Development Context Driven Change Awareness and Analysis Framework

Anita Sarma, Josh Branchaud
Matthew B. Dwyer
Univ. of Nebraska, Lincoln
{asarma, jbrancha, dwyer}@cse.unl.edu

Suzette Person[‡], Neha Rungta[†]
[‡]NASA Langley Research Center
[†]NASA Ames Research Center
{suzette.person,neha.s.rungta}@nasa.gov

ABSTRACT

Recent work on workspace monitoring allows conflict prediction early in the development process, however, these approaches mostly use syntactic differencing techniques to compare different program versions. In contrast, traditional change-impact analysis techniques analyze related versions of the program only after the code has been checked into the master repository. We propose a novel approach, DeCAF (**De**velopment **C**ontext **A**nalysis **F**ramework), that leverages the development context to scope a change impact analysis technique. The goal is to characterize the impact of each developer on other developers in the team. There are various client applications such as task prioritization, early conflict detection, and providing advice on testing that can benefit from such a characterization. The DeCAF framework leverages information from the development context to bound the iDiSE change impact analysis technique to analyze only the parts of the code base that are of interest. Bounding the analysis can enable DeCAF to efficiently compute the impact of changes using a combination of program dependence and symbolic execution based approaches.

Categories and Subject Descriptors

D.2.9 [**Software Engineering**]: Management—*programming teams*; D.2.4 [**Software Engineering**]: Software/ Program Verification

General Terms

Verification, Algorithms

Keywords

Change impact analysis, distributed software development, change awareness, conflict prediction

1. INTRODUCTION

Software development is largely performed in teams and involves parallel development. The distributed development paradigm requires team members to coordinate their changes. A developer needs to understand how his changes may impact other ongoing changes [9] and how changes made by other developers may impact his tasks. Teams typically depend on software configuration management (SCM) systems to manage team development; developers use private workspaces as their development sandboxes and synchronize their changes periodically with a central repository [2].

Direct and indirect conflicts occur when developers are not fully aware of ongoing changes and their impact. A merge conflict is a direct conflict that occurs when a developer attempts to check-in her changes, but, a divergent revision of the file exists in the repository. An indirect conflict occurs when the program behavior assumed by one developer is changed by another developer in parallel. Both direct and indirect conflicts can lead to build and test failures [9].

Traditional change impact analysis techniques identify the impact of a change set on the original code base, whereas conflict prediction requires the computation of a developer's changes on not only the central code base, but, also on the other ongoing changes in remote, parallel workspaces. Designing the right change impact analysis technique that can assist in development tasks can be challenging. Some of the challenges are determining (a) how to perform an analysis when there are no changes (i.e., changes that are proposed or not yet completed), (b) which versions should be treated as source and target programs when there are many developers, (c) how to bound the scope of the analysis to answer specific questions and facilitate scalability, (d) how to configure the precision of the analysis such that it provides meaningful results in a timely manner, and (e) how to process and present the results such that they are useful to the developers.

In this work, we propose a novel context driven change awareness and analysis framework, **De**velopment **C**ontext **A**nalysis **F**ramework (DeCAF), that leverages the context of the distributed software development environment to scope the analysis space. DeCAF can be configured to scope the region over which impact is calculated, the precision of the analysis, and the extent of analysis based on the client requirements. DeCAF uses a multi-stage change impact analysis, iDiSE, as the underlying analysis engine [7, 8].

