Crystal: Precise and Unobtrusive Conflict Warnings

Yuriy Brun[†], Reid Holmes[♣], Michael D. Ernst[†], David Notkin[†]
[†]Computer Science & Engineering
University of Washington
Seattle, WA, USA
[♣]School of Computer Science
University of Waterloo
Waterloo, ON, Canada
{brun,mernst,notkin}@cs.washington.edu, rtholmes@cs.uwaterloo.ca

Abstract

During collaborative development, individual developers can create conflicts in their copies of the code. Such conflicting edits are frequent in practice, and resolving them can be costly. We present Crystal, a tool that proactively examines developers' code and precisely identifies and reports on textual, compilation, and behavioral conflicts. When conflicts are present, Crystal enables developers to resolve them more quickly, and therefore at a lesser cost. When conflicts are absent, Crystal increases the developers' confidence that it is safe to merge their code. Crystal uses an unobtrusive interface to deliver pertinent information about conflicts. It informs developers about actions that would address the conflicts and about people with whom they should communicate.

Categories and Subject Descriptors: D.2.3 [Software Engineering]: Coding Tools and Techniques; D.2.6 [Software Engineering]: Programming Environments
General Terms: Design, Human Factors
Keywords: collaborative development, collaboration conflicts, developer awareness, speculative analysis, version control, Crystal

1. Introduction

Developers working collaboratively on a team can find themselves in conflicting states with one another. These conflicts generally manifest themselves in concrete artifacts within the version control (VC) repository. Such conflicts occur in practice and are costly [5, 7, 8, 10, 11, 12].

This paper describes Crystal, a tool that proactively identifies conflicts in version-controlled artifacts. Crystal provides developers with information about the presence and severity of conflicts as soon as they occur, before the relevant artifacts and changes fade in their minds. Crystal helps developers make well-informed decisions about how to identify and prioritize their tasks, particularly VC operations, and about when and with whom to communicate.

Consider an illustrative scenario in which two developers are enhancing a single program (Figure 1). On Monday, Melinda and Bill start working on different program features, periodically checkpointing their work locally. Melinda takes a week to finish her changes, test them to her satisfaction, and share them with the master code repository. Bill takes two weeks to finish his work, avoiding the distraction of potential conflicts by opting not to incorporate

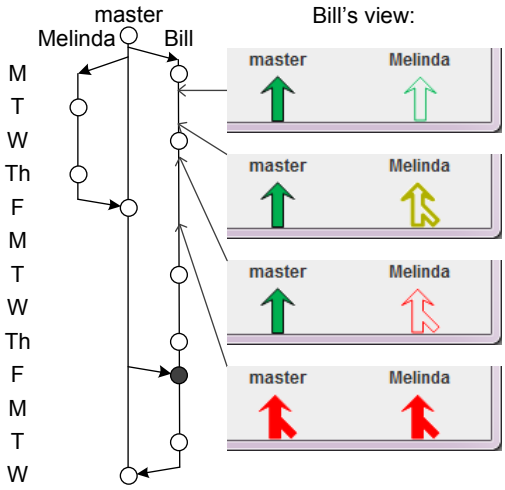

Figure 1: A collaborative development scenario in which Crystal would have enabled Bill to avoid a costly conflict resolution.

On the left is the revision history for Melinda, Bill, and the master repository. Each circle in the history is a code checkpoint. The filled circle indicates the need for manual resolution of conflicts.

On the right is the view Crystal would provide Bill at various points in time. (Shown are actual partial screenshots of our tool. Section 3 describes Crystal's icons.) In the Crystal screenshots, a solid icon represents to Bill a relationship that Bill can affect by performing an action, and a hollow icon represents a relationship that will persist until someone else addresses it.

work from Melinda and the master repository. When he does incorporate Melinda's work (which she has shared with the master) after two weeks, he discovers a conflicting interaction between the implementations of the two features. Bill, unable to resolve the conflict on his own, meets with Melinda. Melinda, who has shifted her focus to yet another feature, has to reconstruct her thought process and assumptions. Their schedules are delayed as they understand the interaction, resolve the conflicts, and produce versions of the features that integrate properly.

