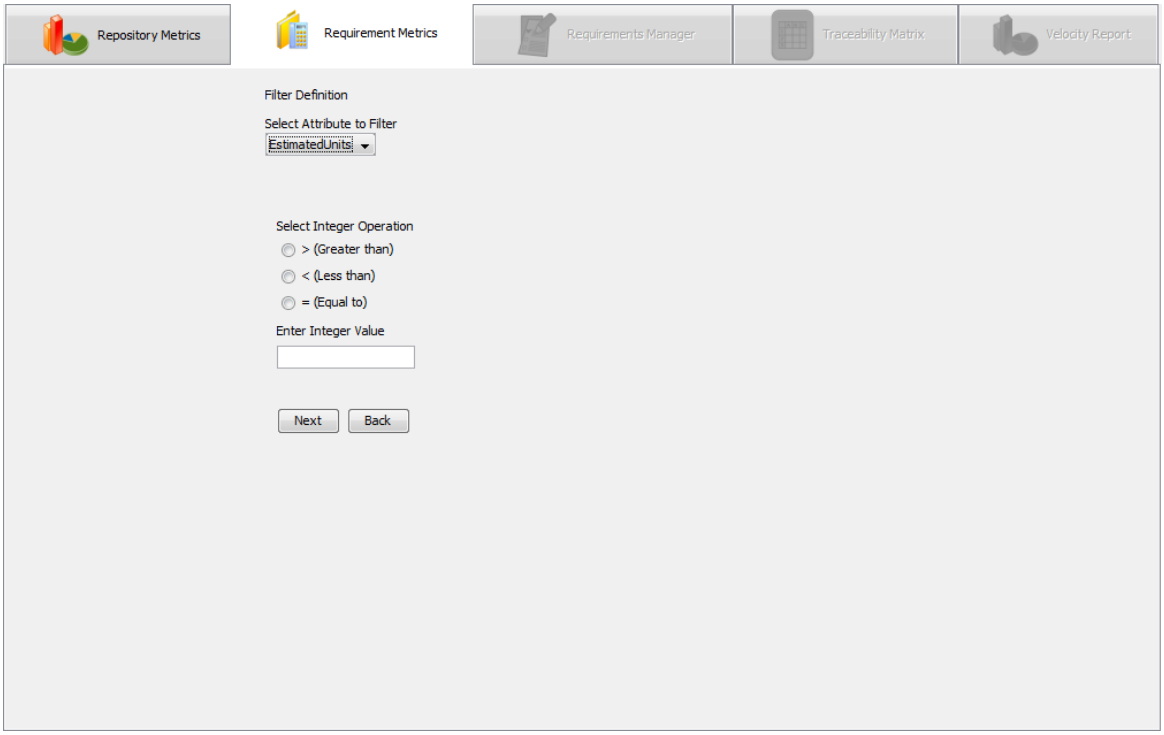

Figure 3: Filter GUI Creation

After defining all the filters, our measurement is created. The measurement can then be evaluated as the following Figure shows.

