Chair of
Software Engineering

Refinements and Git Integration with Notifications and Monitoring

Software Engineering Laboratory: Open Source Eiffel Studio

By: Christopher Dentel
Supervised by: Christian Estler
Dr. Martin Nordio
Prof. Dr. Bertrand Meyer

Student Number: 11-909-868

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

inf | Informatik
Computer Science

ABSTRACT

In this report, refinements to improve the work developed in the previous two reports (*Monitors: Keeping Informed on Code Changes* and *News and Notification: Propagating Relevant Changes to Developers*) were developed and presented. These refinements included usability, efficiency, and new feature requests. Notifications were implemented into another student's project, showcasing the usability of the developed system. The monitor system developed was updated to support a new version of configuration management. Blame was added to the monitor diff view, allowing developers to see who last committed each line of code, and several other enhancements were made to make the monitor system more customizable. GUI enhancements were made to allow users to customize their workspace, and the system was made more flexible by allowing users to change their monitoring preferences after creation. The monitoring system also now will preferentially display violated monitors upon initialization.

Table of Contents

Cloudstudio

Motivation

For most large projects and teams, using an Integrated Development Environment (IDE) is a necessity of software development. But traditional desktop software IDEs do not seek to accommodate teams who are spread out across multiple countries, time zones, languages, and cultures. These are the challenges faced by those organizations which pursue global software development (GSD). The challenges that GSD poses are not necessarily new, and approaches to alleviate the difficulties that come with GSD have been previously investigated with many different approaches. Some investigations have focused on comparing different project management approaches, including agile vs. structured development [1]. Carmel and Agarwal [2] investigated means to reducing the "distance" between teams (national, organizational, cultural, and temporal distances) and reducing collaboration. Several other investigations have not sought to avoid collaboration but instead focused on how to better facilitate collaboration across time-zones [3] [4] [5]. One such study [6] utilized the Distributed and Outsourced Software Engineering course (DOSE), [7] [8] using some of the technologies presented in this paper.

Existing IDEs rely on configuration management that leads to disparities in information awareness, such as discovering at commit that two major refactorings have occurred simultaneously. While this style of configuration management seeks to isolate developers from the changes that other developers are making, Cloudstudio[a] seeks to share information in real-time. As online document collaboration websites are eliminating the need to email documents of varying revisions back and forth, Cloudstudio seeks to allow developers to work simultaneously on a project, sharing information between themselves as they like, while also maintaining the isolation that traditional configuration management affords.

Features

Cloudstudio [9] is a web based IDE, allowing developers to access their projects at any time from any machine. This move to the web eliminates the need to maintain and update different versions of software on local machines, while also allowing developers to work where they want, when they want. But Cloudstudio is not just a web-app clone of an existing IDE. Cloudstudio seeks specifically to meet the needs of developing in distributed environments through smarter configuration management and tool integration. While still only a web-app, Cloudstudio integrates development tools, collaboration tools, and verification tools. Some such tools are listed below.

Development:

- *Languages* – Cloudstudio supports projects in Eiffel, Java, C#, and JavaScript.
- *Configuration management* – Cloudstudio's configuration management system encourages developers to share early and share often. Developers commit to their own private branches and chose to share their changes when they wish.

[a] Try out Cloudstudio at: http://www.cloudstudio.ethz.ch

- *External Development* – Cloudstudio's configuration management system allows developers who do not wish to use Cloudstudio to still contribute to projects. As Git is the underlying backend for the project, developers can directly connect with the repository and work without being bound to the IDE.
- *Import / Export projects* – Existing projects can be imported into Cloudstudio and existing projects can be retrieved.
- *Monitors* – Monitors provide an early warning system for developers and allow them to keep track of the changes occurring in a project that are important to them [10].

Collaboration:

- *Chat / Skype* – Cloudstudio allows developers to see what other developers are currently working on the same project, and provides access to both chat and Skype from the IDE.
- *Code Reviews* – Code reviews are fully integrated into the IDE, allowing developers to invite their team members to discuss changes without leaving the IDE.
- *Notifications* – All tools have access to a news / notification system, which keeps developers up to date on what is going on in their project [11]. The system is highly customizable.
- *Document Sharing (In progress)* – Teams will be able to collaborate on non-code documents, and see each other's results in real-time.