2. SCENARIOS

Consider a scenario where a development team consists of four developers: Joe, Sally, Alice, and Bob. Figure 1 depicts this scenario where team members have their own local workspaces (or local repositories) and a set of files and

ICSE '14, May 31 - June 7, 2014, Hyderabad, India

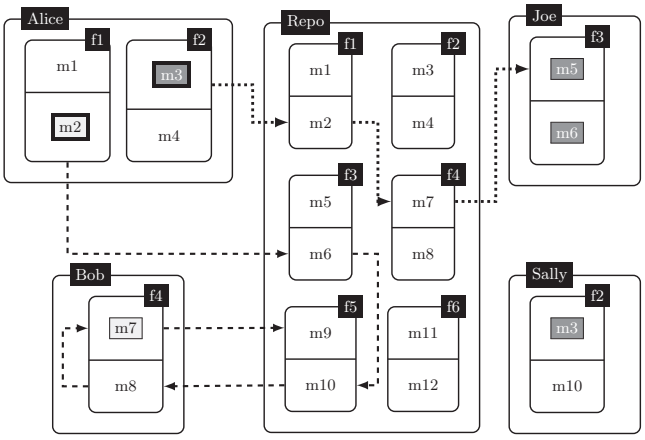

Figure 1: Change Scenarios

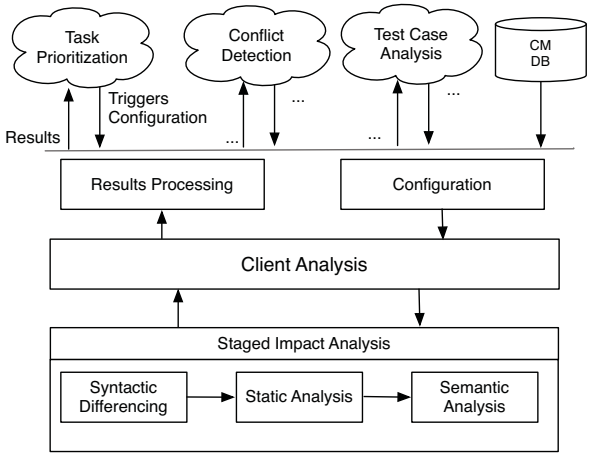

Figure 2: DeCAF Architecture

tasks they are working on[1], e.g., Bob is working on file f4. Figure 1 depicts the repository for the system (repo), which consists of six files, f1 - f6, comprised of 12 methods, m1 - m12. Let us now consider three development scenarios.

Task Prioritization. Imagine that Alice has been assigned two tasks, t1 and t2. Task t1 involves modifications to method m2 (shown in light gray) and task t2 involves modifications to method m3 (shown in dark gray with white text). Ideally she will select the task that will have the least impact on the rest of the team's ongoing work. Furthermore, imagine that Bob's current work is comprised of a task, t3, which is on the critical path to an upcoming release; we depict t3 with a light gray shading—method m7 in file f4. Alice would like to ensure that her changes do not impact any task that is on the critical path to a release (here, Bob's task t3). Therefore, she wishes to know the answer to the question: "Could working on task t1 impact any critical task (t3)?". The dashed sequence of edges illustrates that such an impact is possible and Alice can prioritize her tasks accordingly, i.e., by working on task t2.

Conflict Detection. Now assume that Alice decides to work on task t2 (by modifying m3). While Alice is modifying m3, Sally also starts to edit m3 and Joe starts working on method m5 (f3). Note that in this case, once everyone has completed their changes there will be a merge (direct) conflict between Alice and Sally, and an indirect conflict between Alice and Joe (the potential impact is depicted by the dotted calling sequence). Also assume that while making her changes Alice would like to know the answer to the question: "How are my changes going to conflict with changes that other developers are working on?". Since both Alice and Sally are modifying m3, a direct conflict exists, but the dotted sequence of edges also illustrates a potential indirect conflict with Joe – through m5. These answers allow Alice to determine whom to communicate with on her team.

Testing Advice. Now assume that Alice's team practices code ownership, whereby changes are only made to a file by the developer who *owns* that particular file. Here, her code ownership includes files f1 and f2, but not f4. Note that we can deduce such code ownership from the versioning system, and that if the team follows strict code ownership, Alice would not face a merge conflict with Sally, as in our previous scenario. Once Alice finishes her changes she would like to know the answer to the question: "How much testing should I do?". The answer to her question will help her identify which test cases she needs to execute so that methods that may be impacted by her changes are tested. If she is aware that her changes extend beyond her code ownership, she might do more extensive testing.

