This page last changed on Aug 17, 2010 by indemar.

Reviewer Payment calculator Component Requirements Specification

Scope

Overview

TopCoder attempts to improve the competition review process and is in need for updating current systems for integrating with new review process outlined in separate Competition Review Process Conceptualization document.
Currently there are two applications involved in review process. TopCoder Website application is used by the reviewers for signing up for the review positions and listing the statistics for projects and users. Online Review application is actually used for managing the review process providing the users with abilities to upload submissions for contests, fill the screening and review scorecards, submitting and resolving appeals, performing contest results aggregation, submitting final fixes and approving the results of contests.
This component will provide interfaces and implementations for calculating the review payments based on some computed statistics.

Attention

This specification was the result of an old architecture. A lot may have changed since then in the Online Review components.
Designers are required to verify that the application requirements are met by the updates and propose solutions in case they are not.
We are not looking for a word by word implementation of this specification, but for your design expertise to make this work.

Logic Requirements

ReviewPaymentCalculator

This is the main interface of this component. It takes arguments for:

- The project id
- The primary allocated payment
- The secondary reviewer payment pool
- The computed statistics

A default implementation will also be provided as detailed bellow.

Implementation

The implementation will compute the reviewer payment for each review statistics passed.
The implementation will execute the following algorithm:

- Iterate the statistics array
- If accuracy || coverage == -1, the reviewer is primary, do the following with the ReviewerPayment object to return:
    - pamentAmount = primaryPayment * primaryPaymentCoefficient (see 1.2.3)
    - reviewerId = statistics.getReviewerid()
    - projectId = projectId

- If accuracy && coverage != -1, the reviewer is secondary reviewer
    - pamentAmount = secondaryPayment * secondaryPaymentCoefficient (see 1.2.4)
    - reviewerId = statistics.getReviewerId()
    - projectId = projectId

Create exceptions as appropriate for any other cases. Extend the exception from the provided base exception.

Primary payment coefficient

The primary will get a base payment of 1.2 multiplied with the current primaryReviewer payment. This factor will already be set in the project properties so a new multiplication with 1.2 is not required.
The base payment is multiplied with the primary reviewer's timeline reliability rating.
*payment = primaryPayment * reviewerStatistics.getTimelineReliability();*
This is the total payment that the primary reviewer will receive for performing the review work.
Note that the base payment will represent the maximum possible payment that the reviewer can get. Because all the ratings the in the [0,1] interval they will actually lower, or in the best case maintain this payment.

Secondary payment coefficient

The secondary reviewers are being paid from a payment shared pool. That is the distribution is not equal as in the current review process but is performance based.
Each reviewer will receive a share of the prize pool proportional with the performance for the reviewer.
The review performance is computed in the statistics class under totalEvaluationCoefficient. This is done to accommodate new algorithms. If it is not available, (value -1) then it will be computed in line using the following default value:
*Overall Performance = Accuracy * Coverage * Timeline Reliability*
Note that all individual ratings are in the interval [0,1], their product will be in the same interval and consistent. Thus, very accurate statistics can be created based on these ratings.
If some ratings are more found to be more important than the other, appropriate weights can be introduced in the above formula to reflect this.
The secondary review prize pool is distributed as follows:

- 20% is retained as bonus which will be awarded to the reviewer having the best overall performance rating from the two secondary reviewers
- 80% will be distributed proportionally to the reviewer's overall performance ratings

Example:
Suppose two secondary reviewers have the following statistics in a competition with secondary reviewer payment pool set to $300:

Reviewer	**Accuracy**	**Coverage**	**Timeline Reliability**
A	0.95	0.55	0.9
B	0.88	0.45	1

Applying the formula: *Overall Performance = Accuracy * Coverage * Timeline Reliability*
A's overall performance is 0.95 * 0.55 * 0.9 = 0.47025
B's overall performance is 0.88 * 0.45 * 1 = 0.398
A has higher overall performance so firstly he/she gets the bonus:
Bonus = $300 * 20% = $ 60.
The rest of the prize pool (Prize pool - bonus) in this case, $240, is distributed based on the direct proportion of their overall performances so
A gets $240 * (0.47025) / (0.47025 + 0.398) = $130
B gets $240 * (0.398) / (0.47025 + 0.398) = $110.
Overall, A gets $190 from the secondary review payment pool and B gets $ 110.

Exception

Exceptions thrown from the manager should extend the exception from the interface diagram and should use BaseException defined classes.

Logging

The exceptions should be logged with ERROR level in the action classes. And user request information should be logged with DEBUG level in the action classes.

Required Algorithms

None

Example of the Software Usage

This component will provide interfaces and implementations for calculating the review payments based on some computed statistics.

Future Component Direction

None.

Enhancement policy

To avoid bloated enhancements that will slow down development without adding much value to the application, an enhancement policy will be used for this component.
Any major enhancement should be approved by the architect before being implemented. Not getting an enhancement approved is a risk of not only not getting the enhancement considered for review scoring but potentially being asked to remove it in final fix, should the given designer win the competition.
Enhancement approvals can be obtained by contacting the architect directly using the "[Send a message]" button from the member's profile page. Please note that using Contact Managers button from Online review will not also contact the architect.
Any enhancement already mentioned in future direction is considered approved.