Suppose, instead, that Bill were using Crystal. Crystal continuously and unobtrusively reports what would happen *if* he were to share his work and incorporate that of others. The right side of Figure 1 shows Bill's view of Crystal. After his first checkpoint on Monday, he is ahead of (has done strictly more work than) both the master repository and Melinda, and he could share (akin to upload, and signified by an up arrow) his changes to either one. After Melinda's checkpoint on Tuesday, Bill is still ahead of the master repository, but Melinda's work would require a merge, which would proceed cleanly (yellow "merge" icon). Bill's work on Wednes-

ESEC/FSE'11, September 5–9, 2011, Szeged, Hungary.

day conflicts with Melinda's in a way requiring human intervention (red "merge" icon). Once Melinda shares her work with the master repository on Friday, Bill's relationship with the master changes.

Using existing VC tools, Bill could, at any time, choose to determine his relationship with the master repository (the left part of each screenshot); however, he is unlikely to do so as he is trying to avoid distractions. Crystal provides earlier information that enables more effective behaviors without significant interruption. On Tuesday, Bill can see that Melinda is working in parallel. On Wednesday, upon checkpointing his change, Bill can see that it conflicts with Melinda's work. Given that information, Bill may choose to contact Melinda to discuss the unintended feature interaction.

This scenario sketches how Crystal might encourage Bill and Melinda to discover and interact about this conflict as much as $1\frac{1}{2}$ weeks before Bill finishes his feature and $\frac{1}{2}$ week earlier than in the best case using the VC system (VCS) alone.

In some cases, Crystal may help *prevent* such conflicts from occurring. Suppose that on Tuesday, Bill noticed that Melinda was working and asked her to share her work so far. Seeing Melinda's changes, Bill might proceed differently on Wednesday. Even if he does not, resolving the conflict then would be easier, as it would be small and fresh in both Bill's and Melinda's minds.

In addition to textual conflicts identified by the underlying VCS, Crystal can also proactively discover higher-order conflicts. For example, suppose Bill's and Melinda's code merged cleanly from the VCS's point of view, but the merged code failed to compile, or failed its test suite. Crystal accurately identifies such conflicts and reports them to the developers as soon as they occur. In the absence of Crystal, these conflicts are likely to be discovered even later, sometime after the code is merged, incurring even more effort to resolve them. Crystal explicitly checks for compilation and testing conflicts; a future version could perform other kinds of analyses, e.g., to identify performance degradation.

Crystal is open-source, platform-independent, available for download — `http://crystalvc.googlecode.com` — and works with the Mercurial distributed VCS (DVCS). We are extending Crystal to work with Git. Crystal could be made to work with a centralized VCS (CVCS) in two ways: (1) Crystal could speculatively merge branches similarly to the way it currently merges repositories, and (2) Crystal could speculatively merge changes in the developers' working copies. In fact, Microsoft, in collaboration with us, is developing a similar tool for a CVCS.

We have previously evaluated Crystal's potential for helping developers by examining 550,000 development versions of nine open-source systems. We found that both textual and higher-order conflicts occur often and last, on average, for 10 days [3]. In this paper, we focus on how Crystal precisely detects conflicts and effectively communicates information to the developers.

The rest of this paper is structured as follows. Section 2 discusses how Crystal precisely detects conflicts. Section 3 describes how Crystal's user interface reports conflicts without distracting the developer. Finally, Section 4 describes how Crystal's implementation scales to large projects and efficiently detects conflicts.

2. Early and precise conflict detection

Crystal precisely reports actual conflicts, determining the relationship between two developers' states by actually creating the merged artifact. In other words, to find out what would happen if Bill and Melinda merged their code, Crystal, in the background, makes a copy of Bill's code and incorporates Melinda's changes. Similarly, once Crystal creates the merged code artifact, it attempts to compile and to execute the test suite on that artifact. Again, Crystal only reports a compilation or testing conflict when the build or