[1]We depict only the files each team member is currently working on; local workspaces may contain copies of additional files from the repo to enable building of the workspace.

3. APPROACH

The DeCAF change awareness and prediction framework is shown in Figure 2. At the heart of the DeCAF framework is a client analysis which uses information from *triggers* based primarily on developer activity and user-specified *configurations* to (i) select the precision of the analysis, (ii) provide bounds for the analysis, and (iii) determine when and how often the analysis should be performed. The staged change impact analysis in DeCAF leverages the specified bounds and precision to scope the analysis. Note that the semantic aspect of the analysis allows DeCAF to provide more and better information compared to other state of the art techniques such as Palantir [9]. Finally, the results of the staged analysis are processed and formatted such that they can be mapped back to the client analysis.

Actual versus Potential Impact. DeCAF can be configured to compute the impact of actual changes made to the local workspace or those committed to the central repository. Local changes can be tracked by instrumenting the IDE, e.g., tracking the task context in Mylyn [6]. DeCAF can also be configured to compute the potential impact of planned changes when provided with code locations that the developer intends to change. In the task prioritization scenario, Sally uses the client to specify her intended changes in method m2 in f1 or in method m3 in f2.

Analysis Direction. In certain development tasks, such as task prioritization, a developer may want to gauge the impact of his changes on other developers. However, in other tasks, such as conflict detection, a developer may need to understand the impact of changes made by other developers on his code. DeCAF supports computing the impact in either direction providing configuration options to specify the source and target versions.

Analysis Extent. Traditional change impact analyses compute the impact of *a given set of changes* in a program

version [7, 8]. DeCAF can be configured to additionally compute how two or more distinct change sets may impact each other. An impact set is first generated for each change set and then an intersection of the impact sets is computed. In the conflict detection scenario, the impact of Alice's changes is intersected (compared) with the impact of Joe's changes.

Analysis Scope. It is computationally expensive to perform the change impact analysis on the entire code base, moreover, the developer who initiates the analysis is often interested in only what impact the changes may have on the parts of the code she is currently changing. The part of code relevant to the developer can be marked as a region of interest (ROI) in DeCAF. The ROI can be specified in terms of a set of files, classes or methods, or using heuristics, such as distance heuristics that include only methods within a specified distance in the call graph from the changed method. Scope specified using distance heuristics is easy to compute using an inexpensive call-graph analysis, and subsequent analyses may perform more detailed analysis of the dependence chains in order to reduce the scope. In the task prioritization scenario in Section 2, the call-graph analysis generates ⟨m2,m6,m10,m8,m7,m9⟩–a call sequence for when Alice is considering a prospective change to m2 (task t1) and the ROI is represented by Bob's ongoing work on m7. The analysis is scoped by truncating the analysis at methods in the ROI, thus eliminating the analysis of m9.

Analysis Precision. The underlying change impact analysis, iDiSE, performs a series of change impact analyses, using the results of an imprecise (but also inexpensive) static analysis to drive a more precise (and expensive) technique based on symbolic execution. This enables DeCAF to balance the trade-off between the cost and precision of the analysis based on the information necessary to support the development task. In the task prioritization scenario, once the analysis detects a conflict, it can be terminated. However, if a conflict exists and the developer wants to know the details of the conflict, she can configure DeCAF to use iDiSE to compute the details of the conflict.

Results Format. The change impact analysis results of iDiSE can be represented in terms of source code locations or program behaviors that may be impacted; the latter expressed as constraints on program variable values computed by symbolic execution. For some development tasks, such as the task prioritization scenario in the previous section, it is useful to also map the iDiSE results onto the development context to identify the impacted developers, tasks, or test cases. DeCAF supports such mappings by using information stored in the SCM system, source code annotations, project task definitions and assignments, or additional information provided by the user.

