RepoGuard: A Framework for Integration of Development Tools with Source Code Repositories

Malte Legenhausen, Stefan Pielicke, Jens Rühmkorf, Heinrich Wendel, Andreas Schreiber
Simulation and Software Technology
Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR)
Linder Höhe, 51147 Cologne, Germany
{malte.legenhausen,stefan.pielicke, jens.ruehmkorf,heinrich.wendel,andreas.schreiber}@dlr.de

Abstract

Today modern software development is not possible without the aid of tools like version control systems, bug tracking systems or instruments that ensure the compliance with code conventions. Unfortunately, all of these tools "live in their own world", are only loosely coupled and do not interact with each other. RepoGuard addresses this problem by linking version control systems to other software development tools. It is implemented as an extension to several version control systems and provides interfaces to integrate other tools. The use of RepoGuard allows maximum control and validation of all committed resources before they are permanently stored. Additionally, RepoGuard provides communication channels in order to inform all relevant stakeholders about the failure or success of the process. Overall, RepoGuard provides simple but effective means to guarantee software quality standards in distributed development processes.

1 Introduction

Software development processes utilize a wide range of different tools. The most important part of the tool infrastructure is a version control system. It contains most of the project related resources such as the source code for different programming languages (e.g., C, Java, or Python), documentation and configuration files, or build scripts. Furthermore the following tools are usually found in such an infrastructure: Code checkers ensure the adherence to code conventions by static or dynamic analysis. Build systems arrange processes for the build automation of frequently performed tasks such as compiling code, running tests and deploying files. Bug tracking systems allow the monitoring of reported software bugs during the development process.

Typically, all of these tools are loosely coupled. Thus, problems soon occur if the validation of coding conventions solely depends on every developer. Only a slight mistake or a difference in the configuration of each developer's tool might lower the overall quality of the source code. Another problem is the management overhead which results from having to provide the same information repeatedly to different tools. A common example is when a developer makes changes to the source code. He again has to add the changes made to corresponding issues in the bug tracking system, additionally he might need to inform other developers about them.

Usually there is no mechanism for automatic validation, logging, and notification. Here RepoGuard acts as a bridge between all these tools. It decreases the management overhead and avoids common pitfalls in the development process. Thus it significantly helps increasing the software quality.

2 Basic Concept

RepoGuard is an advanced validation framework with built-in integrations for several common version control systems. The integration is carried out by utilizing the hook mechanisms each version control system provides. The user may provide configurations which are processed through inversion of control mechanisms. RepoGuard is completely written in the Python programming language which allows for easy integration of other tools. An extensible command line tool for advanced usage is provided which allows for comfortable administration.

RepoGuard is based on two types of constituent parts: *Checks* and *Handlers*. These parts are configured by the user to specify what to do when processing a transaction of changes to the version control system. Checks are responsible for validating the content of a transaction. Handlers are responsible for handling the results and/or the output of Checks. The complete architecture of RepoGuard is shown

