Indirect Conflicts: An Exploration and Discussion of Tools, Process, and Developer Insight

Jordan Ell
University of Victoria, Victoria, Canada
jell@uvic.ca
IEEEauthorblockNDaniela Damian
University of Victoria, Victoria, Canada
danielad@csc.uvic.ca

***Abstract*—Developer awareness techniques have been proposed and studied to aid in developer understanding and efficient developer output in terms of source code produced. These techniques involve the sharing of information regarding source code changes in a specific project in order to either catch potential issues before they are fully integrated or to help developers debug pre-existing issues. These techniques have largely focused on either *direct* or *indirect conflict* issues. While the techniques and tools for direct conflicts have had large success, tools either proposed or studied which focus on indirect conflicts have had many common struggles and setbacks. To better understand why indirect conflict tools either fail or have little developer investment, we performed a grounded theory study with 97 developers involved in either an interview or web survey. We identified the main features that characterize indirect conflict issues, what processes and techniques developers use to aid in preventing and solving indirect conflict issues, and what developers would look for in a tool to aid them in these conflicts.**

I. Introduction

II. Related Work

III. Methodology

Our study was performed in two parts. First, a round of semi-structured interviews were conducted which addressed the larger who, what, when, where, and how about indirect conflicts. Secondly, a survey was given out which was used to confirm what was learned from the interviews on a larger sample size as well as get a larger opinion base on general issues.

A. Interview Participants

We selected our interview participants from a large breadth of both open and closed source software development companies and projects. The population which participated in our interview came from the following software groups: IBM, Mozilla, The GNOME Project, Microsoft Corporation, Subnet Solutions, Ruboss Technology Corporation, Amazon, Exporq Oy, Kano Apps, Fireworks Design, James Evans and Associates, and Frost Tree Games. Our participants we chosen based on their direct involvement in the actual writing of software for their respective companies or projects. These participants' software development titles included: senior developer, lead developer, software developer, software developer in test, and junior developer. In addition to software development, some participants were also chosen based on their previous experience of software development as well as their current experience with project management at some capacity. These management type participants involved in project planning to some capacity between technical architecture, project maintainer, chief technical officer, or program manager.

B. Interview Procedure

Participants were invited to participate in the interviews by email and were sent a single reminder email one week after the initial invitation if no response had been made. We directly emailed 22 participants and ended up conducting 19 interviews. Interviews were conducted in person when possible and recorded for audio content only. When in person interviews were not possible, one of Skype or Google Hangout was used with audio and video being recorded but only audio being stored for future use. We were quite pleased with the response rate to our initial invitation emails as our breadth of participants was rather large spanning multiple open and closed source project and our interview answer saturation was quite high (to be seen in Section IV). However, we were disappointed in our word of mouth campaign. We had asked several participants to pass along our interviews invitations to other members of their respective organizations but did not receive any additional participants through this method.

Interview participants first answered a number of structured demographic items. Next, participants were asked to describe various software development processes they use or encounter in their current working environment. These structured process questions were used for later analysis to provided context to any answer for questions that involve indirect conflicts. They next were engaged in a semi-structured round of questions pertaining to 12 categories of questions.

- Software development tools for dependency tracking and awareness.
- Software development process for preventing indirect conflicts.
- Software artifact dependency levels and where conflicts can arise.
- How developers find internal or external software dependencies.

- Examples of indirect conflicts from real world experiences.
- How indirect conflicts are detected and found.
- How indirect conflict issues are solved or dealt with.
- Developer opinion of preemptive measures to prevent indirect conflicts.
- Developer opinion on what types of changes are worth a preemptive action.
- Developer opinion on who is responsible for fixing or preventing indirect conflicts.

Each of the categories was used as a point of aggregation for later analysis as a way of answering the X research questions laid out in Section I. While each of the 12 question categories had a number of starter questions, interviews largely became discussions of developer experience and opinion as opposed to direct answers to any specific question. In general, once the category was explained to the participant, little prompting was necessary as the participant would jump right into an experience, or issue he or she had with the question or category. However, not all participants had strong opinions or any experience on every category mentioned. For these participants, answers to the specific categories were not required or pressed upon. We attribute any non answer by a participant to either lack of knowledge in their current project pertaining to the category or lack of experience in terms of being apart of any one software project for extended periods of time. We account for these non answers in our analysis as seen in Section IV. Interviews lasted from 15 minutes up to 75 minutes. The length of the interview largely depended on the amount of experience the participant had in software development as they were able to give more experiences and more informed opinions.