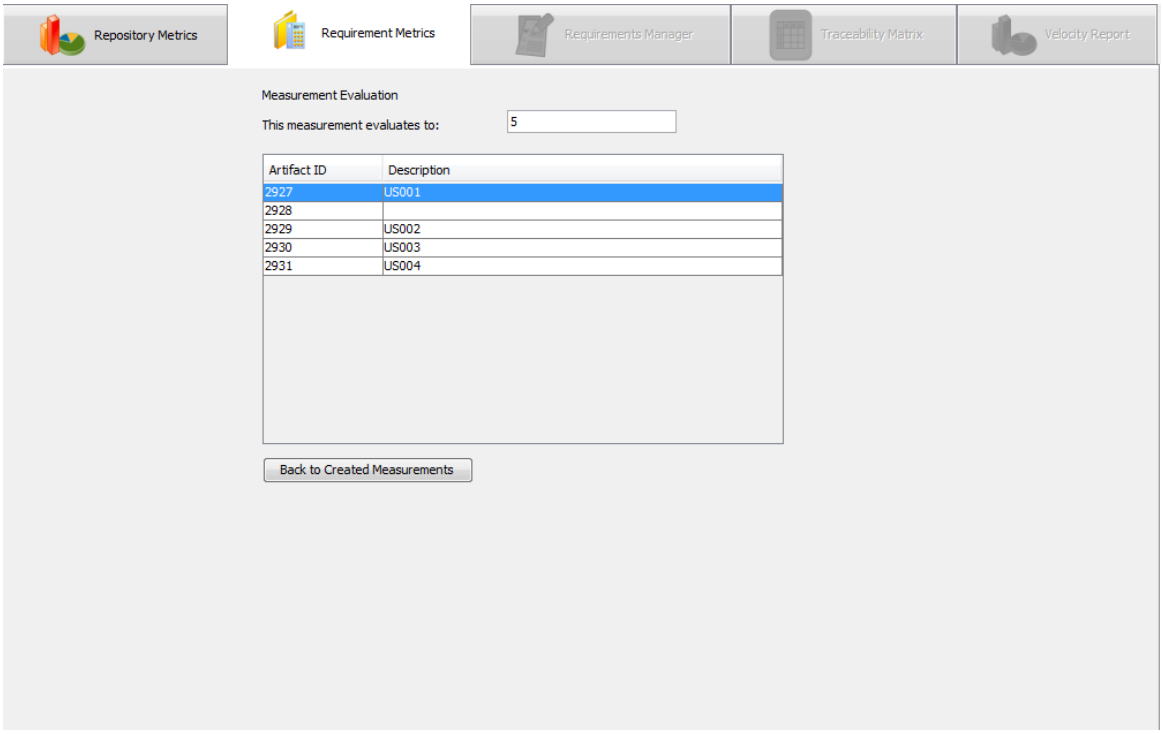

Figure 4: Measurement Evaluation

Unfortunately, for the metric that we initially define, our GUI prototype does not allow the creation of its measurement. This is because of the lack of support for complex filter creation via our GUI. We could not complete this feature because of time constraints and because we could not find much commonality between the existing artifacts. However, WPI Suite programmers can still create the required measurement using the following code.

```
// Create measurement with artifact of interest
Class artifactType = UserStory.class;
Measurement measurement = new Measurement(artifactType);

// Give it a name and description
measurement.setName("UserStory Measurement");
measurement.setDescription("All completed UserStories by either
Don or Chao in 'Iteration 1' of 'Test Project 1'.");

// Create filter: userstory.project.name  =  “Test Project 1”
IQueryHelper queryHelper = new
StringQueryHelper(StringQueryHelper.Operation.CONTAINS, "Test
Project 1");
Filter filter  = new Filter("name", queryHelper, true);
ForeignKeyHelper foreignKeyHelper = new
ForeignKeyHelper(Project.class, filter);
filter  = new Filter("projectID", foreignKeyHelper, false);
measurement.addFilter(filter);

// Create filter:  userstory.iteration.name = "Iteration 1"
queryHelper = new
StringQueryHelper(StringQueryHelper.Operation.CONTAINS,
"Iteration 1");
filter  = new Filter("name", queryHelper, true);
foreignKeyHelper = new ForeignKeyHelper(Iteration.class, filter);
filter  = new Filter("iterationID", foreignKeyHelper, false);
measurement.addFilter(filter);

// Create filter: userstory.author = Chao
CompoundFilter compoundFilter = new CompoundFilter(null, null,
true);
queryHelper = new
StringQueryHelper(StringQueryHelper.Operation.CONTAINS, "chao");
filter  = new Filter("username", queryHelper, true);
foreignKeyHelper = new ForeignKeyHelper(User.class, filter);
filter = new Filter("authorID",foreignKeyHelper, true);
compoundFilter.addFilter(filter);

// Create filter: userstory.author = Don
queryHelper = new
StringQueryHelper(StringQueryHelper.Operation.CONTAINS,
"dgaxho");
filter  = new Filter("username", queryHelper, true);
foreignKeyHelper = new ForeignKeyHelper(User.class, filter);
filter = new Filter("authorID",foreignKeyHelper, true);
compoundFilter.addFilter(filter);
measurement.addFilter(compoundFilter);
```

With the measurement implemented, the user can create the metric we defined completely within the GUI with no need for additional code. To do this, users first

provide the metric's name and description. After which, they select the previously created measurement as shown in the following Figure.

