Title: 웹 소프트웨어의 신뢰성 16

- ✓ **Instructor: Gregg Rothermel**
- ✓ **Institution: KAIST**
- ✓ **Dictated: 강단비, 김주현, 김지현, 주다은**

[00:00]

Today I will present a paper which is practical fault localization for dynamic web applications.

This is pulblished at ICSE 2010 and the authors are as followed.

The first author is the one of the most active researcher in web application and fault localization. Actually this… There have many fault localization techniques to locate program fault efficiently.

But past techniques are known as not practical, because of each inaccuracy.

But in this paper, the authors improve accuracies significantly, compared to past techniques.

So in this paper, the authors suggest an approach that helps developers to try to locate program faults in PHP applications.

To do this, they use 2 well known techniques which are CONCOLIC execution and Tarantula which is representative coverage-based fault localization technique.

So, they implement their idea, their tool called 'Apolo' and, they conducted experiment.

The experiment result show that most faults in the program could be found by expecting less than 1% of executed program statements.

So I will present this presentation from motivation to conclusion.

I will briefly introduce motivation of this paper, and then, I will explain the high level idea of this paper.

And then, I will explain 2 steps to localize program faults automatically.

Okay, this, as you know, program debugging is very time consuming task.

When we test program, and the program failures are detected.

Developers usally find the program faults by inspecting and tracing failed execution.

So, tracing all failed executions consumes a lot of time, because we need to inspect full program.

If a failed execution, executes full of the program source code.

So in this paper, the authors propose a tool localizing program faults in PHP application on server side.

That is they try to suggest a technique that helps programmer find the program faults when the failures are detected.

So, to do this, they adapt to techniques, which are CONCOLIC testing, CONCOLIC test generation technique and coverage-based fault localization technique.

Okay. This is simple PHP program.

You know syntex of PHP? Maybe, ya…

So, in this program, you don't need to know the detail of source code, but in the line 18, there is a program fault.

Which is this line print, HTML source code, when this condition is evaluated as true.

So… But, in this line 18 H2, HTML tag is not closed, because this H2 tag is omitted on line 18.

So, when an execution executes line 18, then the marked for HTML will be generated.

So, we can detect failure in the execution.

But, to find the faulyt statement which is line 18, programmers visit inspect all lines executed in the execution.

For example, if a execution executes line 8 to line 11, and line 12 and line 16, 17 and 18, we need to inspect all the lines executed by the execution.

[05:00]

So this is obviously time consuming task.

So the authors suggest a technique that can pinpoint this line, or that can rank suspicious statements that may contain program faults.

So this is overall approach of this paper.

When program P is given, they suggest idea generate test cases while executing program P.

To do this, they use CONCOLIC execution.

I will explain this technique in the later slide.

And after passing this phase, the approach outputs this information.

This… Which are executed statements of each test case and each test case's result.

If a failure is detected while executing an execution, and a test case which executes the program will be maked as fail, because the failure is detected.

If a test case, uh… While executing the program, there is no failure, then the test case for the execution will be marked as pass.
And then, by using this information, the suggest approach measure possibility that each statement contains program fault.

To do this, they adapt Tarantula, which is one of the most famous and presentative coverage-based fault localization technique.

Also I will explain this technique in the later slide.

By using this technique, the ouput of this technique is suspiciously the rank of each statement in P.

If a statement have high suspiciousness, it will be ranked as 1.

And a programmer will inspect statements in the order of decreasing suspicioness.

So this is the first work which is fully automatic tool that finds and localizes marked form HTML errors in PHP code on the server side.

Actually they have other tests that target C code but this is first work on PHP, PHP code on the server side.

(student speaking)

In the JAVA…

There is the technique that targets JAVA script but they adapt, the use fault localization technique which is not coverage-based.

They use slice-based technique.

So as I mentioned, the programmer will inspect the statement in the order of decreasing suspicioness.

So I will explain test case generation and execution in their approach.

In this paper the authors, the approach generated test cases in a systemitic way while executing the program.

To do this, they execute an application on some initial or random input.

And then, they generate an additional inputs obtained by solving constraints, derived from exercised control flow paths.

That is, if an initial input, then, it…, the approach executes the input concretely and symbolically.

And the approach get pass constraints of input while executing the program.

So it is usually called CONCOUC execution, because it executes… CONCOLIC execution.

Which means that combine concrete execution and symbolic execution.

I will explain CONCOLIC execution in the example.
Let's assume there is a 3 symbolic variable in the conditional statement.