After the interviews, we extracted a list of answers, opinions, and experiences for each of the 12 categorical question fields. We combined commonly mentioned opinions and answers where possible, and noted where experiences with indirect conflicts from participants were similar but generally kept the experiences tied to the individual. Where more direct answers were given, we used the wording of our participants rather than our own descriptive aggregations. The results of this preliminary analysis was a list of short hypotheses supported by direct developer experience and opinion for the 12 categories. These hypotheses were then used as a template for creating the second step survey, in order to further test the answers given by interview participants and validate any possible conclusions to be drawn.

C. Survey Participants

We selected our survey participants from a similar breadth of open and closed source software development companies and projects as the interviews participants with two large exceptions. The software organizations that remained the same between interview and survey were: Mozilla, The GNOME Project, Microsoft Corporation, Subnet Solutions, Ruboss Technology Corporation, Amazon, Kano Apps, Fireworks Design, and Frost Tree Games. However, participants who took part in the round of interviews were not asked to participate in the surveys but rather to act as a contact point for other developers in their team, project, or organization who may be interested in completing the survey. Aside from this aforementioned list, two large groups of developers were asked to participate as well, these being GitHub users as well as Apache Software Foundation (Apache) developers. The GitHub users were selected based on their large amounts of development activity on GitHub and the Apache developers were selected based on their software development participation on specific projects known to be used heavily by other organizations and projects.

D. Survey Procedure

The created survey was based off of the 12 categorical hypotheses given by the round of interviews. The survey was designed to test these hypotheses and to acquire a larger sample size of developers who may have similar or different opinions from those already acquired from the interviews. The survey went through two rounds of piloting. Each pilot round consisted of five participants, who were previously interviewed, completing the survey with feedback given at the end. The previous interview participants were selected based on their domain knowledge expertise with indirect conflicts in order to provide what we think would be amount to better feedback and resulting in a more polished survey. The final survey consisted of the following questions: 2 multiple choice questions for developer experience and demographic information; 3 level of agreement questions for an indication of what types of environments indirect conflicts are more likely to occur; 6 level of use questions to indicate what types of software changed developers find trouble some with respect to indirect conflicts; and 9 short answer questions for indication of when indirect conflicts occur, what types of processes are used to prevent or react to indirect conflict, and to provide an outlet for general opinion on the matter. Each non demographic question was made optional as it was shown through the interviews that some questions require more experience from participants than may be provided.

Survey participants were invited to participate in the survey by email. No reminder email was ever sent as the survey responses were not connected with the invitation email addresses and thus participants who did respond could not be removed from a reminder list. We directly emailed 1300 participants and ended with 78 responses giving a response rate of 6%. We attribute the low response rate with a couple of factors. One, the surveys were conducted during the months of July and August while many participants may be away from their regular positions. Two, our GitHub and Apache participants could not be verified as to whether or not they actively support the email addresses used in the invitations. And finally, the survey was considered by some to be long and require more development experience than may have been typical of some of those invited to participate.

After the survey was closed, preliminary analysis was applied to the short answer questions in the same manor as the interview responses. We extracted a list of answers, opinions, and experiences for each of the 9 short answer question fields while combining commonly mentioned answers where possible. The results again from this preliminary analysis is a list of common and unique hypotheses to each of the 9 short answer questions.

IV. Results

We now present our results of both the interviews and surveys conducted in regards to our research questions outlined in Section I. We first present and discuss the general nature of indirect conflicts in terms of when, what types, and how often indirect conflicts occur. We next show the findings for the process related research questions in regards to how developers react to, processes to prevent, and reactionary steps to indirect conflicts. Lastly, we present and discuss findings for what potential tools are wanted by developers and what researchers are currently developing.

A. The Nature of Indirect Conflicts

Although a detailed manual inspection of a project's lifetime and the indirect conflicts it experiences throughout its development is beyond the scope of this paper, we first wanted to explore the general nature of indirect conflicts. This being the case we explored, through our interviews and surveys, what software developers believed to be the case when it came to how indirect conflicts occur, when and how often they occur, and the types of software object's that incur indirect conflicts.