Verification and Testing:

- Auto Proof – Auto Proof is a static verification tool for Eiffel which allows for proving Eiffel programs in the browser without the need for any additional specifications [12] [13]. Postconditions are tested against possible preconditions to determine if there are cases in which satisfactory preconditions yield unsatisfactory post conditions.
- *Auto Test* – An entirely automated unit-testing suite which infers tests based off contracts [14] [15]. Developers select how long they wish to run the suite for, and Auto Test exercises the classes to test the bounds of the contract.
- *Auto Fix (integration with Cloudstudio in progress)* – While Auto Test tests the bounds of the contracts of a given class and reports failures, Auto Fix will attempt to generate fixes for the errors found [16] [17]. It uses a combination of both static and dynamic analysis to generate fixes, and then regression tests the fixes to determine if they are a suitable candidate to fix the error found.

Details of Implementation

Cloudstudio is developed using Google Web Toolkit and is deployable as an app-engine app. Cloudstudio's editor is an Eiffel program which has been compiled to JavaScript via an Eiffel to JavaScript compiler developed by Alexandru Dima [18]. For back-end data storage, Cloudstudio uses MySQL.

Cloudstudio is being developed at ETH Zürich by the Chair of Software Engineering. Cloudstudio's principal members include Professor Bertrand Meyer, Dr. Martin Nordio, and

Christian Estler. Over 13 masters and bachelors students from several universities have also been involved in implementing Cloudstudio.[b]

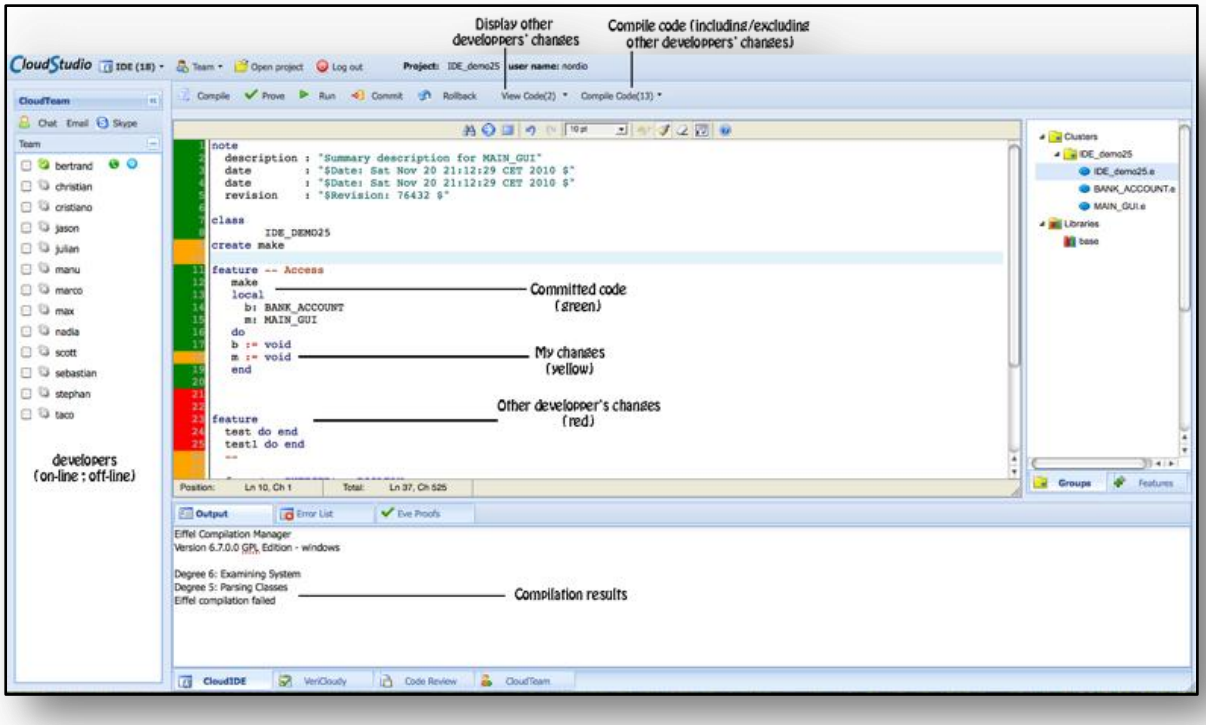

Figure 1: Cloudstudio IDE view.[c]

As is evidenced from the previously mentioned features, Cloudstudio seeks to expand upon the functionalities which are necessary for development while also adding in the features that promote the collaboration necessary to be successful in a distributed development environment. Tools like chat integration, integrated code-reviews, notifications, and monitors bridge the gap that occurs when knowledge cannot naturally circulate through teams that are centrally located. The features in Cloudstudio are tightly coupled yet also flexible, allowing developers to take advantage of the features that benefit them, while not shackling them to the entire suite of tools.