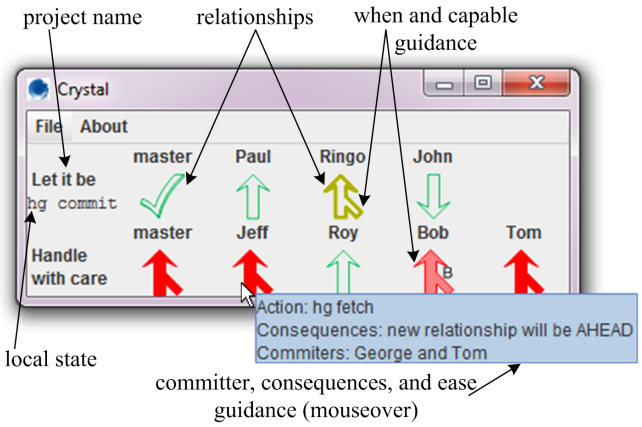

Figure 2: A screenshot of George's view of Crystal. George is following two projects under development: "Let it be" and "Handle with care". The former has four observed collaborators: George, Paul, Ringo, and John; the latter has five: George, Jeff, Roy, Bob, and Tom. Crystal shows George's local state and his relationships with the master repository and the other collaborators, as well as guidance based on that information.

a test actually fails. Because the computation happens in the background, the developers can continue to work without interruption; in certain situations, we expect the developers to ignore Crystal, much as they sometimes ignore project bulletin boards and email.

We refer to the idea of attempting a set of actions on the developer's state in the background and reporting on the outcomes of those actions as *speculative analysis* [2].

Awareness tools [1, 6, 9] notify developers when they *might* have conflicting changes. This approximation is computed differently in various tools. Some determine if a co-developer is working in the same file, some report any change to the repository (e.g., FASTDash [1]), others report concurrent changes to the AST (e.g., Syde [9], etc. These approaches can lead to the inclusion of false positives — reporting potential conflicts that do not evolve into actual conflicts. Furthermore, few current awareness tools try to automatically detect higher-order merge conflicts; again, Crystal is precise as it uses the project's tool chain to dynamically detect conflicts by execution of the build system and test suites. We refer the reader to [3] for a more detailed description of related work.

Crystal can, in rare situations, also report false positives. Checkpointed changes that are later *discarded* can cause a teammate to see a pending conflict that later disappears. This can happen when a developer checkpoints exploratory code or a partial change.

3. Reporting conflicts

Crystal unobtrusively reports four kinds of information: the developer's local state, relationships with other developers or repositories, the possible actions (which is derived from the local state and relationship with the master repository, and which we omit from this paper for brevity), and guidance about those actions. The remainder of this section summarizes Crystal's interface and the information it reports; more details on both are available in [3].

3.1 Example Crystal use

Figure 2 shows a screenshot of Crystal's main window. The window displays a row of icons for each of a developer's projects. In this example, there are two projects: "Let it be" and "Handle with care". The former has four collaborators: George (the developer running Crystal), Paul, Ringo, and John. The latter has five col-

laborators: George, Jeff, Roy, Bob, and Tom. Each developer can independently choose whether or not to run Crystal.

On the left-most side of each row, underneath the project name, Crystal displays the local state (Section 3.2). This tells George, in the native language of the underlying VCS, whether he must checkpoint changes (`hg commit`, in Mercurial) or resolve a conflict. Then, for each repository (master and other collaborators', whether or not they are running Crystal), Crystal displays the relationship with that repository (Section 3.3). If George has the ability to affect a relationship *now*, the icon is solid, which combines the *When* and *Capable* guidance (Section 3.4). If George *cannot* affect the relationship, the icon is hollow. If the relationship is of the *might* variety — George might or might not have to perform an operation to affect the relationship — the icon is solid but slightly unsaturated (see the relationship with Bob in the "Handle with care" project). These features allow George to quickly scan the Crystal window and identify the most urgent issues — the solid red icons — followed by other, less severe icons. George can also quickly identify whether there is something he can do now to improve his relationships (in the example, George can perform actions to improve his relationships in the "Handle with care" project, but not in "Let it be"), and whether there are unexpected conflicts George may wish to communicate with others about. Holding the mouse pointer over an icon displays the action George can perform and the *Committer*, *Consequences*, and *Ease* guidance (Section 3.4), when appropriate.

3.2 Local states

There are five possible local states for a developer.

uncommitted There are uncommitted changes in the working copy.

in conflict The local repository is in conflict with itself; that is, (in DVCS terminology) it has two heads that are not automatically mergeable. This happens, for example, when incorporated and local changesets conflict.

build failure The repository's version of the code fails to build.

test failure The repository's version of the code builds but fails its test suite.

OK The repository's version of the code builds and passes its test suite.

3.3 Repository relationships

We have identified seven relevant relationships that can hold between two repositories. Figure 3 displays the icons Crystal uses to denote each relationship.

SAME The repositories have the same changesets.

