Overview

This is the third in a series of articles on how Rational Collaborative Lifecycle Management (CLM) solutions support software development compliance.

Segregation of duties is a key principle in protecting a system from unauthorized changes.

According to Wikipedia, “Separation of duties (SoD) is the concept of having more than one person required to complete a task. In business the separation by sharing of more than one individual in one single task shall prevent from fraud and error. The concept is alternatively called segregation of duties or, in the political realm, separation of powers.”

In this article we’ll look at three different ways that Segregation of Duties can be implemented in CLM:

- Using roles and permissions to support segregation of duties is the easiest way to support this principle. It is effective for segregation of duties rules that map easily to the roles and permissions that are configurable in Team Concert.

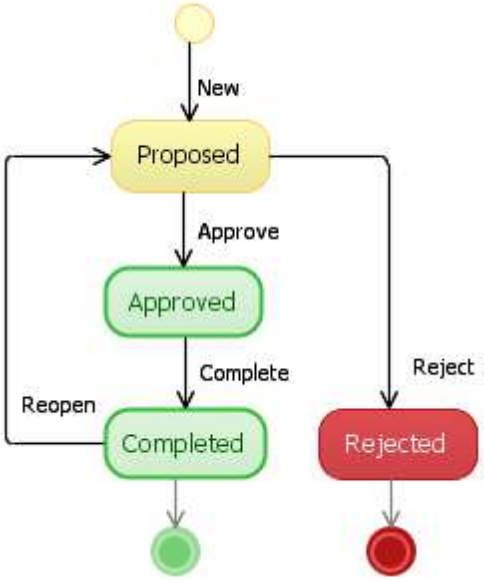
- Some segregation of duties rules are not easily configured with permissions because they involve multiple roles and a specific process configuration. Violations of these rules can be reported through generation of an audit report. Configuring an audit report template is a bit harder than simply setting up permissions, but our examples help you get started.

- If you want to prevent violation of the rules that cannot be easily configured, the extensibility features of the CLM tools allow you to do that. Our example shows you what is possible, albeit a more advance implementation.

Using Roles and Permissions to Support Segregation of Duties

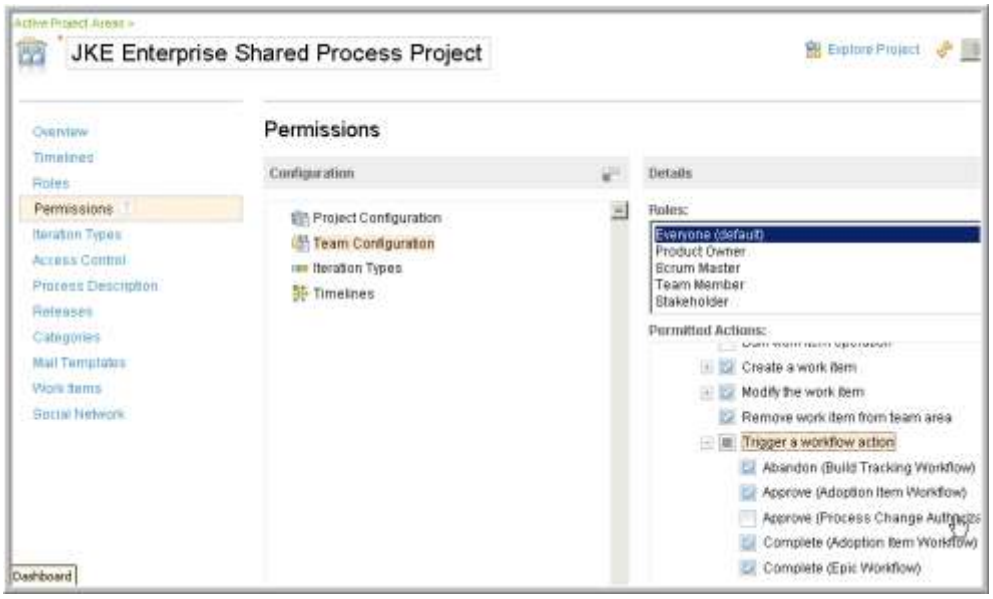
In this example we have established roles and permissions set up in a Process Engineering project such that the approval of a process change is restricted to a specific role. There are many places in the CLM solutions where permissions can be assigned based on roles: permissions at the project area level, team area level, by iteration, on operation behavior, and in workflow permissions (as we are showing here). It is critical in a regulated environment to understand which users have which permissions at a given point in time (e.g. for a build or release). For more detail on how permissions are processed see these articles: https://jazz.net/library/article/215, https://jazz.net/library/video/106