RQ1 *How do indirect conflicts occur?*

From the interviewed participants, 63% of developers believe that a large contributing factor to the cause of indirect conflicts comes from the changing of a software object's contract. One developer stated that "design by contract was invented to solve this problem and it does it quite well", while another noted that software object contracts do solve the problem in theory, but that doesn't mean that problems don't occur in practice. In light of object contracts, 73% of interviewed developers gave examples of indirect conflicts they had experienced which stemmed from not understanding the far reaching ramifications of a change being made to an object contract towards the rest of the project. Of those 73%, examples were broken down as follows. 21% of examples had to do with the changing of legacy code, with one developer saying "legacy code does not change because developers are afraid of the long range issues that may arise from that change". 36% of interviewed developers mentioned that indirect conflicts occur when an update to an external project, library, or service occurs and the resulting integration fails noting that major releases of libraries can be a large issue with one developer saying "their build never breaks, but it breaks ours". Finally, 29% of developers interviewed explained that changes to databases are often a large source of indirect conflicts saying "[schema changes] break stuff all over the place".

RQ2 *How often do indirect conflicts occur and at what times in a project do they occur more frequently?*

From the interviewed participants, 58% of developers explained that indirect conflicts occur all the time in their development life cycle with a minimum occurrence of once a week. Of those 58%, developers noted that a lot of their development time is put into understanding code to avoid indirect conflicts, more serious issue tend to happen once a month, and that the conflicts that do occur tend to be quite unique from each other.

In terms of when in a project indirect conflicts are more likely to occur 63% of developers interviewed said that when a project is in the early stages of development, indirect conflicts tend to occur far more frequently than once a stable point is reached. Developers said "At a stable point we decided we are not going to change [this feature] anymore. We will only add new code instead of changing it." and "the beginning of a project changes a lot, especially with agile". 21% if developers interviewed mentioned that the development process changes once a stable point of a project is met and the "add and deprecate model solves a lot of these issues".

For those developers interviewed, we asked how often indirect conflicts occur in their software development experience, how often they occur before and after a project's first release, as well as how often indirect conflicts occur late in a project's lifetime. Table I shows the results of these questions.

Of those surveyed that could not give direct answers to the questions in Table I, the more common answers were as follows: "People rushing to meet [the release date] is where most of my indirect conflicts occur.", "Indirect conflicts after a release depends on how well the project was built at first", "They tend to slow down a bit after a major release, unless the next release is a major rework.", and "Conflicts continue to occur at roughly the same rate after the first release, with spikes during large revamps or the implementation of cross-cutting new features.".

RQ3 *What software artifacts and environments are susceptible to indirect conflicts?*

From the developers interviews, 9 were currently working with large scale database applications and all 9 listed database schema as a large source of indirect conflicts. Out of the 5 developers interviewed that currently work on either software library projects or in test, all 5 said that methods or functions were the root of their indirect conflict issues. Some other notable artifacts were user interface displays for web development, and full components in component base game architecture.

As per environments, Table II shows which development team environment developers believe to be the most prone to indirect conflicts.

B. Processes of Indirect Conflicts

The results that follow pertain to software development process and its relation to indirect conflicts. We explore how developers react to indirect conflicts, which stakeholders are responsible for their fixes, and if any development processes can be used to prevent or catch indirect conflicts.

RQ4 *How do developers catch indirect conflicts?*

TABLE I: Results of survey questions to how often indirect conflicts occur, in terms of percentage of developers surveyed.

Question	Daily	Weekly	Bi-Weekly	Monthly	Bi-Monthly	Yearly	Unknown
How often do you experience indirect conflicts in your software development?	17%	27%	21%	17%	3%	5%	10%
How often do indirect conflicts occur in the early stages of a projects development?	32%	18%	4%	5%	0%	5%	36%
How often do indirect conflicts occur before the first release of a project?	13%	29%	6%	8%	1%	3%	40%
How often do indirect conflicts occur after the first release of a project?	6%	18%	8%	18%	1%	5%	44%
How often do indirect conflicts occur late in a projects lifetime?	6%	5%	5%	18%	8%	12%	46%

TABLE II: Results of survey questions to development environments in which indirect conflicts are likely to occur, in terms of percentage of developers surveyed.