Thus far, we have discussed how DeCAF can be configured based on the development task and the type of answers sought. We can imagine other types of development contexts that could be used to scope the change impact analysis. For example, if a user is interested in knowing only the impact of her changes that are outside her code ownership, then DeCAF need not perform an in-depth analysis if all of the impacted code is within her code ownership. Similarly, if the user is interested in knowing how her changes may affect a particular developer or a particular file, then the impact on only those entities needs to be computed. Therefore, development heuristics such as code ownership, active workspaces (where files are being modified by users), team policies (code that is "frozen" before a release, or public APIs that should not be impacted) could also be used to scope the analysis performed by DeCAF.

4. DISCUSSION

In this section we discuss the parts of the DeCAF architecture that we have implemented, our current challenges, and our proposed solutions to those challenges. The key elements of the DeCAF architecture include:

∘ *Triggers* to determine when and how often to invoke the analysis, based on developer activity.

∘ *Configurations* to provide a mechanism to control the analysis scope and specify the required precision.

∘ The *Client Analysis* uses information from the configuration to compute the scope of the analysis.

∘ The *Staged Change Impact Analysis* uses options specified in the configuration to compute the set of impacted artifacts.

∘ *Results Processing* uses the impacted artifacts to provide answers for a specific client analysis.

Certain components of the DeCAF architecture can be designed and implemented more easily than others. The implementation of the triggers, configurations, and client analysis in the DeCAF framework is straightforward. For a client analysis such as conflict detection, the analysis is triggered by a workspace monitor when a developer successfully compiles his code or commits the code to his local repository in a distributed SCM system such as Git. While the analysis for a task prioritization activity is explicitly invoked by the developer, the configurations for the scope and the precision of the analysis are specified by the developers.

The client analysis is a core component of the DeCAF engine. It uses the information from the triggers and configurations to generate the set of analysis artifacts for the change impact analysis and processes the results from the change impact analysis. The client analysis is agnostic to the underlying change impact analysis to a large extent. This design decision allows different change impact analyses to be plugged into the DeCAF framework. Most change impact analysis techniques take two related program versions as inputs. The client analysis includes additional computation algorithms to translate the impact results to elements in the development context. For example, if Alice is interested in the impact of her changes on both Bob and Joe, the client analysis invokes the impact analysis twice where $I_0 := (\texttt{Alice}, \texttt{Bob})$ and then $I_1 := (\texttt{Alice}, \texttt{Joe})$, and then generates the intersection of the two impact sets I_0 and I_1 to compute the impact on both Bob and Joe.

We use iDiSE as the change impact analysis within the DeCAF framework. There are three stages to the iDiSE analysis (a) syntactic differencing, (b) static change impact computation, and (c) behavioral change impact computation using symbolic execution [7, 8]. These are increasingly more expensive to compute, but they also provide better precision. An important goal in this work is to reduce the number of false positives reported to developers by techniques based solely on syntactic or static impact analysis algorithms. The number of false positives can be reduced by using the semantic analysis component of iDiSE that is based on symbolic execution. While this approach can produce generally more

precise analysis results, it also introduces a significant challenge: symbolic execution is generally performed on a specific program unit, e.g., method. We are currently working on approaches that limit symbolic execution to specific parts of the program in a way that provides meaningful results in the development context. This involves devising bounding schemes for symbolic execution that are based on the development context rather than the traditional bounding mechanisms such as depth. Development of these bounding techniques will be a key to not only scalability of the more precise symbolic execution based analysis, but, is also important to generate information relevant to a particular client analysis.

Finally, the client analysis processes the output of the change impact analysis to generate the set of impacted results: impacted program statements, path conditions, or test cases. We are currently working on algorithms that will allow us to process the output to answer specific questions, such as which task should be prioritized over another.