[b] To learn more about the Cloudstudio development effort, visit: http://se.inf.ethz.ch/research/cloudstudio/
[c] Graphic from: http://se.inf.ethz.ch/research/Cloudstudio/

Monitors & Notifications

This report presents several refinements to work implemented in two previously completed reports. The first is a report on monitors [10], the ability to keep informed on code changes that are occurring across multiple projects. The second is a report on a notification system developed for Cloudstudio [11]. A brief description of the work in each is presented below.

Monitors

In small, localized teams, information proliferation about ongoing code changes is a natural consequence of the immediate proximity of the developers. However, larger projects and especially distributed projects face challenges where developers may not be aware of all changes that are occurring within a specific project. Previous work [10] introduces the concept of Monitors, the ability to keep an eye on those changes which are important to a developer. A "monitor" is added to a particular "aspect" of code (library, file, class, feature, pre/post condition, invariant, etc.). When the monitor is added, a "shadow" of the current state of the monitored aspect is archived. The degree to which the aspect should be monitored is specified within a "comparator," which continuously compares the current state of the aspect as well as the shadow to determine if a "violation" has occurred. If a violation has occurred, the monitor informs the developer through one or more specified notification means.

While the aforementioned work on file monitors developed the concepts necessary for the implementation presented in that work, use and peer critique revealed several opportunities for the implementation to be refined and made more effective. In addition there was work to be done in maintaining the monitor implementation through a transition of the Cloudstudio configuration management system.

Notifications

In any software development team, staying up to date on the current changes occurring in the project is mandatory. There are many forms through which that is accomplished: standup meetings, mailing lists, configuration management logs, and many more. Typically, a lot of information is proliferated naturally because developers working on projects are usually co-located. However, when these developers move into a distributed environment, this natural information proliferation cannot occur, and much more attention must be placed on keeping each other up to date on the changes that are occurring.

As most Integrated Development Environments are not designed with web connectivity in mind, creating a part of the IDE that is entirely dedicated to notifying users of recent changes in their projects has not been explored. However, with an IDE that resides fully in the cloud, creating a notification system that collects and distributes notifications from each tool in the IDE can become a reality. In work developed consecutively [11], a notification system that delivers notifications in real-time as well as notifications that were sent while offline was designed and implemented.

The final notification system is well designed, highly extendible, server push driven and tailor-maid especially for Cloudstudio. However, like with monitors, use by both clients to the framework and the final product revealed areas for improvement.

Refinements

This report documents each of several refinements which were made to improve the usability of Cloudstudio's monitoring and notification tools. For each refinement, the motivation for the refinement, the implemented solution, challenges faced, and an analysis of the implementation will be presented.

Configuration Management Transition

The inefficiencies of the configuration management system used in Cloudstudio resulted in duplication when archiving shadows with the original implementation of monitors [10]. The configuration management system, presented in Section 4.2 of "Collaborative Software Development on the Web" [9], does not keep a history of diffs, but instead stored a semi-raw blob in a MySQL database. When file monitors were implemented for this configuration management system, a physical copy of the shadow had to be saved, as the configuration management did not keep track of any previous state. In addition all of the developers' file versions were stored together, which made for tedious reconstruction to check for violations of all of the monitored versions of the files.

In addition to the difficulties faced in detecting violations, the configuration management system also posed problems during version rollbacks. As all versions of a line were stored together, it became impossible at rollback time to determine what lines should be changed to revert to the original state. As a result, a rollback issued by one developer also destroys the current uncommitted progress of all other developers for that class.

Work done by Sandra Weber [19] addresses these deficiencies through the migration from the previously described configuration management system to a system backed by Git. While this new configuration management system was greatly enhanced both the usability and resource footprint of Cloudstudio, it was not backwards compatible with the existing monitor system.

One of the main challenges in migrating to using the new Git based system was a lack of clear documentation. Several operations required obfuscated method calls with combinations of null and non-null parameters to achieve the desired results. In addition, several functionalities of the version control system were not fully implemented at the time of the migration, resulting in the discovery of bugs or calls to methods which had yet to be defined. For instance, there does not exist a rollback function that is capable of rolling back a single file to a specified revision. If such behavior does become available, it would be very beneficial to have it integrated with monitors.