AHEAD The repository has a superset of the other repository's changesets.

BEHIND The inverse of AHEAD.

The remaining four relationships represent repositories that share an initial sequence of changesets followed by distinct sequences of changesets.

TEXTUAL✘: (pronounced "textual conflict") The distinct changesets cannot be automatically merged by the VCS.

BUILD✘: The repositories can be automatically merged by the VCS, but the resulting merged code fails to build.

TEST✘: The repositories can be automatically merged by the VCS and the resulting merged code builds but fails its test suite.

TEST✓: The repositories can be automatically merged by the VCS and the resulting merged code builds and passes its test suite.

Higher-order conflicts, such as BUILD✘ and TEST✘, are not considered by existing VCSes. Although this paper discusses only these two higher-order relationships, others naturally arise for other analyses; for example, consider when a test suite passes but a performance analysis or code style checker does not.

Figure 3: Crystal associates an icon with each of the seven relationships. The color of each relationship icon represents the severity of the relationship: relationships that require no merging are green, that can be merged automatically are yellow, and that require manual merging are red.

3.4 Guidance

Knowing how each action may affect the developer's state and relationships can help developers make better-informed decisions.

We classify the guidance information into five types. One type concerns the relationship: *Committer*. The other four concern the possible action: *When*, *Consequences*, *Capable*, and *Ease*.

Committer: Who made the relevant changes?
For example, two developers may be in the TEXTUAL✘ relationship, but the cause of the conflict could be a change made by a third developer. This information is important to identify with whom the developers looking to resolve this conflict should communicate, which, in turn, decreases the time required to fix technical problems [4].

When: Can an action that affects the relationship be performed now, or must it wait until later?
For example, a developer may not be able to affect a TEXTUAL✘ relationship until the other relevant developer shares his changes with the master repository.

Consequences: Will an action — perhaps one on a different repository — affect a relationship?
For example, a developer may be BEHIND because she has not yet incorporated from or because another developer has not yet shared with the master. In the former case, an incorporate action would affect the relationship. In the latter case, it would not.

Capable: Who can perform an action that changes the relationship?
For example, of two developers in TEXTUAL✘, the one who shares with the master first can no longer resolve the conflict, whereas the other developer can.

Ease: Who can most easily resolve a conflict?
Suppose two developers created conflicting changes and one has shared his with the master. If the other were to incorporate the changes, she would have to resolve the conflict. What if the first developer has made a set of follow-up changes that he has not yet shared? If these changes resolve the conflict, then it is likely better for the second developer to wait until the first one shares.

3.5 Limiting distractions

We designed Crystal to convey important information without being overwhelming or distracting. Towards this end, we used the following principles:

- do not asynchronously open windows,
- do not require the developer to keep the main window open,
- keep the main window compact,
- summarize all projects and relationships (allowing for quick visual identification of those that may require attention), and
- make all relationship, possible action, and guidance information available, but hidden until a developer shows specific interest.

Crystal's icons convey urgency and ability through use of color, shape, and saturation in fixed locations in the main window, and mouseover tooltips provide the rest of the information. The system tray icon summarizes the most urgent state and is available at all times, including when the main window is closed.

4. Scalability

This section describes how Crystal's design allows it to scale to large projects and compute the relevant information efficiently.

4.1 Large projects

Crystal scales to large projects that involve many developers and repositories. A developer explicitly instructs Crystal (via a GUI or a configuration file) which repositories to observe. For example, a developer may be interested in only the relationships with other developers in his collaborative team and the per-team development repositories of the other teams.

Crystal can provide information about relationships even with developers who are not using it, easing adoption by avoiding a requirement that the whole team uses the tool. Each developer can independently choose whether or not to run Crystal. Advantages accrue when more members of a team use Crystal, but this is not necessary.

Crystal allows developers to select a subset of the tests to execute, to integrate more smoothly into large development projects with extensive test suites. Naturally, for large projects with build scripts and test suites that take a long time to execute, Crystal will experience that latency. However, it would still identify relevant information sooner than other existing methods.

4.2 Computation efficiency

Crystal provides a developer with information on his development state and the relationships between his repository and collaborators' repositories. Thus, Crystal needs access to that developer's repository and working copy (if the working copy is inaccessible, Crystal does not report certain local states, e.g., uncommitted), and the locations of the other repositories. In some development environments, access to others' repositories is trivial. For example, many corporate development configurations include a common file system. In other environments, it is possible for developers to have their local repositories on machines that are often offline. For such environments, we leverage Dropbox[1] to enable developers to easily share their development states by symbolically linking their working copy to their Dropbox shared folder.