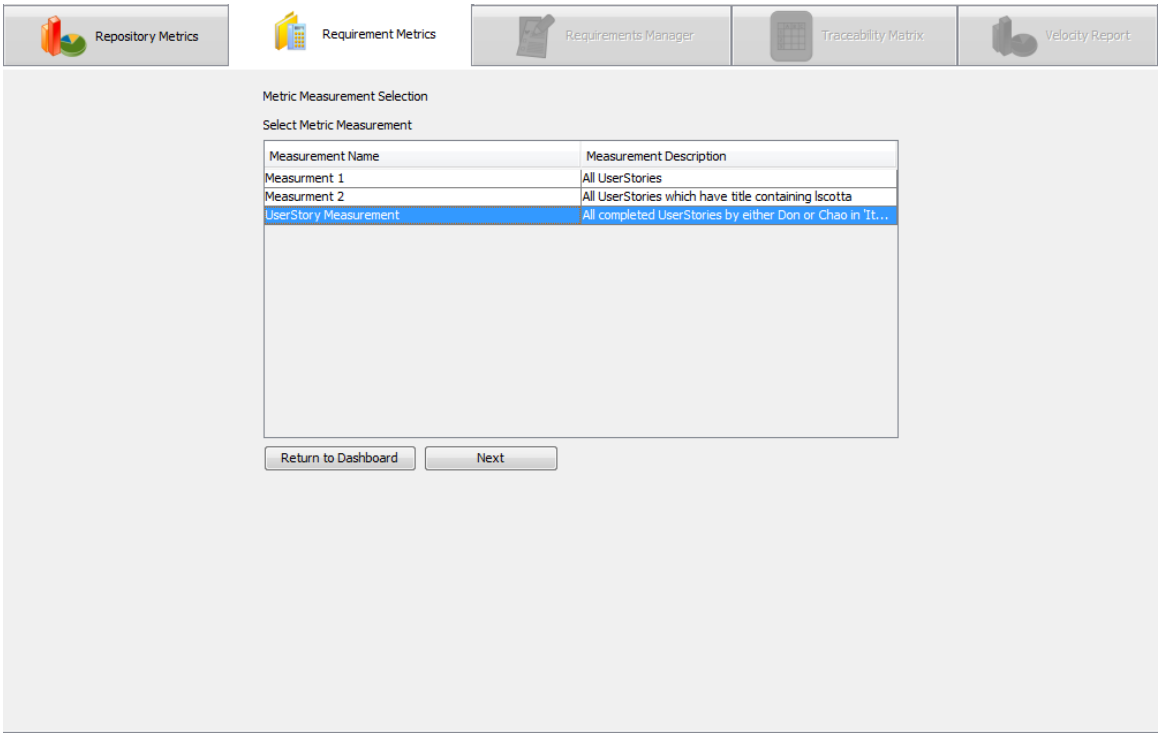

Figure 5: Measurement Selection

After this, users select the measurement attribute to distribute the metric on. This is shown in the following Figure.