[10:00]

Line A, we will call it as A, and line 11, we will call this fragment as B, and line 17, we will call this fragment as C.

And this is control photograph of this program.

Each node is labeled as line number of conditional statement or line numbers of its basic block.

Red line circled node is conditional statement.

So, when an initial test case TC1, which consists of some inputs given, then let's assume that TC1 executes constraint, executes the program in the path which is A, B.

This is just end, A, B, and C.

Then TC1, after executing TC1, we can get path constrait of TC1 which is A and B and C.

And the CONCOLIC execution generates another test cases by using this path constraint.

They will negate last conjunct of path constraint, wich is not C.

Which is C, they will conjunct C, they will negate C, as like this.

But generating this test case is unsatisfiabl, because…

Generating this case is not possible because B and not C, cannot be satisfied at the same time.

So they will regate last conjunct of this past constrate again.

At this moment B will be regated like this.

Then the CONCOLIC Excultion genarates test cases that satisfy this pastconsequence.

By this way CONCOLIC Excultion genarate test cases by excuting the program.

So this is the detailed strategy for PHP applications.

They addapt CONCOLIC excultion for the PHP application.

The inputs of the approch is that program P which consist of PHP scripts.

and a setup is treatable PHP scripts from initial state.

They don't explicably mention about the deffinition of state.

But we can infer that state C is consist of DV or special state in the web.

And the output oracle detacts program failuer if program failuer occurs.

While the program execution.

and then by using this inputs the approch generate a procedure to localize program faults.

First step is generate a random test for each excutable PHP scripts at initial state C.

If an initial state is given, and only some of PHP scripts in the program will be executed.

so they generate a random test for each PHP script.

and then, they execute a random tests concretely and symbolically.

By this way they can generate test cases in a systemitic way.

So these two steps are concrete execution.

And then in step four they find another executable PHP scripts from the generative test cases in step two and step three.

when the state is changed, there will be other PHP scripts they can be excuted.

So they find executable PHP scripts in step four.

And then they repeat step one to step four.

[15:00]

There is no termination condition.

But in the experiment, they limite time for the test case generation

So the output of the algorithm is coverage of program P when executing each test case.

And exedution result of each test case which is pass or fail.

By using this information, the coverage based false·true technique will be performed.

I explained this pase, and by usong this information,

the tarantula technique will masure possibility that each what each state contains program fault.

So the possibility is suspiciousness that a statement contaions program fault.

We cannot exactly calculate probability that a statements contains program faults.

But the technique infer possibility that a statement contains program fault.

So to do this trantula technique is used for measure suspiciousness of each statements.

So in the previous page, we saw that by genetically test cases and execluting program,

we can program coverage of each test case and test case result.

And then tarantula gives suspiciousness to each statement,

that representing possibility that a statement contains program fault, by using this information.

The main intuition is that if a statement is highly correlates with failuar execlution,

that is if a statement is executed the all of the failuer test cases then the statement may contain program faults

than other statements that are not executed by the ?[17:36] executions.

I will explain in the example.

This is a simple PHP program and if we generate a test cases which are DC1 to DC4.

Let's assume that DC1 cover from line 1 to line 8, and line 11 to line 13 and line 16.

And DC2 and DC3 and DC4 as follows.

And then DC1 will be marked as fail because line 13 which contains program fault.

This executed by DC1.

As you can see there is no closing H2 tag on line 13.

And DC2 also will be marked as fail because it is execute line 13.

And the oracle will detact malformed HTML caught.

But in case of DC3 and DC4 will ba marked as pass because there is no malformed HTML caught.

and the oracle will say DC3 and DC4 passed program correctly.

(Student questioning)

O.K I think that is out of this paper.

There will be other technique, such as HTML variator.

Yeah but I think they don't mention about such techniques.

But maybe malformed HTML will degrade performance or there synthetic error so oracle anc detact it.

[20:00]

I don't know detailed idea of the oracle.

(Student questioning)

So by using this information is gathered from the test case generation and problem equition.

In this approch they use CONCOLIC excultion to generate this information.

And by using this information the coverage based false true present technique
Calculate suspiciouessness of each statemant that contain program fault.

So this is the suspiciousness matrix of each line in the program.

In the numorator it is failed L of the total failed.

Failed L stands for the number of failed exeicutions that execute line L.

Total fail is the number of total test cases in the program.

In this case total failed will be the number of total failed test cases.

will be two because there is to test cases.

Test L is the number of passes test cases that execute lins L.

And total test is the number of passed test cases in the program.

In this case there are two test cases that passed program.

and this is exactly same as numorator in the matrix.