Environment of conflicts	Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
Developing alone (conflicts in own code)	18%	20%	19%	24%	19%
Developing in a team between 2 - 5 developers (Inter-developers conflicts)	3%	8%	22%	49%	18%
Developing in a multi team environment (Inter-team conflicts)	1%	11%	14%	39%	35%

Of the 19 developers interviewed, 69% mentioned forms of testing (unit, integration, etc) as the major component of catching indirect conflict issues, subscribing to the idea of "run the regression and integration tests and just see if anything breaks". 31% of developers said build processes (either nightly builds or building the project themselves), and others mentioned code reviews while those dealing with a user interface mentioned user complaints from run time testing.

Of the 78 developers survey, 49% mentioned forms of testing as the major tool used to catch indirect conflicts, 33% said build processes, while 31% used work their IDE or IDE plugins to catch indirect conflicts. Other developers surveyed also mention code review process (14%) and project expertise (6%) as factors of catching indirect conflicts.

RQ5 *How to developers react to and solve indirect conflicts?*

Once an indirect conflict has occurred, usually alerted to the developer by test or build failures, 75% of developers interviewed said they checked historical logs to help narrow down where the problem could originate from. Most developers had the mindset of "Look at the change log and see what could possibly be breaking the feature.". The log most commonly referred to was the source code change log to see what code has been changed, followed by build failure or test failure logs to examine errors messages and get time frames of when events occurred. Developers also mentioned their personal expertise in the project as a large component of solving indirect conflicts as well as issue or work item trackers as many tests and builds can be linked to these management tools.

Of the developers surveyed, 37% confirmed that they used some type of source control management tools to help them solve their indirect conflicts while 23% said they used native IDE tools and 21% said they use features of the language's compiler and debugger in order to solve indirect conflicts. Interestingly, only 13% of developers mentioned a form of communication with other developers in aid to solving these conflicts and only 4% mentioned the reading of formal documentation.

C. *Tools for indirect Conflicts*

RQ6 *When are pre-emptive indirect conflict tools best used and what would developers want from them?*

The developers who were interviewed had a major concern with any pre-emptive tool in that the amount of false positives provided by the tool may render the tool useless. Developers said "this could relate to information overload", "this would be a real challenge with the number of dependencies", "I only want to know if it will break me", and "it depends on how good the results are in regards to false positives". These concerns combined with RQ2 suggest that pre-emptive tools are best used in unstable phases on the project when indirect conflicts are more likely to happen, thus reducing the number of false positives. One developer in particular noted "You will spend all your time getting notified and reviewing instead of implementing". Interviewed developers also suggested that their proper software development process are already in place to catch potential issues before they arise such as code review, individual knowledge, IDE call hierarchies (static language analysis), or communication within the project. Finally, most interviewed developers mentioned that their current work flow does not subscribe to the ideas of pre-emptive prevention and that they just "work until something breaks".

As per what developers would want from such pre-emptive tools, Table III shows how developers feel towards being alerted in certain types of functional changes within code which they are using.

RQ7 *What kinds of tools do developers want to help them handle and solve indirect conflicts?*

The developers who were surveyed, the following list represents the more common suggestions for how an indirect conflict resolution tool should be implemented.

- Aid in debugging indirect conflicts by finding which recent code changes are breaking a particular area of code or a test.
- Automated testing to show how code changes affect old tests and the ability to automatically write new tests to compensate for the change.
- IDE plug-ins to show how current changes will affect other components of the current project.
- Analysis of library releases to show how upgrading to a new release will affect your project.
- Built in language features to either the source code architecture (i.e. Eiffle or Java Modeling Language) or the compile time tools to display warning messages towards potential issues.
- A code review time tool which allows deeper analysis of a new patch or pull request to the project allowing the reviewer to see potential indirect conflicts before actually merging the code in.
- A tool which is non-obtrusive and integrates into their pre-existing development styles without them having to take extra steps.

V. Discussion

VI. Future Work

VII. Conclusions

VIII. Acknowledgments

We would sincerely like to thank all participants who were willing to be interviewed or who participated in completing our survey. We thank these people for taking time out of their day to participate and for sharing their developer experience with us. Without these people our research could not have been possible.

TABLE III: Results of survey questions to source code changes that developers deem notification worthy, in terms of percentage of developers surveyed.

Code change type	Never	Occasionally	Most Times	Always	I Don't Care
Method signature change	5%	8%	12%	68%	7%
Pre-condition change	5%	27%	37%	23%	7%
Main algorithm change	11%	45%	19%	15%	11%
User interface change	12%	32%	20%	27%	9%
Run time change	13%	29%	25%	20%	12%
Post-condition change	7%	28%	32%	23%	11%