From: "Harald Gall and Nenad Medvidovic" <icse2011-papers-chairs@borbala.com>
Subject: ICSE 2011 Paper Notification [210]
Date: 17 November, 2010 4:02:43 PM EST
To: rtholmes@cs.washington.edu, notkin@cs.washington.edu
Cc: icse2011-papers-chairs@borbala.com, icse2011-papers-webadmin@borbala.com
Reply To: icse2011-papers-chairs@borbala.com

Aloha,
Dear Reid and David,

Thank you for your submission to ICSE 2011. The program committee met on November 12-13, 2010, to consider the submissions to the Research Paper track. We are pleased to inform you that your paper,

"Identifying Program, Test, and Environmental Changes That Affect Behaviour"

has been accepted for presentation in the technical program and for publication in the conference proceedings. The competition was strong: only 62 of the 441 submissions were accepted, giving an acceptance rate of 14%.

We enclose a set of reviews of your paper. In your preparation of the final paper, please make sure to incorporate the comments of the reviewers.

VERY IMPORTANT INFORMATION:

1. Fairly soon, you will receive an author kit from the ICSE 2011 Publications Chair.

2. The final camera-ready paper is due on Friday, February 11, 2011. This deadline is firm --- if you miss the deadline, your paper will not appear in the proceedings.

3. You will find information about the format of the camera-ready version of your paper on the ICSE 2011 conference website. The format specifications are the same as were required for your initial paper submission (ICSE 2011 Format and Submission Guidelines). Your paper must not exceed 10 pages.

4. As mentioned in the call for papers, your paper submission and acceptance implies that one of the paper's authors must attend and present the paper at the conference, which will be held May 21-28 in Waikiki, Honolulu, Hawaii.
Early in the Spring, closer to the conference, we will post instructions on the web site to help you prepare for that presentation.
Failure to attend the meeting to present the paper will result in the removal of your paper from the proceedings; more specifically it will not appear in the IEEE or ACM Digital Libraries.

We want to congratulate you on the acceptance of your paper, which is part of a strong and exciting conference program we are assembling for ICSE 2011. We look forward to seeing you at the conference.

Sincerely,

Nenad Medvidovic and Harald Gall
Program Committee Co-Chairs
ICSE 2011

=--==--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=

First reviewer's review:

Summary of the submission <<<

The paper discusses code changes and the static and dynamic divergences caused by them. The paper does this by exploring how static code changes should relate to dynamic code changes. The authors then form some basic assumptions that some kinds of divergences might be harmful (i.e., a change may have had undesired consequences) whereas other kinds of changes appear as desired. A simple example is an added functionality (method call) which adds to the static structure and should also add to the dynamic structure.

Evaluation <<<

I think the authors are onto a nice idea but is it useful? I strongly concur with the authors that commonalities and differences between static and dynamic structures could reveal problems. However, it seems that at no point the detected problems were validated. Thus, the question I have is whether the problems detected by the approach are a) true problems and b) important problems. There should be some measure (perhaps precision/recall) that shows that the approach is indeed able to identify the problematic kinds of changes.

What the evaluation showed was that the user would not be overwhelmed by the proposed approach. That is the areas perceived to be problematic are few and far apart. This is good and useful. But were these few places identified truly problematic? I cannot tell.
Why was the study on three open source systems limited to 10 commits only? You could have chosen a much larger number as everything appears to be automated.
Even if I presume that the approach is able to identify certain problematic areas then how serious are these problems. What kinds of problems can the approach not detect? I am thinking of code changes that do not change the underlying call graphs but rather the pattern of interaction: e.g., method A used to call method B when X and Y happened, now it is only calling B when Y happens - it seems to me that you are dealing with call graphs instead of call trees which could not deal with such subtleties. But are these subtleties really important?...
I am also unsure as to what we are to learn from the industrial study in 4.2. The patterns appear similar to the open source studies. I see no conclusions there except for informal statements made by the developer. For example, "Oh, that's interesting, we have..." is about the observation that S+ points out additions to the code that apparently where never tested (hence no D+). However, this is a rather weak benefit. Now, it is indeed important to know failure to test (don't get me wrong here) but wouldn't a simple branch coverage technique for testing not take care of this?
Minor: missing V2D label in figure 1