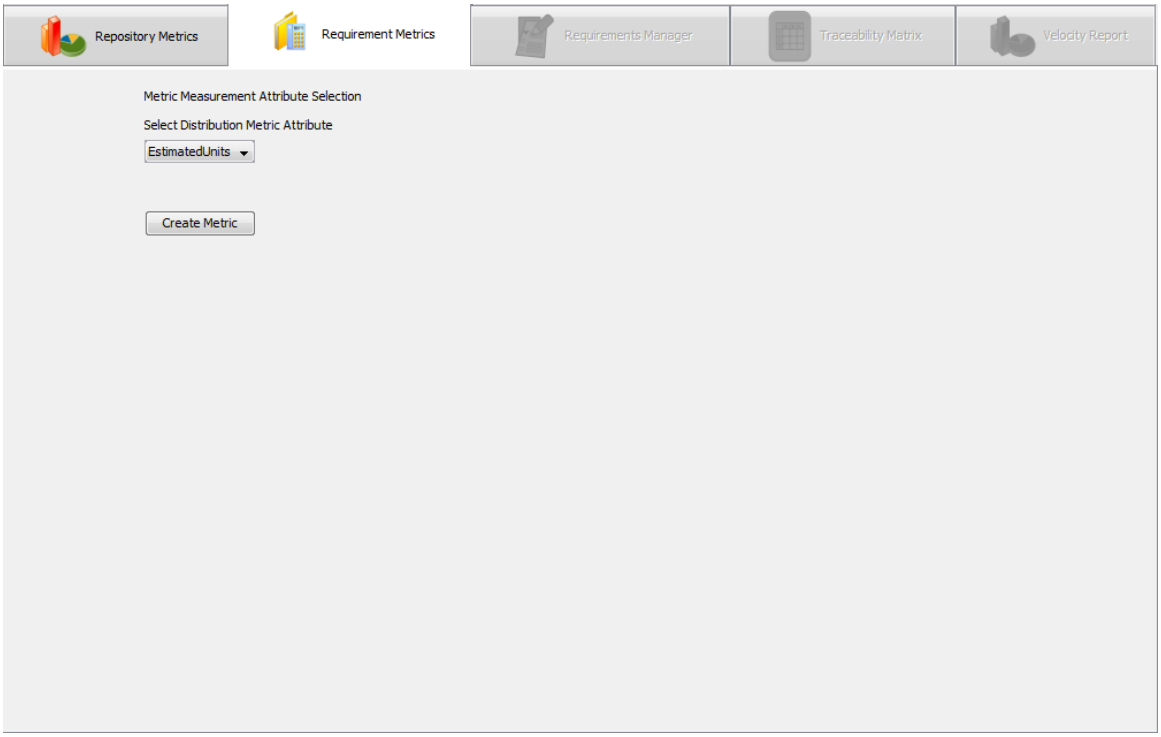

Figure 6: Select Attribute to Distribute on

Now that the metric is fully defined, it can be evaluated as shown.