We started with the Scrum process template and added work item type called “Process Change Authorization(PCA)”. The PCA was created from the Adoption item work item type and has a simple state model:



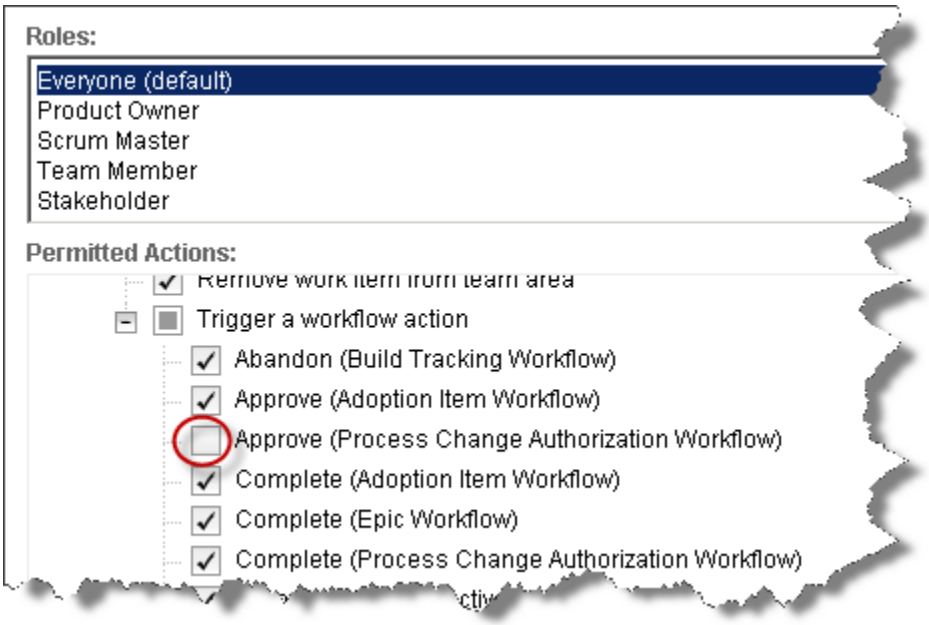
We also have established a role called “Change Control Board”. Only users with this role assigned can approve or reject a PCA.

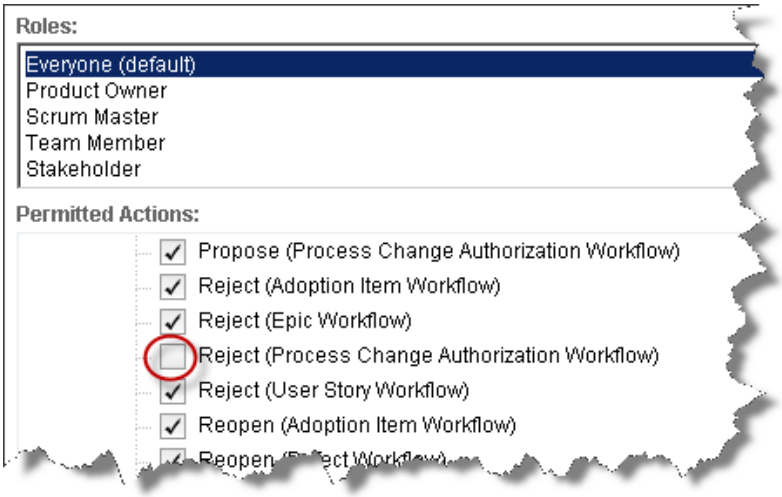
To establish this permission, we have two steps – first, ensure that not Everyone can perform the state transition and then ensure that only the Change Control Board role can perform the state transition. This configuration is done in the Team Configuration of the Permissions for a Project Area.