Regardless of these challenges, the final configuration management system provides a much more efficient backing for the monitor system. Instead of being forced to copy the entire contents of a monitored file into a separate database table row, the new system must only keep track of the revision hash of the monitored file at the time the monitor is placed. This reduces the overhead of monitoring tremendously. When a monitor's comparator requests the shadow, this revision hash can then be queried in the configuration management system to produce the necessary shadow. In addition, preforming rollbacks reaps this benefit as well, and the rollbacks can be performed without damaging the current progress of the other developers, a limitation of

the earlier monitoring and configuration management combination. This configuration management system will also be more conducive to the development of the further work described in the report on monitoring [10].

Context for Monitors

Monitors were developed to allow developers to keep an eye on their current projects and finished projects. However, these finished projects may still experience intermittent development, and monitors can be very beneficial here. Monitors are useful for informing a developer that maintenance is occurring on their code or their code's suppliers. However, after a while a developer may not recall why it was important for to monitor a particular aspect. In addition, as a project nears completion, a developer may wish to use a more or less aggressive comparator based on their preferences. In the first implementation of monitors [10], there was no opportunity for developers to document their reasons for monitoring a file. In addition, there was no ability for a developer to modify their comparator preferences after the monitor had been placed. Both of these limit the usability of monitors throughout the life of a project.

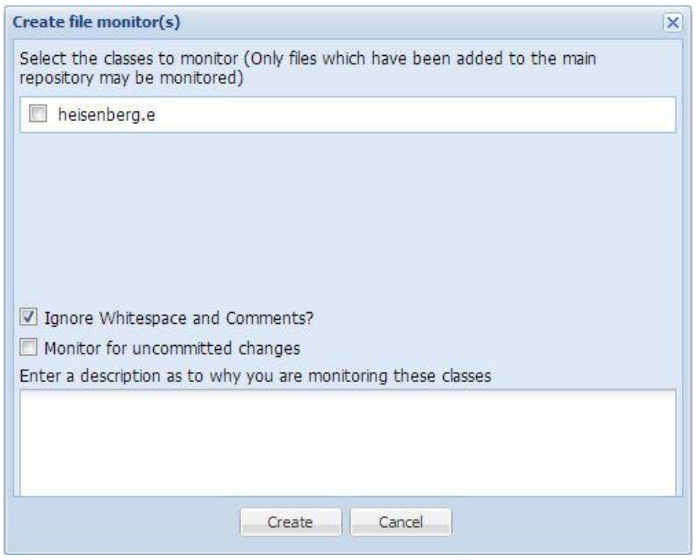

Figure 2: The file monitor add menu with description field

The ability to add a description as well as to modify the comparators for file monitors was added to meet these needs. Upon adding file monitors, the developer is now presented with the opportunity to provide a description (see Figure 2). This is saved with the other monitor properties. When a monitor is violated, the developer receives a notification describing which file has been violated as well as the description that they provided when the monitor was placed. Additionally, the developer can use the menu illustrated in Figure 3 to view the details of the specified monitor. This menu is accessible through the toolbar in them monitor tool and shows the developer several relevant details about the file monitor: the file name, when it was last modified, the user provided description, as well as the ability to change the two comparator options for file monitors (whether to include whitespace and comments and whether to monitor other user's uncommitted changes). When the user dismisses the menu, any updated settings are saved.

If the comparator is modified, the monitor system will recompute whether the aspect being monitored is violated. It will then also appropriately update the UI to correctly display the status of this monitor. This analysis and subsequent updating revealed that the code powering the user interface for monitors was poorly structured, and refactoring this code was a large challenge due to the many different combinations of events which could occur. The first version of the UI developed for the first implementation [10] was much less event-driven than the implementation developed in this report. In attempting to achieve the consistency necessary for the auto-updating, the programmatic approach to keeping the different view components failed. The final implementation is almost entirely event driven, with some slight exceptions where parents may discretely exercise their child without firing an event. This refinement also made it easier to surface violated monitors when the tool is navigated to. While the tool used to open to a blank screen and waited for the developer to select which monitor to examine, this new implementation will open the first violated monitor and will display the violated diff.