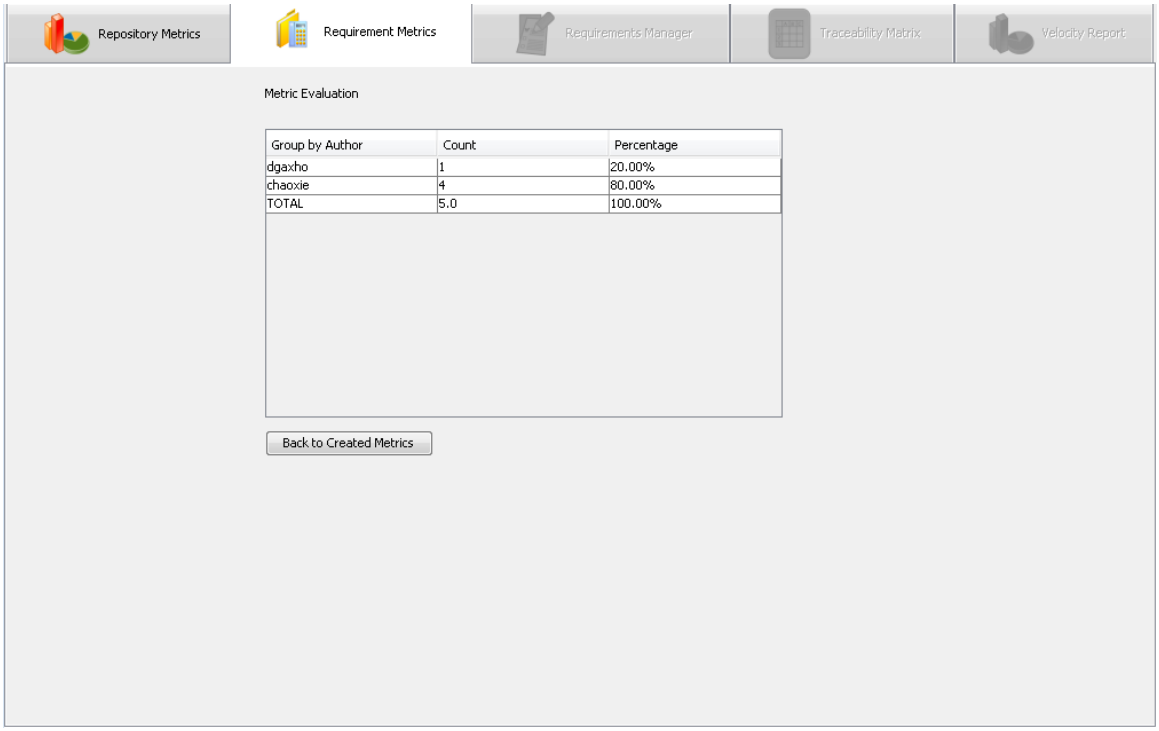

Figure 7: Metric Evaluation

5. Future Work and Conclusion

Because we spent a great deal of time on both the design of our framework, our support for both metric and measurement creation, and our GUI creation prototypes, there were some features that were moved outside the scope of this project. One feature that we hope future WPI CS students can expand on is the implementation of more built-in measurements and metrics that are useful for most WPI Suite software development teams. Our framework provides support for all requirement metrics and measurements that were presented in our background research and this is a good starting point for determining which metrics are generally useful enough to be implemented by default.

Another feature we would like to see in the future is further development of our GUI measurement creation prototype to support creation of more complex measurements. Because our focus was always on the framework design, we didn't have enough time to flesh out this portion of our GUI, but we do feel that allowing for creation of any measurement completely within the GUI would greatly increase the productivity of software development teams that seek to create requirement measurements and metrics using our framework.

An additional feature that further builds on our GUI is support for more display methods for metrics. While we provide our metric evaluation results in a table display format, the information would be much easier to interpret if our results could be passed into components that generate graphs that better represent the information. Ideally, we would like to see support for multiple graph types, including Pie, Bar Graphs and Scatter Plots.

Finally, while our project uses ActiveObjects to obtain requirement data from WPI Suite's database, it stores generated measurements and metrics completely within memory. Changing the database schema to allow for storage of measurements and metrics within the WPI Suite database would make our module much more useful, as it would allow sharing of measurements and metrics among multiple project members.

To summarize, we accomplished our goal of providing a framework that expands WPI Suite's capabilities for project health detection by allowing dynamic creation of requirement measurements and metrics. We additionally provided a GUI based method for this creation process, and while this component could greatly benefit from future development, in its current state it is still flexible enough to allow many measurements and metrics to be created without the need for additional code.

Glossary

AO – ActiveObjects ORM

Measurement – process that assigns numbers or symbols to attributes of entities in the real world according to clearly defined rules.

Metric – a measure, along with procedures to carry it out and interpreting its assessment.

ORM – Object Relational Mapping

References

Hood, C., Wiedemann, S., Fitchtinger, S., & Pautz, U. (2008). Germany: Springer-Verlag Berlin Heidelberg. Retrieved from http://books.google.com/books?id=8bAjMdORQv0C&printsec=frontcover#v=onepage&q=&f=false

Kandula, G., & Sathrasala, V. K. (2005). Product and management metrics for requirements. (Master, Umea University).

Laatukonsultointi P. Kantelinen Oy. *Requirements management tools.*, 2009, from http://www.laatuk.com/tools/reg_mgmt_tools.html

Requirement metrics. Retrieved 09/27, 2009, from http://publib.boulder.ibm.com/infocenter/reqpro/v7r1m0/index.jsp?topic=/com.ibm.reqpro.help/integ/c_req_metrics.html

Shroff, V. (2001). *Requirements management metrics*. Calgary, Alberta, Canada: University of Calgary.

Six, J. M. (2009). *Writing usability requirements and metrics.* Retrieved 09/27, 2009, from http://www.uxmatters.com/mt/archives/2009/02/writing-usability-requirements-and-metrics.php

Software metrics guide. Retrieved 09/27, 2009, from http://sunset.usc.edu/classes/cs577b_2001/metricsguide/metrics.html

Walls, C. (2009). *Modular java : Creating flexible applications with OSGi and spring*. Raleigh, N.C.: Pragmatic Bookshelf.

Wiegers, K. *10 requirements traps to avoid.* Retrieved 09/27, 2009, from http://www.processimpact.com/articles/reqtraps.pdf