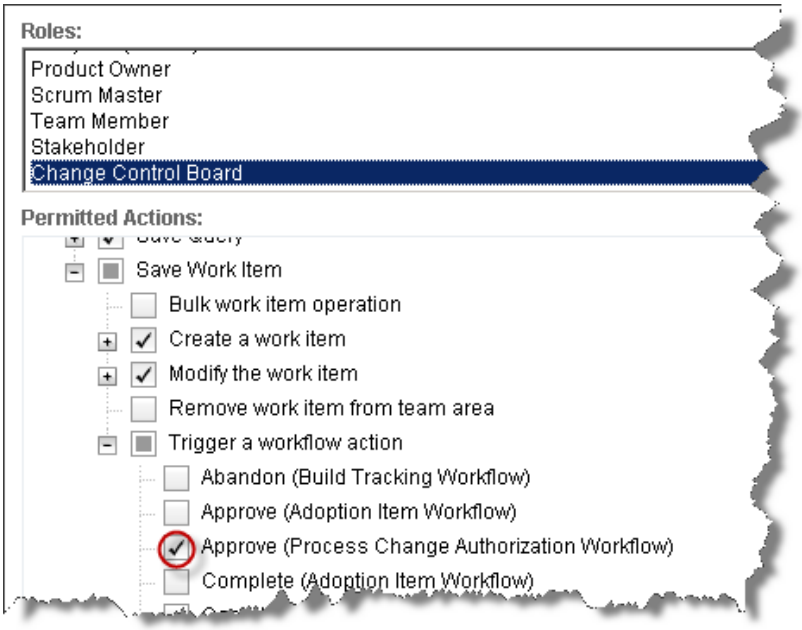
Here we show the **Permissions** tab and **Team Configuration** and highlight the permissions set for **Everyone**.

We scroll down to see that the **Work Items/Save work item/Trigger a workflow** action permissions restrict Everyone from approving or rejecting a Process Change Authorization.





When we select the **Change Control Board** role and ensure that this role has permission to both approve and reject Process Change Authorizations. Note that the Change Control Board inherits permissions from the Everyone role, so it is only necessary to specify the permissions that are different.

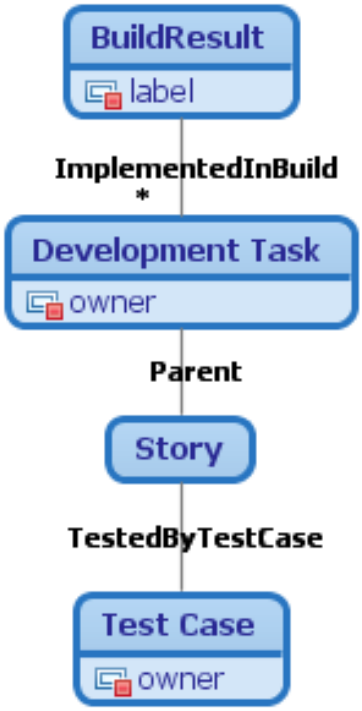


Now when someone who is not in a Change Control Board role attempts to Approve or Reject a PCA, the “Permission Denied” message is displayed:

Reporting on Violations of Segregation of Duties for Testing and Implementation

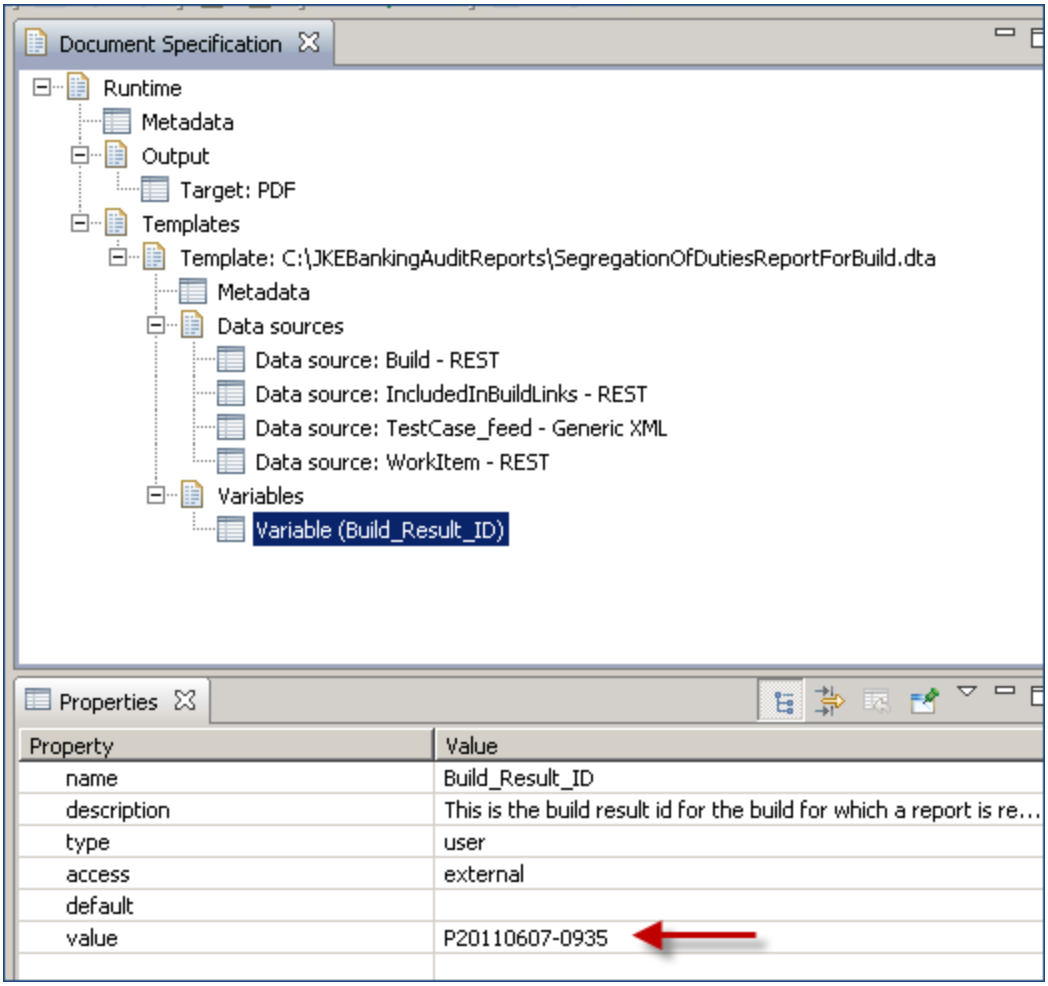
In this example, we generate a report on a Build that flags any violations of the segregation of duties between development and testing. It will also report a quality violation for any task included in the build for which a corresponding test case is not found. This is a custom report that was built using Rational Publishing Engine and the reportable REST APIs exposed by Rational Team Concert and Rational Quality Manager.