5. RELATED WORK

Several recent approaches based on heuristics and simple program analysis have attempted to address the problem of conflict prediction. FastDash [1] uses workspace monitoring to identify when the same file is being updated in multiple workspaces to detect merge conflicts. Syde [5] performs a more sophisticated analysis to extract changes at the Abstract Syntax Tree level to precisely identify the type of change. CollabVS [3] and Palantir [9] use a combination of workspace monitoring and simple syntactic analysis (e.g., call-graph analysis) to identify direct and indirect conflicts. Crystal [2] and the approach by Guimarães et al. [4] integrate ongoing changes by maintaining a shadow repository where each local commit (in the case of Crystal) and each change in the workspace (in the work by Guimarães et al.), is integrated and built, and test scripts executed to identify which changes would conflict and at what level (merge, build, or test failures). Safe Commit [10] uses a combination of impact analysis and test/build scripts. It first decomposes changes to their atomic level and then identifies the set of dependent changes that can be safely committed.

Existing workspace awareness techniques for distributed development environments identify the impact of changes occurring in parallel workspaces. For example, Crystal [2] analyzes a single stable change in a workspace with respect to the repository, and Palantir [9] analyzes the impact of ongoing changes across workspace pairs.

Our work differs from existing approaches in that it leverages information about the development context to configure the underlying change impact analysis to help answer the kinds of questions that are being asked by the client. These client requirements drive the level and precision of the change impact analysis that is performed.

6. CONCLUSIONS

In a distributed software development environment conflicts may arise due to changes being made in parallel by different developers. In this work we present **De**velopment **C**ontext **A**nalysis **F**ramework (DeCAF) which leverages the development context to scope a change impact analysis technique. The key novelty of DeCAF is that it attempts to characterize the impact of one developer's work on other developers, and the impact of changes being made by other developers on their tasks. Various client analysis can be enabled by DeCAF, such as task prioritization, early conflict detection, and testing advice among others. DeCAF enables the user to specify a bound around the parts of the code base to analyze, and the precision of the analysis, based on the needs of the client analysis. The configuration options enable the impact analysis to provide meaningful results in a timely manner.

Our initial evaluation plan for DeCAF is to assess its efficiency and effectiveness to support task prioritization and conflict detection tasks. To generate the artifacts for our evaluation, we have analyzed the git repositories for two large open source projects to extract the commits. We are using the commits as a proxy for tasks and will evaluate DeCAF on its ability to identify task dependencies and potential conflicts. We will also use the commits within a specific time period (1 to 2 weeks) as tasks and the associated changes for the commits to analyze the impact of changes post-hoc.

7. ACKNOWLEDGMENTS

This research is supported by grants AFSOR FA550-10-1-0406 and NSF CCF-1253786 and HCC 1110916.

8. 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, 2007.

[2] Y. Brun, R. Holmes, M. D. Ernst, and D. Notkin. Proactive detection of collaboration conflicts. In *ESEC/FSE*, pages 168–178, 2011.

[3] P. Dewan and R. Hedge. Semi-Synchronous Conflict Detection and Resolution in Asynchronous Software Development. In *European Conference on Computer Supported Cooperative Work*, pages 24–28, 2007.

[4] M. L. Guimarães and A. R. Silva. Improving early detection of software merge conflicts. In *Proceedings of the 2012 International Conference on Software Engineering*, ICSE 2012, pages 342–352, Piscataway, NJ, USA, 2012. IEEE Press.

[5] L. Hattori and M. Lanza. Syde: A tool for collaborative software development. In *ICSE*, pages 235–238, 2010.

[6] M. Kersten and G. C. Murphy. Using task context to improve programmer productivity. In *FSE*, pages 1–11, 2006.

[7] S. Person, G. Yang, N. Rungta, and S. Khurshid. Directed incremental symbolic execution. In *PLDI*, pages 504–515, 2011.

[8] N. Rungta, S. Person, and J. Branchaud. A change impact analysis to characterize evolving program behaviors. In *ICSM*, 2012.

[9] A. Sarma, D. Redmiles, and A. van der Hoek. Palantir: Early detection of development conflicts arising from parallel code changes, 2012.

[10] J. Wloka, B. G. Ryder, F. Tip, and X. Ren. Safe-commit analysis to facilitate team software development. In *ICSE*, pages 507–517. IEEE, 2009.