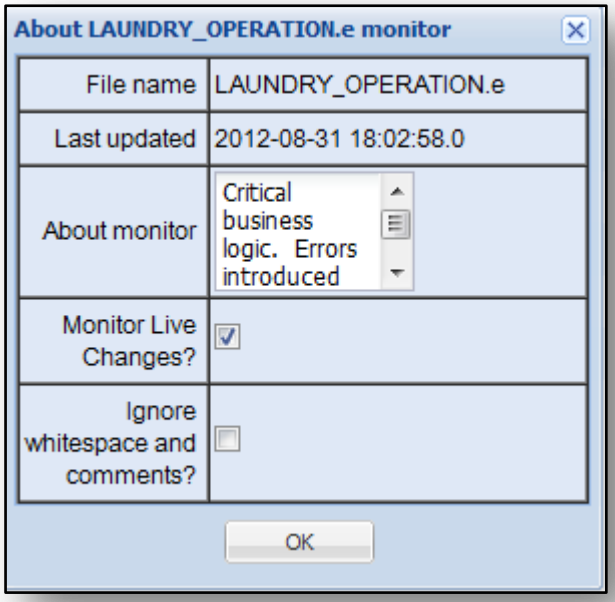

Figure 3: About Menu displaying the message as well as options to modify comparator

Blame

While monitors are effective for helping developers detect when critical code segments have changed, it is up to the monitoring developer to then determine the next course of action to take. Sometimes this may be to do nothing, but other times it may be to begin a conversation about the changes proposed. In order to do this, the developer needs to be able to know who caused the change that violated the monitor. While a developer could look through the configuration management logs to determine who is responsible, it would be helpful to have this information tied into the monitor tool itself.

With the move of the backing configuration management system to Git, Cloudstudio gained access to the ability to determine the last modifying developer for each line (blame). Blame was added as a column to the monitoring tool to help monitoring developers determine with whom they need to communicate. This tool is also beneficial for a proactive developer who wishes to consult the last responsible developer before making changes to a file.

One of the challenges encountered, and something which remains a challenge, is getting the blame out of the Git configuration management wrapper. As mentioned previously, the configuration management system was still evolving at the time that this project was completed. The methods available at that time were not reliable and were highly exception prone. However, the monitor system is "blame ready" whenever blame is conveniently accessible via the configuration management system.

Additional developer integration of notifications

A previous paper on Cloudstudio's notification system [11] recommends a usability study to determine how easily other Cloudstudio developers can add their notifications to the developed system. There are many tools in Cloudstudio which could make use of the notification system, one of the tool's authors used the system.

Denis Cutic of Politecnico di Milano completed a semester project in which he implemented code reviews into Cloudstudio[d]. The code review invitation system used a notification to inform the invited members that they were invited to participate in a code review. Documentation was prepared describing the creation process and then shared with him. He found this documentation to be sufficient, and asked only one question during his implementation, which ended up being an issue caused by a MySQL error. While Denis Cutic's success with the system does not prove it to be easily useful, it does suggest that the system is usable by other developers and in more diverse use cases.

Notification Behavior Upgrades.

In the interim period between the creation of the notification system and the implementation of the refinements here presented, the notification system was exercised by several team members. Feedback was generally positive, but there were three issues that were raised.

Firstly, there was confusion as to the expected behavior when a user clicked on an IDE Notification (a client to the notification system). If the user was in the IDE portion of the site, nothing appeared to happen – the notification was not dismissed and the user was not taken to a different view of Cloudstudio. This behavior was intended in the design of the IDE Notification. The notification was designed to not dismiss on click, but only when manually dismissed. Additionally, the on click event for an IDE Notification was to take the user to the IDE view (where they already were). Yet while this behavior was not a bug and was expected, it did create the illusion of a lack of response when the item was selected. The notification will now dismiss on click, and previously dismissed notifications are available in the news view.

[d] Project title, but no report listed here: http://se.inf.ethz.ch/people/nordio/events_students.html

Secondly, there were issues with the *onLoadEvent*, and a suggestion for further work is described in the original notification implementation report [11]. This mechanism was designed to allow for an event to be fired when a notification is pushed to a logged-in user. However this mechanism also fired events when the notifications were loaded from the database, causing numerous *onLoadEvent*s to execute at login time. As a temporary refinement, this behavior has been entirely disabled for now. It will still be beneficial to pursue the refinement to its fullest by changing the behavior so that *onLoadEvent*s are only executed when a notification does not arrive through the initial database load.