To limit the computation necessary to extract the relationships between repositories, Crystal follows the following algorithm. First, Crystal checks the history of the two repositories to identify the changesets each contains, and only re-computes the relationship if at least one history has changed. If the sets of changesets are the same, then the relationship is SAME. If one repository contains strictly more (respectively fewer) changesets, it is AHEAD (respectively BEHIND). If both repositories contain changesets the other does not (and Crystal has not previously computed their relationship), Crystal makes a local clone of one repository and uses the VCS to attempt to incorporate the changesets from the other repository. If the VCS reports a problem with incorporation, the relationship is TEXTUAL✘. If the integration succeeds, Crystal runs the build script. If that script fails, the relationship is BUILD✘. Finally, if the build script succeeds, Crystal runs the test suite and determines whether the relationship is TEST✘ or TEST✓.

Cloning repositories, especially remote ones, can be costly. To address this issue (and to enable faster start-up times), Crystal keeps a cached clone of each project, bringing it up to date before updating the relevant relationship. This has significantly increased Crystal's performance in all common cases. In the rare and discouraged situation of changing existing VC history (e.g., rebasing), the cache may contain changesets that no longer exist in a repository. This can cause problems and require the developer to clear the cache.

[1] http://www.dropbox.com

5. Contributions

We have described Crystal, a tool that proactively detects and reports on collaboration conflicts. Crystal identifies conflicts precisely and reports them unobtrusively. Developers quickly learn if there are actions they should perform and with whom they should communicate. As a result, they can limit the severity of, or even prevent, costly conflicts.

Acknowledgments

The Crystal beta users provided valuable feedback. This material is based upon work supported by the National Science Foundation under Grants CNS-0937060 to the Computing Research Association for the CIFellows Project and CCF-0963757, by a National Science and Engineering Research Council Postdoctoral Fellowship, and by Microsoft Research through a Software Engineering Innovation Foundation grant.

References

[1] J. T. Biehl, M. Czerwinski, G. Smith, and G. G. Robertson. FASTDash: A visual dashboard for fostering awareness in software teams. In *CHI*, pages 1313–1322, San Jose, CA, USA, Apr. 2007.

[2] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Speculative analysis: Exploring future states of software. In *FoSER*, pages 59–63, Santa Fe, NM, USA, Nov. 2010.

[3] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *ESEC FSE*, Szeged, Hungary, Sep. 2011.

[4] M. Cataldo, P. A. Wagstrom, J. D. Herbsleb, and K. M. Carley. Identification of coordination requirements: Implications for the design of collaboration and awareness tools. In *CSCW*, pages 353–362, Banff, AB, Canada, Nov. 2006.

[5] C. R. B. de Souza, D. Redmiles, and P. Dourish. "Breaking the code", Moving between private and public work in collaborative software development. In *GROUP*, pages 105–114, Sanibel Island, FL, USA, Nov. 2003.

[6] P. Dewan and R. Hegde. Semi-synchronous conflict detection and resolution in asynchronous software development. In *ECSCW*, pages 159–178, Limerick, Ireland, Sep. 2007.

[7] J. Estublier and S. Garcia. Process model and awareness in SCM. In *SCM*, pages 59–74, Oxford, England, UK, Sep. 2005.

[8] R. E. Grinter. Using a configuration management tool to coordinate software development. In *CoOCS*, pages 168–177, Milpitas, CA, USA, Aug. 1995.

[9] L. Hattori and M. Lanza. Syde: A tool for collaborative software development. In *ICSE Tool Demo*, pages 235–238, Cape Town, South Africa, May 2010.

[10] S. Horwitz, J. Prins, and T. Reps. Integrating noninterfering versions of programs. *ACM TOPLAS*, 11:345–387, July 1989.

[11] D. E. Perry, H. P. Siy, and L. G. Votta. Parallel changes in large-scale software development: an observational case study. *ACM TOSEM*, 10:308–337, July 2001.

[12] T. Zimmermann. Mining workspace updates in CVS. In *MSR*, Minneapolis, MN, USA, May 2007.