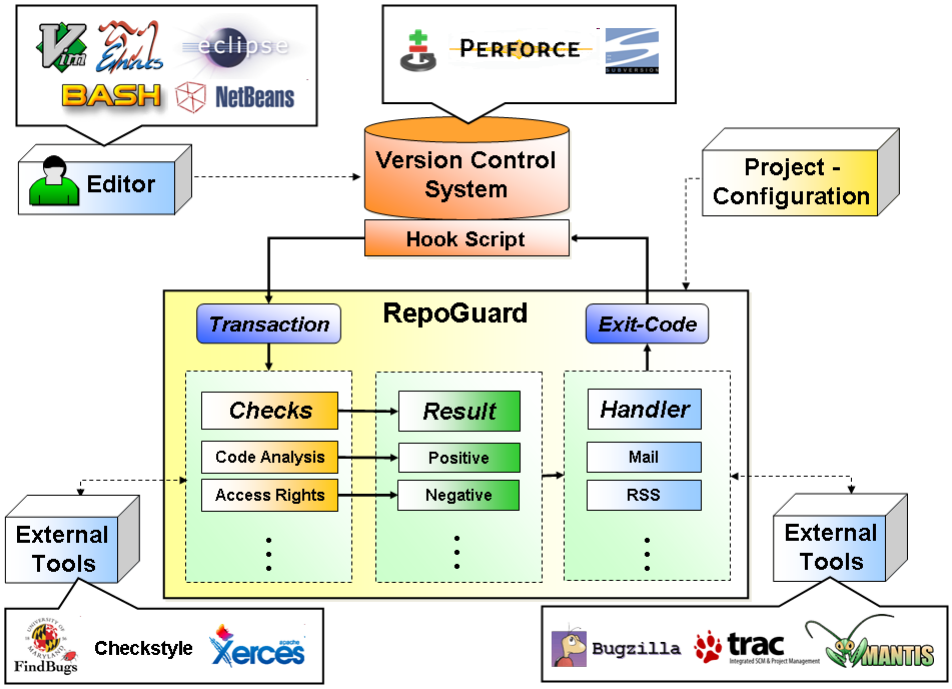

Figure 1. Architectural Overview of RepoGuard

in Figure 1. There, the developper interacts with the version control system either through an IDE (such as Eclipse or NetBeans) or performs a commit directly through the command line or other tools. Then, on the server-side where the version control repositories reside, checks validate the content of that very transaction. Upon success, an e-mail might be sent, the corresponding bug entry is updated to contain the commit message as well as the contents of the commit etc. Similar actions may take place when a check fails.

Extensibility is supported via an easy to use plug-in engine and a flexible configuration mechanism for Check and Handler execution. The plug-in engine offers the possibility to integrate self written Checks and Handlers, to integrate modules that provide the access to other version control systems or bug tracker, and to extend the administration command line tool. The following version control systems are currently supported: Subversion [6], Git [3], and Perforce [5].

2.1 Checks as Control Units

Checks are the control units of RepoGuard. Although they can be configured to act only as warnings they usually inspect a transaction and either approve or reject it. Every aspect of a transaction can be examined, in particular all contained files and their properties. Additionally the current repository state can be taken into account. Is there an open bug tracking id for this check-in? Does the developer have access rights to check-in those files? Is there a unit test for the source files? Are all XML files standard compliant? These are examples for tests a Check can perform. Besides these validations a Check can perform any specified action which is required before or after a check-in.

Each Check has to return a state, either success or failure, and may also include a description message for the state. This message can contain information regarding the execution of the Check, like reasons for a failure or advanced log information, and will later be processed by a Handler.

RepoGuard contains several built-in Checks: these include checking Java code through Checkstyle [2], checking Python code by use of Pylint [8], making sure keywords are contained in each file, bug tracking ids are provided in the commit message, verifying access rights and more.

2.2 Handlers as Report Units

Depending on the configuration Handlers can be executed after each Check or after running all Checks. A Handler is a script that is able to process the incoming Check results and messages. It could convert the information into an appropriate format, which includes information about the whole change set (for example, the date or author), or may provide links to the browser interface of the repository.

After processing all Check's results a Handler sends the

information as an e-mail, stores it in a log file, updates an RSS feed, attaches them to a bug id of the bug tracker or simply produces an output to the console of the developer. Integration of any software system in order to process the information is possible.

2.3 Bug Tracking System Integration

One of the main goals of RepoGuard is the integration of version control systems and bug tracking systems via a combination of Checks and Handlers. For this purpose, a "bug tracking" Check analyses whether every commit message contains a *valid* bug id as a reference. An existing bug id is considered valid, if it is assigned to the committing developer and if its state is set to *inprogress*.

After the Check has successfully run a Handler appends the resulting information (commit message, changed files, etc.) as note to the bug report into the bug tracking system. Out of the box support for Mantis [4], Bugzilla [1], and Trac [7] is included.

3 Workflows

The workflow component of RepoGuard allows the adjustment of the execution order of Checks and Handlers to the current project requirements. The order of Check executions is defined by a queue. Handler executions are organized in queues as well but their order is irrelevant in theory. At runtime each Check in the queue will be processed sequently. Their return values, consisting of a state and a message, will be combined to a final state. This state is used to decide which Handlers have to be executed to create a process summary. If the final state indicates a success the configured Success-Handlers will be executed, otherwise the Error-Handlers. This final result of the workflow is used to decide whether the transaction will be accepted or rejected.

Usually, workflow execution is aborted immediately when a Check fails, but you have the option to configure this in a more flexible way. The result of a failed Check can be transformed into the states *warning*, *delayonerror* or *abortonerror*. When a result state is transformed into warning the check execution is always interpreted as success and will not influence the final result. If a state is transformed into *delayonerror* the result of the Check is interpreted as error but the workflow execution will continue and only fail at the end. The default value of *abortonerror* means that the execution will be stopped if the Check returns an error.