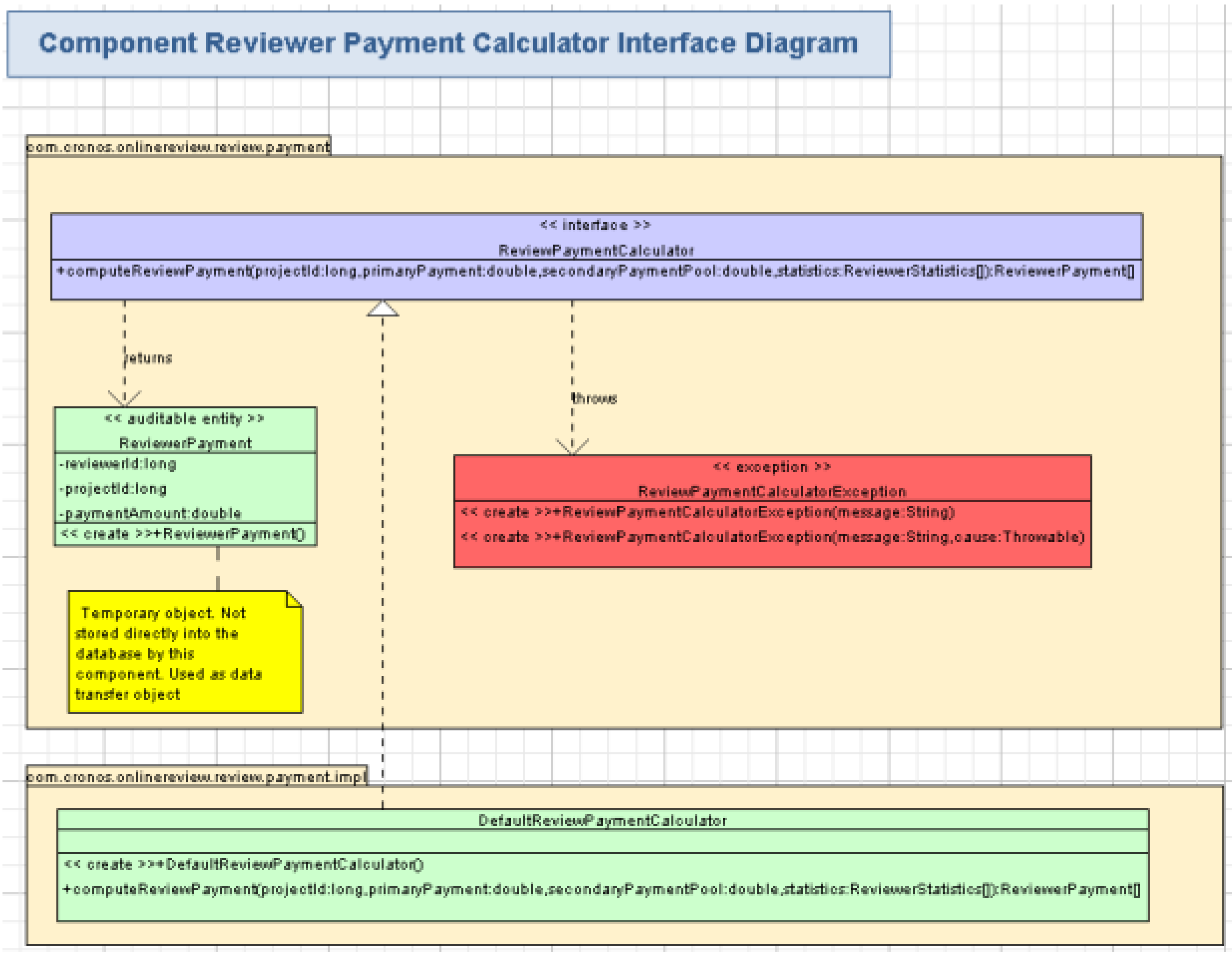
Interface Requirements

Graphical User Interface Requirements

None.

External Interfaces

Designs must adhere to the interface diagram definition found in the architecture TCUML file provided. Changes to the interfaces should be approved in the forum.


Component Reviewer Payment Calculator Interface Diagram
com.cronos.onlinereview.review.payment
<< interface >>
ReviewPaymentCalculator
+computeReviewPayment(projectId:long,primaryPayment:double,secondaryPaymentPool:double,statistics:ReviewerStatistics[]):ReviewerPayment[]
returns
throws
<< auditable entity >>
ReviewerPayment
-reviewerId:long
-projectId:long
-paymentAmount:double
<< create >>+ReviewerPayment()
Temporary object. Not stored directly into the database by this component. Used as data transfer object
<< exception >>
ReviewPaymentCalculatorException
<< create >>+ReviewPaymentCalculatorException(message:String)
<< create >>+ReviewPaymentCalculatorException(message:String,cause:Throwable)
com.cronos.onlinereview.review.payment.impl
DefaultReviewPaymentCalculator
<< create >>+DefaultReviewPaymentCalculator()
+computeReviewPayment(projectId:long,primaryPayment:double,secondaryPaymentPool:double,statistics:ReviewerStatistics[]):ReviewerPayment[]

Environment Requirements

- Development language: Java1.5
- Compile target: Java1.5

Package Structure

com.cronos.onlinereview.review.payment
com.cronos.onlinereview.review.payment.impl

Software Requirements

Administration Requirements

What elements of the application need to be configurable?

- Any manager subject to design decisions

Technical Constraints

Are there particular frameworks or standards that are required?

None

TopCoder Software Component Dependencies:

- Base Exception 2.0
- Configuration API 1.0

**Please review the TopCoder Software component catalog for existing components that can be used in the design.

Third Party Component, Library, or Product Dependencies:

None

QA Environment:

- Java 1.5
- Solaris 7
- RedHat Linux 7.1
- Windows 2000
- Windows 2003
- Oracle
- JBoss 4.2.2

Design Constraints

The component design and development solutions must adhere to the guidelines as outlined in the TopCoder Software Component Guidelines.

Required Documentation

Design Documentation

- Use-Case Diagram
- Class Diagram
- Sequence Diagram
- Component Specification

Help / User Documentation

- Design documents must clearly define intended component usage in the 'Documentation' tab of the TopCoder UML Tool.