Abstract

Modern software is built by teams of developers that work in a collaborative environment. The goal of this kind of development is that multiple developers can work in parallel. They can alter a set of shared artifacts and inspect and integrate the source code changes of other developers. For example, bug fixes, enhancements, new features or adaptations due to changing environment might be integrated into the system release.

At a technical level, a collaborative development process is supported by version control systems. Since these version control systems allow developers to work in their own branch, merging and integration have become an integral part of the development process. These systems use automatic and advanced merging techniques to help developers to merge their modifications in the development repositories. However, these techniques do not guarantee to have a functional system.

While the use of branching in the development process offers numerous advantages, the activity of merging and integrating changes is hampered by the lack of comprehensive support to assist developers in these activities. For example, the integration of changes can have an unexpected impact on the design or behavior of the system, leading to the introduction of subtle bugs. Furthermore, developers are not supported when integrating changes across branches (cherry picking), when dealing with branches that have diverged, when finding the dependencies between changes, or when assessing the potential impact of changes.

In this dissertation we present an approach that aims at alleviating these problems by providing developers and, more precisely, integrators with semi-automated support for assisted integration within a branch and across branches. We focus on helping integrators with their information needs when understanding and integrating changes by means of characterizations of changes and streams of changes (*i.e.,* sequence of successive changes within a branch) together with their dependencies.

These characterizations rely on the first-class representation of systems' histories and changes based on program entities and their relationships rather than on files and text. For this, we provide a family of meta-models (*Ring*, *RingH*, *RingS* and *RingC*) that offer us the representation of program entities, systems' histories, changes and their dependencies, along with analyses for version comparison, and change and dependency identification. Instances of these meta-models are then used by our proposed tool support to enable integrators to analyze the characterizations and changes. *Torch*, a visual tool, and *JET*, a set of tools, actually provide the information needs to assist integration within a branch and across branches by means of the characterization of changes and streams of changes respectively.

Keywords: object-oriented programming; meta-models; history and version of programs; visualization; semantic merging; program analyses