RepoGuard can execute multiple workflows on a single transaction. This is useful if you commit files which belong to different projects in your repository, each project needs a separate configuration. When multiple workflows are executed the final state will be determined by combining all single states with an AND operation. If one workflow fails

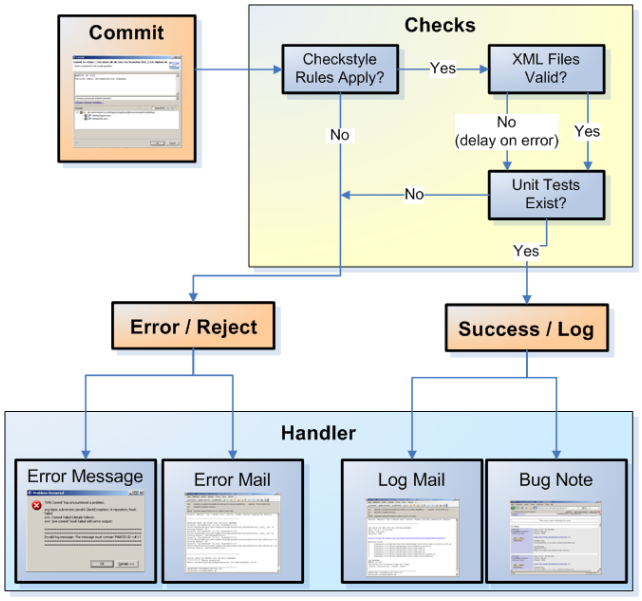

Figure 2. A Typical Workflow in RepoGuard

the complete transaction will be rejected. A visualization of a typical workflow is shown in Figure 2. Upon commit, several checks on the server-side validate desired properties (Such as: Does the static code analysis succeed? Are XML files valid? Do proper unit tests exist?) Upon failure of these checks (i.e., at least one error occured and was considered to be severe) the commit aborts with an error message that is displayed to the developer. In this example, an additional error notification is being sent via e-mail. Another typical reason for failure is the absence of a corresponding bug-id in the commit message (when no bug-id is provided, the commit message and revision cannot be linked to the bug tracking system). Upon succes, a success mail is being sent to interested parties. Also a note is appended to the corresponding bug-entry of the bug tracking system that contains details of the commit (author, commit message, files contained) as well as a link to the corresponding revision in the version control system.

3.1 Configuration of Workflows

A configuration file is used to define workflows, the setup of Checks and Handlers, and parts specific to the structure of the repository called profiles.

A repository can be divided into different profiles and for each profile a separate workflow can be defined. The selection of profiles is determined by the paths of the files included in the transaction which are defined as regular expression patterns. If a file included in a transaction matches the pattern of a profile this file is processed by the corresponding workflow. Thus in one commit RepoGuard may

even execute multiple profiles.

The configuration of Checks and Handlers is specified in a special section, outside the profile definitions. This makes the reuse of those configurations in several profiles possible. Examples for configuration values in this section are paths to binaries of external tools or the SMTP server used to send e-mails.

To reduce redundancy between several configuration files of different repositories RepoGuard provides an inheritance mechanism. You can provide template configuration files which configurations of individual repositories may extend from.

RepoGuard also provides a validation mechanism for the configuration files in order to avoid mistakes that can cause a halt of the complete check-in process or, even worse, lead to an unintentional repository state.

4 Conclusions

RepoGuard has been created to improve the quality of source code in the various repositories of the German Aerospace Center (DLR). Having started as a simple validation framework for Subversion, it has now matured to be utilized in many of DLR's internal and external development projects. In particular, this has proven to be successful for DLR-internal development projects. These are typically highly distributed due to DLR's 13 sites within Germany. When introducing RepoGuard, one initially observes a high number of rejected commits which is a good indicator for the success of the tool. Also, in the beginning developer acceptance might be low but increases by and by. This makes the exploration of more advanced features like configuration file management in your repository possible. Despite very varying requirements in different projects, the flexibility of the configuration and the good extensibility due to the usage of Python allows the widespread introduction of RepoGuard into development processes.

References

[1] Bugzilla. http://www.bugzilla.org.

[2] Checkstyle. http://checkstyle.sourceforge.net.

[3] Git – fast version control system. http://git-scm.com.

[4] Mantis bug tracker. http://www.mantisbt.org.

[5] Perforce software - the fast software configuration management system. http://www.perforce.com.

[6] Subversion website. http://subversion.tigris.org.

[7] The trac project. http://trac.edgewall.org.

[8] Logilab. pylint. http://www.logilab.org/pylint.