These are the records and their relationships used in the Money That Matters sample application that are relevant for this example:

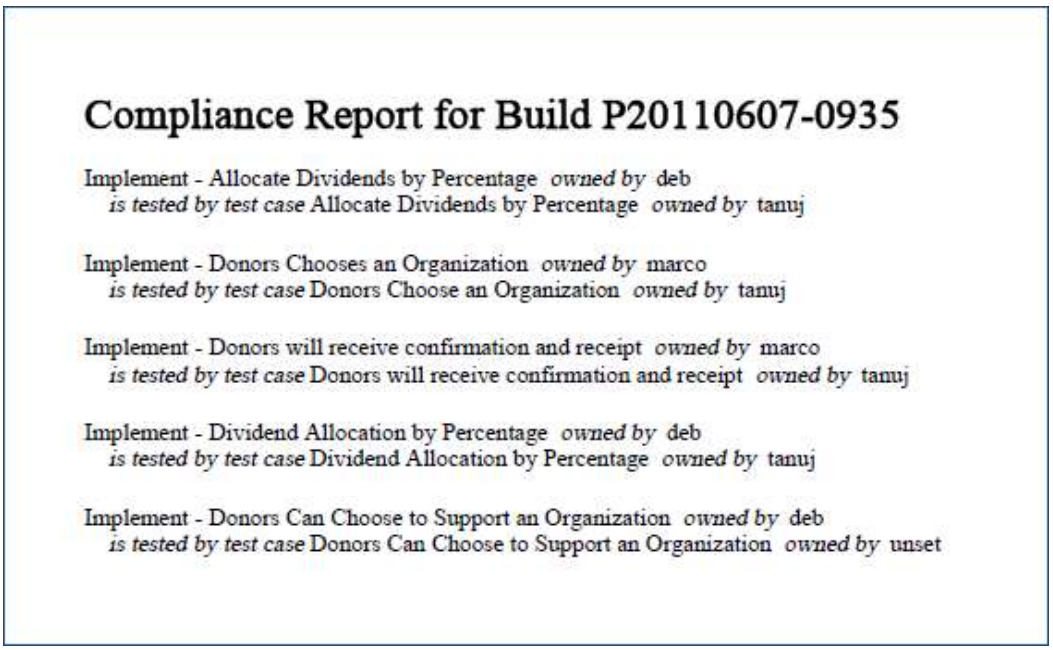


Here is the Build Result in the example that we will report against:

The document specification for the report uses an input variable that defines the Build Label of the build result on which we want to report. This value can be changed to make the report against any build result desired.