Points in favour or against <<<

+ nice idea
- not clear whether results are useful

=--==--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*

Second reviewer's review:

Summary of the submission <<<

This paper provides a new perspective on how a developer can perceive code changes. It does
so by simple comparing the static and dynamic call graphs on the old and new code base,
and across them. As a result the analysis can identify inconsistencies (e.g. static calls
lacking the corresponding dynamic call, or a dynamic call that was not preceded by a

static call). The proposed approach is assessed by analyzing partitions sizes and potential
uses on 27 versions of open source system, and performing an informal and more qualitative
study in industry.

Evaluation <<<

I found the ideas of the paper quite interesting, especially the partition/categorization
emerging from the analysis of differences between static and dynamic call graphs across versions.
Developers often struggle in understanding the impact of changes because of all the moving parts involved in a system and this relatively inexpensive approach could provide some answers, highlighting particular kinds of situations in an integrated way.

As I read the paper though, I kept thinking that many techniques exist to detect the issues found by the approach. For example, simple checking the change in call-coverage metrics would reveal the same as the "s+d" partition. Another example, changes in the environment could be used to check whether old and new components behaves alike (from the industrial study) is really just regression testing.
In some parts I was also unsure whether some partitions would really capture what is intended. For example, a simple change in the initial value of a variable may affect a predicate that determines whether a call is made or not. This change would not be reflected in the static call graph but would change the dynamic one, ending up in the sd+ partition which makes me question whether the granularity at the call level is enough.
The assessment did not help with these two issues that much. It did not convey how much is lost due to the lack of precision, nor was it very convincing in showing how the approach differs from simply using static or dynamic approaches along (the subsection on this only reports that the partition is smaller with the current approach, but that is just because part of the data goes into another partition).
In spite of the size of the assessment effort, in the end the paper fails to strongly convince about the uniqueness and added value of having this type of change partition.

Still, in the end I feel the ideas are interesting enough and there is a value in establishing a framework to classify changes quickly enough to give developers timely feedback, which is why my evaluation is on the positive side.

Points in favour or against <<<

+ interesting framework to classify changes to support developers
- value of having change partition framework fizzles out as the paper goes on

=--==--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*

Third reviewer's review:

Summary of the submission <<<

Describes an approach that combines static and dynamic analysis to classify changes in caller-callee pairs to a system resulting from a code change. Five categories of caller-callee changes are described, of which 3 are of particular interest to the developer. The approach and an implementation are described, as are two studies. One examines 10 versions each of three open source systems, determining how many pairs end up in each category and analyzing some interesting changes qualitatively. The other involved comparing nightly builds of an industrial product, and interviewing the product manager. In all cases, the number of interesting pairs is manageable, allowing developers to focus on them and check whether they really indicate problems or not.

Evaluation <<<

A really excellent idea that has great deal of practical value. I can see this sort of analysis becoming standard with builds, as some other forms of analysis are.

The second paragraph of the intro is rather vague. It would be good to have an example here.

The paper seems to consider the "consistent" category of even more interest to the developer than "not executed". This surprises me. Wouldn't the developer consider the "consistent" ones as expected, and not be inclined to examine them? A bit of discussion would be good, especially if this point of view is wrong.

The text about Fig. 2 mentions a "key clash", but there is no indication of keys in the code - which also makes the cache example look odd; it looks like the cache is just remembering all the integers returned, in a way that is useless. Of course, it still demonstrates

the point being made.

Figure 3, esp. 3(a), is hard to impossible to read when printed in black and white.

Points in favour or against <<<

+ A really excellent idea that has great deal of practical value. I can see this sort of analysis becoming standard with builds, as some other forms of analysis are.
+ Retrospective evaluation showing that the number of interesting dependencies is small
+ Industrial case study
+ Readable

=--==--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*=--=*