The third refinement was to remedy "a disappearing notification widget." When an IDE Notification was selected, it often appeared that the widget through which this notification was accessed disappeared. This behavior only exhibited itself when a developer clicked on a commit notification for a JavaScript project. The implementers of the JavaScript IDE had failed to add the notification widget that the other IDE's had, creating the illusion of a disappearing widget when the JavaScript toolbar replaced the toolbar of the previously loaded project. This widget has been added to the JavaScript toolbar.

Table of Figures

References

[1] H.-C. Estler, M. Nordio, C. A. Furia, B. Meyer and J. Schneider, "Agile vs. Structured Distributed Software Development: A Case Study," in *7th International Conference on Global Software Engineering*, IEEE, 2012.

[2] E. Carmel and R. Agarwal, "Tactical Approaches for Alleviating Distance in Global Software Development," *IEEE Softw.,* vol. 18, no. 2, pp. 22-29, March 2001.

[3] E. Carmel, Global software teams: collaborating across borders and time zones, Upper Saddle River, NJ, USA: Prentice Hall PTR, 1999.

[4] M. Nordio, R. Mitin, B. Meyer, C. Ghezzi, E. Di Nitto and G. Tamburrelli, "The Role of Contracts in Distributed Development," in *Proceedings of Software Engineering Approaches for Offshore and Outsourced Development*, 2009.

[5] J. A. Espinosa, K. Nan and E. Carmel, "Do Gradations of Time Zone Separation Make a Difference in Performance? A First Laboratory Study," in *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE 2007)*, IEEE, 2007, pp. 12-22.

[6] M. Nordio, H.-C. Estler, B. Meyer, Ghezzi, C. Ghezzi and E. Di Nitto, "How do Distribution and Time Zones affect Software Development? A Case Study on Communication," in *Proceedings of the IEEE International Conference on Global Software Engineering (ICGSE 2011)*, IEEE, 2011.

[7] M. Nordio, R. Mitin and B. Meyer, "Advanced Hands-on Training for Distributed and Outsourced Software Engineering," in *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering (ICSE)*, 2010.

[8] M. Nordio, C. Ghezzi, B. Meyer, E. Di Nitto, G. Tamburrelli, J. Tschannen, N. Aguirre and V. Kulkarni, "Teaching Software Engineering using Globally Distributed Projects: the DOSE course," in *Collaborative Teaching of Globally Distributed Software Development - Community Building Workshop (CTGDSD)*, ACM, 2011.

[9] M. Nordio, H.-C. Estler, C. A. Furia and B. Meyer, "Collaborative Software Development on the Web," 2011.

[10] C. Dentel, "Monitors: Keeping Informed on Code Changes," Independent Research, ETH Zürich, 2012.

[11] C. Dentel, "News and Notification: Propagating Relevant Changes to Developers," Software Engineering Laboratory: Open Source Eiffel Studio, ETH Zürich, 2012.

[12] M. Nordio, C. Calcagno, B. Meyer, P. Müller and J. Tschannen, "Reasoning About Function Objects," in *TOOLS-Europe*, J. Vitek, Ed., Springer-Verlag, 2010.

[13] J. Tschannen, C. A. Furia, M. Nordio and B. Meyer, "Verifying Eiffel Programs With Boogie," in *First International Workshop on Intermediate Verification Languages (BOOGIE 2011)*, 2011.

[14] Y. Wei, H. Roth, C. A. Furia, Y. Pei, A. Horton, M. Steindorfer, M. Nordio and B. Meyer, "Stateful Testing: Finding More Errors in Code and Contracts," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2011.

[15] B. Meyer, A. Fiva, I. Ciupa, A. Leitner, Y. Wei and E. Stapf, "Programs That Test Themselves," *IEEE Computer,* vol. 42, no. 9, pp. 56-55, 2009.

[16] Y. Pei, Y. Wei, C. A. Furia, M. Nordio and B. Meyer, "Code-Based Automated Program Fixing," in *26th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, IEEE, 2011.

[17] J. Tschannen, C. A. Furia, M. Nordio and B. Meyer, "Usable Verification of Object-Oriented Programs by Combining Static and Dynamic Techniques," in *Proceedings of the 9th International Conference on Software Engineering and Formal Methods*, 2011.

[18] A. Dima, "Developing JavaScript Applications in Eiffel," Masters Thesis, ETH Zürich, 2011.

[19] S. Weber, "Automatic Version Control System for Distributed Software Development," Masters Thesis, ETH Zürich, 2012.