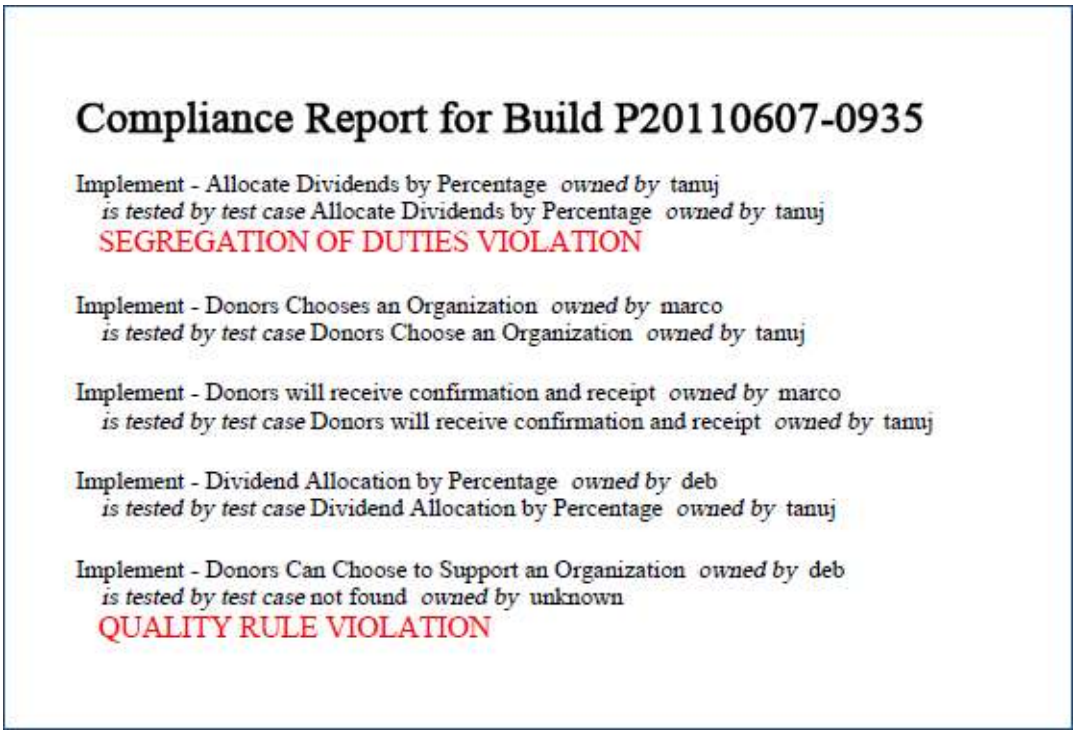


The published document from the Money That Matters example looks like this:



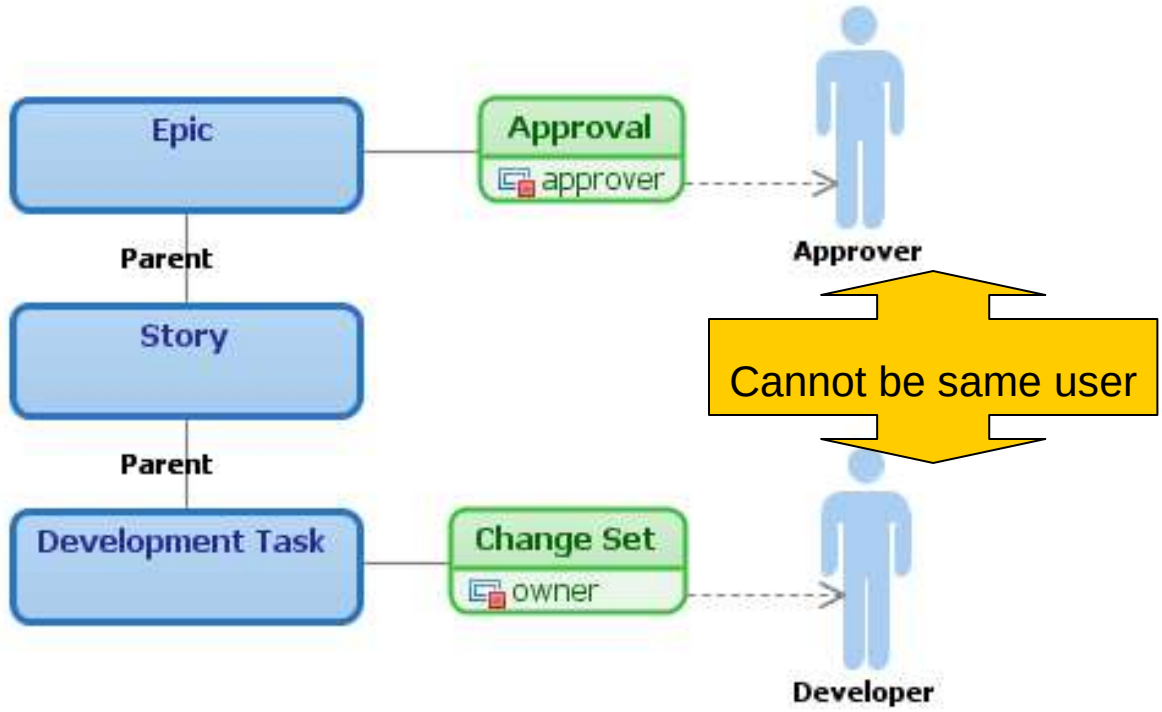
Although the report looks a bit suspect because the test case name and implementation work item name are almost exactly alike, this is correct. The naming conventions used in the Money That Matters sample application are designed to make the relationships between elements easily understood.