In this case line 1 will be calculated as the suspicioussness of line 1 will be calculated as 0.5

Because failed L is two because the number of failuer test case that execlute line one is two.

DC1 and DC2 execlute line one.

And total failued is the number of total test failuer test cases is two.

because there are two failed test cases in the program.

Also there are in case of this one the number of passed test caes that execlute line L is two.

Because there are two test cases passing line one.

And there are also two total passed test cases in the program.

and this is exactly same as this one.

So in this case line the suspicioussness of line one is calculated as 0.5.

As you can see the equation is that if many failuer test cases execlute line L.

then the suspicioussness increases.

This is because, as you know, to cause program failuer.

The program fault which is the cause of program failuer should be execluted

So if line L contains program fault the line L will be execluted many failed test cases.

If line L contains program 4th, the line L will be executed many failed test cases.

So, the suspicions of line L will be increased, it will be increased.

So this is, in case of line A, the suspicions will be calculated as 1.0, this is typo.

So…okay.

As you can see, there are two failed test cases that execute line A.

And there are two total failed test cases execute line A.

And there are no past test cases, okay, there are no past test cases execute line A and there are two failed test cases executing line A.

[25:00]

So, it will be calculated as 1.0.

So, by this way, we can calculate this suspicion of each statement in the program as like this.

So by using this suspicion is, we, the technique also give suspicion rank of each statement in the program.

In this case, line 8 and line 13 will be, will have rank one, so rank one.

And we can find program fault by expecting statements in the order of decreasing suspicions and we can, we can find the program fault by expecting line A and line 13.

So this will help programmer find, find the program fault effectively.

(Student speaking) I have a question not about this, just about…because if a work is constructed by HTML combinator, so when line 2, if we don't have, there is no bar like CPHP, a solver, in this time deadline is away or past because the…

(Student speaking) the evaluator is just about HTML?

Yeah, actually, they used…they also detect exclusion failure in addition to HTML evaluator.

So they can detect program failure, they not only use HTML failure but also exclusion failure detector.

Okay.

So, this is, this is how the fault, the coverage-based localization technique works.

However, the previous, the Tarantula, original Tarantula technique is not as, not enough to localize program fault effectively, so authors combined their technique with, two techniques suggested in the paper.

The first technique is called the Output Mapping Technique.

They use HTML validator that reports locations of HTML code where it is syntactically incorrect. HTML validator reports locations when the malformed HTML is detected.

So they, they…okay.

So, they…HTML validator reports locations of HTML code where it is syntactically incorrect.

And then, output mapping technique which is implemented as the map, reports locations of PHP code that generate reported incorrect HTML code.

So by this way, when the malformed HTML code is detected, then they maps HTML code to PHP script that generate malformed HTML code.

So in this case, when the PHP code generate malformed HTML, when the PHP code is reported as, reported by output mapping technique, then the suspicion of line L that is reported by the technique is calculated as 1.

And otherwise, it is, it is calculated as 0.

I don't know why…

Okay.

[30:00]

(Student speaking) I have a question.

(Student speaking) When they make output mapping table, how can they formulate the statement in HTML to the statement in PHP, is it automat, is it possible to make that…

Yeah, they, they briefly explain about that in the implementation section.

They, they instrument interpreter and then they construct map for each PHP script that generate HTML.

So that they maintain the map to map PHP script to HTML code.

I don't know how exactly they implement the map but they briefly explain about that.

(Student speaking) So HTML mapping data tell the location of the…

HTML code.

(Student speaking) HTML code, but does not tell the….

PHP, yeah.

They implement additional ones to map PHP script to HTML code.

(Student speaking) ?

Maybe, yeah.

Interpreter, maybe, I think you may find implementation section in the paper

(Professor speaking) I'll read the same.

(Professor speaking) They're our shadow interpreter, so they've got a shadow interpreter, creates the mapping by recording the line number of the originating PHP statement whenever output is written out using the [?31:57] statements.

(Professor speaking) The producing statement is found in the map using the positional information reported by an oracle checking the output performance.

So, in addition to this technique, they combine output mapping technique with Tarantula technique.

So the intuition is that if a statement is reported as suspicious one by Tarantula and output mapping technique at the same time, the statement may contain program fault with high probability.

So, when the line L, line L is reported as output mapping technique, when the line L, the suspicion is of, the suspicion of line L in output mapping technique is 1 and the suspicion of line L in Tarantula technique is greater than 0.5, they give suspicion of line L as 1.1.

Otherwise, they just give suspicion is, suspicion of line L by using Tarantula technique.

So, also they…give, actual, originally, Tarantula gives suspicion is to each statement.

But in this paper, they suggest a new approach that is, they give suspicions to each branch instead of each conditional statement.

For example, when, if a statement S is 'if' statement both S true and S false receive suspiciousness.

This is because of missing statements in the program.

I will explain it in an example.

As you can see, this is another PHP script in this presentation.

As you can see, when body tag is closed when 'if' statement is evaluated as 'true'.

If 'if' statement is not evaluated as 'true', then there will be no body, no closing body tag in the HTML code.

So, that can make malformed HTML code so the oracle will detect program failure.

[35:00]

But this is, there is no faulty line in this program source code because the program faults comes from because of missing statement which is body tag, which is closing body tag.

But in the original Tarantula technique, the Tarantula technique gives suspicions to each statement so we cannot, we don't know which portions of source code contain program fault.

For example, line number 10 cannot have highest rank since it is executed by both pass and failed test cases.
Line number 10 is executed by passed test cases as well as failed test cases and the line 10 is evaluated false.

But if we can give suspicion to each branch instead of each commitial statement.
Then line number 10 force will have high suspiciousness since it is executed by only failurure test cases.

So they can skip branch number 10 and force is only executed by failure test cases so the by the tarantula matrix as we saw on a previous page.

The suspicion of this branch is calculated as 1.

So we can, the progammer, they know, may have hints for finding program foot.

So by using those techniques they conduct experiment to compare their suggested idea and original tarantula technique in the PHP applications on the server site.

So they compare accuracy of 5 techniques.

The accuracy is defined as percentage of executed statements that need to be expected for finding programming fault while expected statement in order of decreasing suspiciousness.

That we as we saw in the previous example, it means that they measure ranking of faulty statement in tarantula matrix and their suggested matric.

So they compare accuracy they try to show, accuracy of their technique is better than original tarantula technique.

So they compare those 6 techniques which are original tarantula technique and the second one is output mapping and third one is tarantula + output mapping as we saw in the previous page.

And fourth one is tarantula with condition modeling as we saw in the previos page.

And fifth one is that tarantula with condition modeling + output mapping technique.

And last one is tarantula with condition modeling or output mapping technique.

Tarantula with conditioning modeling is used for PHP execution failure and output mapping is used for malformed HTML failure.

As output mapping just maps malperformed HTML to PHP script, so it cannot be applied to PHP execution failures, so they use output mapping for malformed HTML failure and they use TC with conditional modeling, they use tarantula with conditional modeling for PHP execution failure.

So the subject programs are those 4 programs which are..I don't know how..what program does but and they use those 4 programs and they generate test cases by concolic execution and they limit time budget, they give time budget as 20 minutes.

[40:00]

And then they expect program and they find program faults in each program.

In case of factforge there are 20 faults in the program, in case of web chest there are 15 faults in the program.

And they detect 130 failures in pack forge.

This number different, the number between this one and this one is different because several fail executions can occur because of 1 faulty statement.

So this is the result.

As we can see tarantula with conditional modeling + outperforms other techniques.

XEC is percentage of statements that need to be examined for finding programming fault and why XEC percentage localize faults among 75 faults in the program.

As you can see more than 80% of following faults localize by expecting 1% of executed fault statement.

(Student questioning)

No, they manual expect program, they find fault in the program to find faulty program.

(Student questioning)

They don't use their technique for finding fault in the program, they just manual expect program for the experiment.

So they argue that their technique significantly better than original tarantula technique as you can see taratula technique, this line?

Blue line but their technique is this line, as you can see significant difference between tarantula and TC + O, in terms of percentage of statements that need to be examined.

(Student questioning)

TC + O...there are 2 kinds of failures in this subject program, first one is malformed HTML failure and second one is PHP execution failure.

So in case of malformed HTML failure they just use outpeforming technique for malformed HTML failure.

But in case of PHP execution failure they just use tarantula with conditional modeling.

But in this case they use 2 techniques at the same time for any kinds of program failure.

So in conclusion, the autors proposed effective fault localization techniques for PHP applications.

Their approach increases accuracy compare to original tarantula techniques significantly.

So based on this experiment the authors argue that this approach is very practical faults localizing technique, but as you may know coverage of localization of technique has critical assumptions that severely and negatively affect practicability of suggested technique.

As you know we need to have 100% correct test oracle to detect program failures.

Also they assume that when programmer expect statement, the programmer know the statement is faluty or lie.

However in here deciding a statement faulty or not is very difficult as you may know.

Thank you for listening, any questions?