Now if we tweak the example data a bit we can cause some violations, as reported when we run the report again:

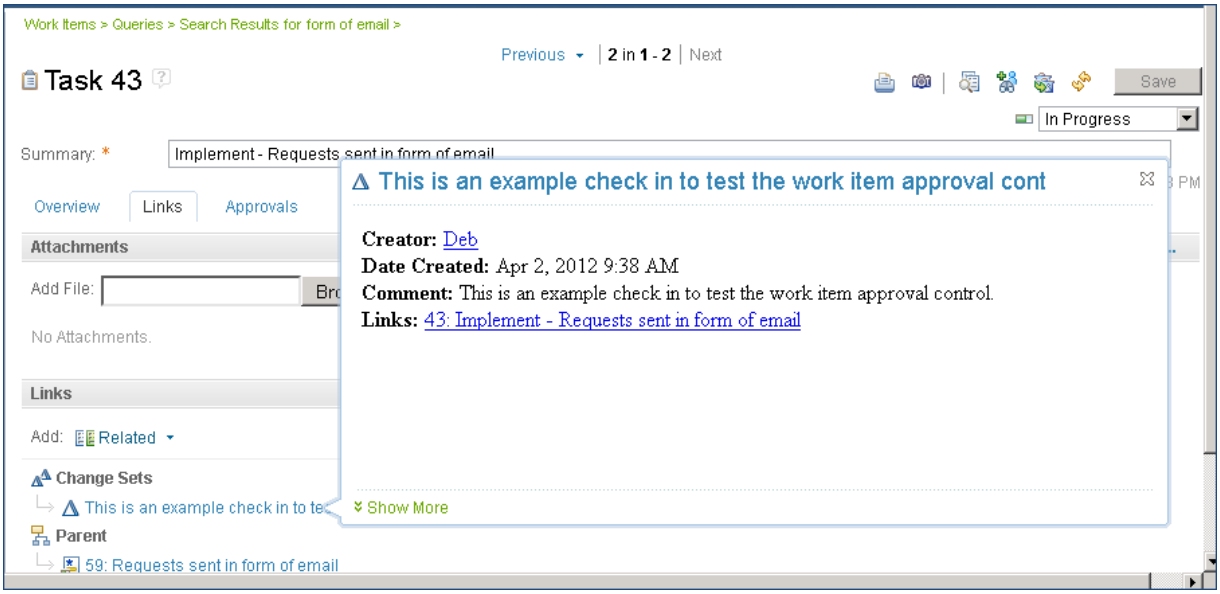


Using Automated Enforcement of Segregation of Duties for Change Sets and Approvals

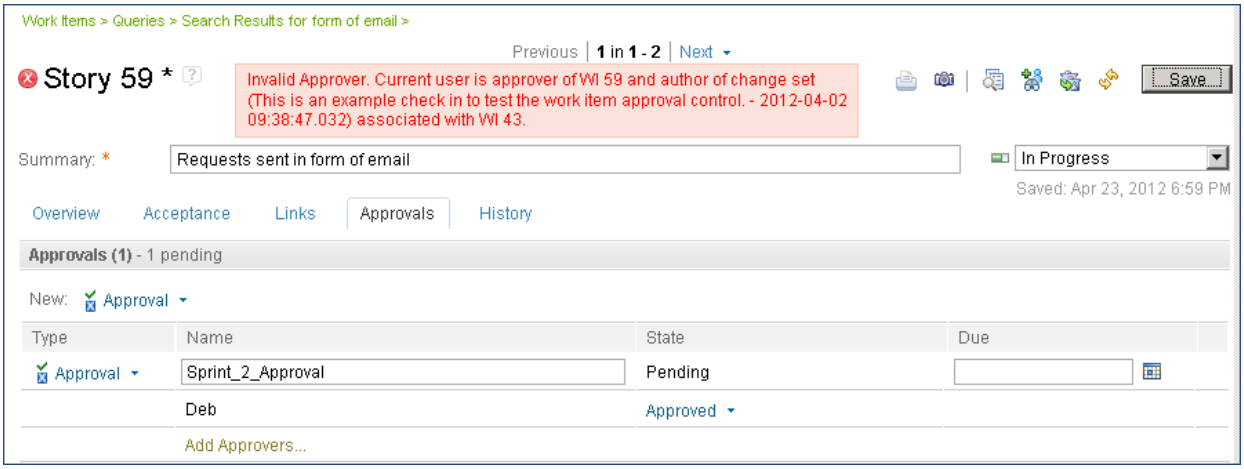
In this example, we use custom extensions built for Rational Team Concert that prevent violation of the segregation of duties rules for implementation and approvals. In the first scenario, a user is not allowed to save a work item with a change set where she is the change set owner if she is also the approver of any record in the parent hierarchy of the work item. In the second scenario, a user is not allowed to save a work item where he is an approver if he is the owner of a change set on any record in the child hierarchy of the work item he is approving. In this way, the system prevents violation of a segregation of duties between implementers and approvers. Here is a graphical example:



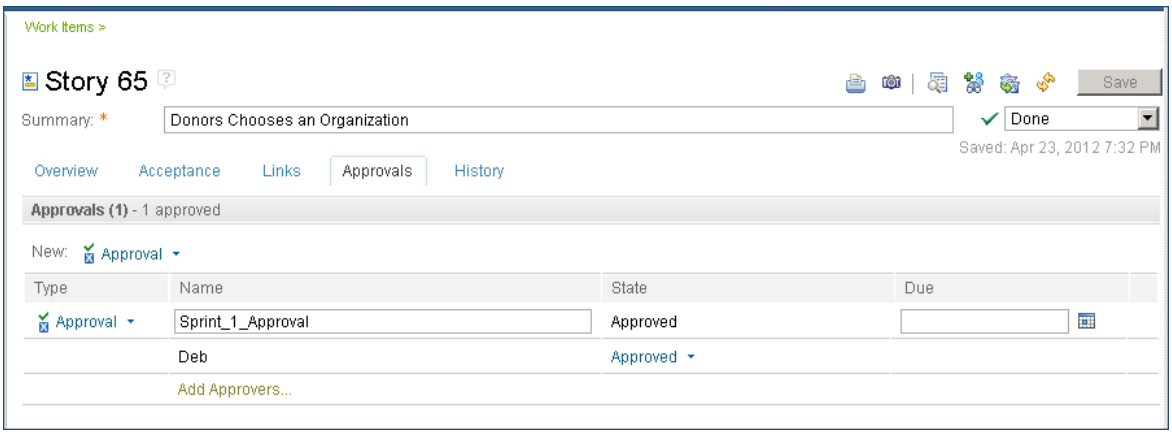
In the Money That Matters example, we examine *Task43: Requests sent in form of email.* We see that the change sets associated with the work item are all owned by *Deb*.



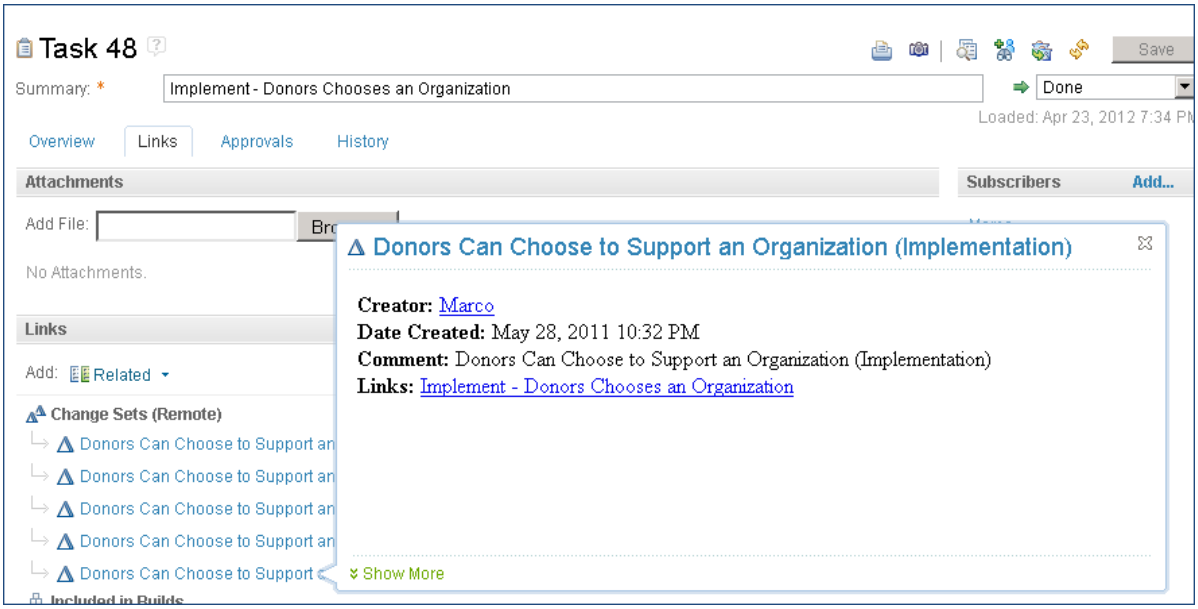
Now we navigate to the Parent *Story59: Requests sent in form of email* and see that Deb is also listed as an approver. When Deb sets the Approval to **Approved** and tries to **Save** the story, this is the result:



Now we will try to contribute work for a Story that we have previously approved. On *Story 65: Donors Chooses and Organization*, we see that Deb has previously provided an approval.

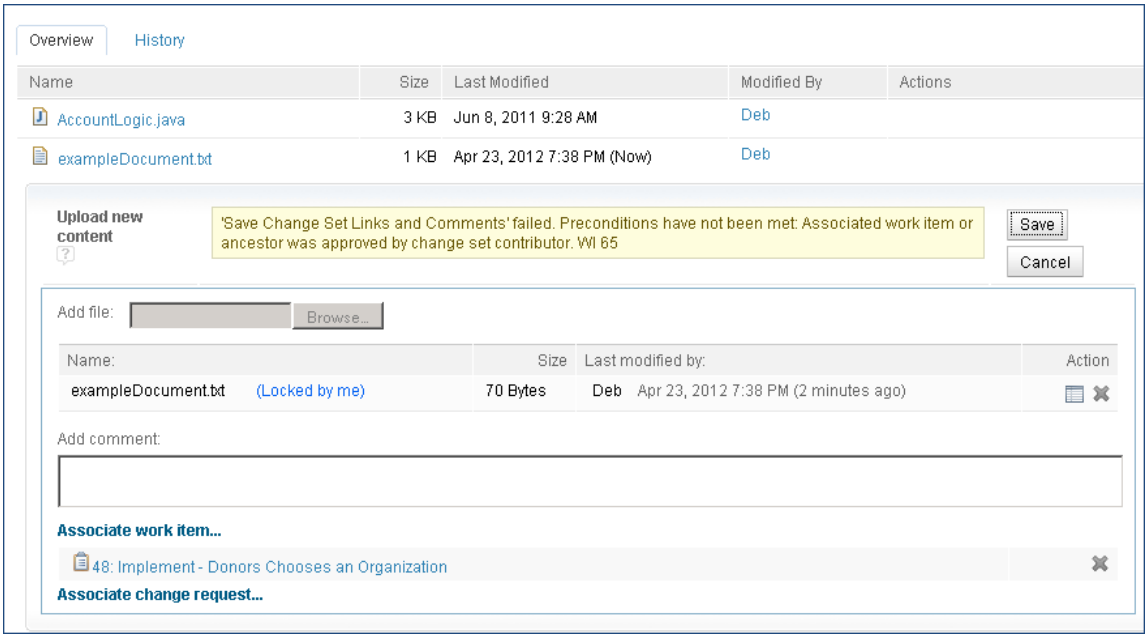


We navigate to the child *Task48:* Implement – *Donors Can Choose to Support an Organization*



Notice on the **Links** tab that all of the change sets on this work item so far are owned by *Marco*.

Logged in as Deb, we are in source control and upload a new file and **Associate work item** *Task48: Implement – Donors Chooses an Organization.* When we save the upload, we get this message.



Creating Custom Operation Advisers is an advanced extensibility feature of CLM. For more information see https://jazz.net/library/article/784

This example was built by the Rational Lab Services Integration Engineering Team.

Summary

In this article we looked at three ways to manage segregation of duties rules. With roles and permissions, privileges can be granted to users based on what the process will allow them to do. When the segregation of duties rules are more complex than what the roles and permissions support, there are two other options that we showed. With audit reports, you can use the document generation capabilities with Reportable REST APIs to generate a report of rules violations. With Team Concert custom operation advisors, you can create custom extensions that prevent actions that violate segregation of duties rules.

The options you choose to implement will be determined by your specific internal controls that pertain to segregation of duties as well as your project and roles structure.