# 웹 소프트웨어 신뢰성

- ✓ **Instructor: Gregg Rothermel**
- ✓ **Institution:** 한국과학기술원
- ✓ **Dictated: 강우리, 김지수, 임보라, 이병찬, 박지영**

## [0:00]

Okay, hi.

My name is Kyung-lok and today I'll present a paper called 'A Framework for Automated Testing Of JavaScript Web Applications.'

This is a problem statement.

As Java script web applications are becoming more complex, the need for java script testing is increasing accordingly.

But there aren't many tools available right now and even they usually require manual test case generation which is inefficient.

So this paper tries to propose framework for automated testing of java script applications.

And first, next is the background knowledge of java script and one of the characteristics of java script is that this language is usually event driven nature as the java script is…

I mean the main purpose of the java script is to make the web page GU like interactive, so most of the cord fragment of the java script is composed of events and event handlers.

So, for instance, user clicks the button then the relevant event handlers will be invoked then, the result of that event handlers will be used in another event handler and so on.

And next is the interaction with DOM.

Document –Object Model is an API for HTML and XML.

It provides structural representation of document and at the same time it enables the language such as Java Script to modify its content and visual presentation.
And finally feedback – directed random testing.

Random testing is very easy to implement and easy to use and also it produces so much, it can produce so much test cases.

But the downside of random testing is that is creates many illegal and redundant testing inputs.

What do I mean by illegal is as you can see in the left side, example, setting the Mons to minus one of date object is somewhat ridiculous but if you use the random testing policy these kind of illegal test inputs that can be generated.

And redundant test is the case in the right side.

Um… you can see that the test DS whether it is NT or not.

**[2:59]**

But beforehand the A, alphabet A is already insulted in the S.

So, it is meaningless to test its empty condition.

But again random testing can generate such a redundant test cases.

So feedback directed random testing is to avoid such a situation.

This kind of testing technique will generate new inputs based on the… from the legal or non redundant test cases so to avoid such a useless test cases.

And here is the framework this paper proposes and Artemis of a framework for automated feedback directed test generation for Java Script web applications.

And there are three components of Artemis.

First is execution unit which models the browser and server.

These components with actually try run execute the test cases.

Next is input generator which produces new input sequences.

And finally the prioritizer will try to prioritize the test cases according to their importance.

And this is the algorithm process.

I will explain it one by one.

First the state is the state of web application and it's composed of server states such as session data or database state and plus the local browser state, for instance the 'Cookies'.

And event is represented as event perimeter map and form state map and browser environment.

Event perimeter map is the perimeter map that will be passed to event handlers.

And form state map is the input values of HML form.

And browser environment is such as screen size or current time and so on.

So the event, initial event is represented as that one.

The type main means that event type is main, which is the pseudo event, which is called when the javascript is initial loaded first time, at the first time.

And form state map will be empty in the first place, and B in it is the initial default broader state.

And test input is composed of URL and entry state and sequence of events that I just explained.

**[5:55]**

URL, for instance, URL could be http://example.com/index.html and state entry is just a default state of the web application.

And sequence of events will be initially empty.

So, you can see that diagram state machine.

The S zero is initial state.

And E1, E2, E3 is event can be called from the state, initial state.

For instance, as E1 could be mouse click or, and E2 could be mouse holder event.

And in this state machine, state, I mean, you can transit between states by invoking event within that current state.

So, and this is how the framework will generate new testing inputs.

And first is by extending the current testing inputs, it's a new event.

Second is by modifying the test events in the current testing input.

And third is by searching from different page URL, I will explain one by one, next pages.

And one thing to mention is new testing inputs will be generated only when the resulting state, I mean, which means the next state hasn't been visited. Okay?

So in the first case, extending the current testing inputs.

Since you can see that test, t1 which is the first test case, extended the initial test t0 by adding E2 event to sequence of events.

And, so for instance, E2 could be mouse click event.

So this means this test case tests mouse clicking event in http://example.com/index.html.

And the state is currently S0, the initial state.

And next one is test input.

So this is, T2 is generated by modifying E2 event, which was mouse clicked.

So, say E2 was, E2 was mouse click, left clicking the mouth, then E2/1 can be right click of the mouse.

And finally we can generate another test case by starting from the different page URL, you can see that T3, we have transitted the URL to test.com from example.com.

## 🔊[9:02]

And this could be another test case.

I would like to describe how algorithm works, actually I didn't expand how we can actually variate the event, events, it is, it will be done by test input generator.

And prioritizer will make the test case which is important in the first, need to be executed first.

And I will explain this components.

There, actually the author devised the three types of prioritization functions.

First is default, which gives every test case the same priority, which means every priority get ,will get,I mean, will be randomly chosen.

And next one is P1 : Coverage, which utilized coverage information.

This function will prefer event sequences with relatively low branch coverage, which means, I mean, it's because executing the task case which, whose events are already covered is fruitless.

This function will choose the task case which contains the event sequence, which is not yet discovered yet.

(Professor speaking)

So, not exactly relatively low branch coverage, you mean traverses consist perhaps high coverage of things not already covered, right?

Yeah.

Okay, okay.

So, the coverage function, okay, the coverage function is defined as number of covered branch divided by number of discovered branch.

And I think the text is incorrect.

Contaminated, but this means the score of task case C is 1 minus coverage score e1 times coverage score e2 and to e end, every event sequences include in task case.

And P2 is, P2 utilized with read and write information and this function will prefer event sequences that shares the specific variable by reading or writing to it.

So, for instance, e1, if e1, event 1 writes a variable code A and e2 reads the variable A and do something, then this function will prefer that sequence, which, okay, okay.

## [12:05]

And the function is below, but I think you couldn’t identify what is.

It just counts the number of shared variables thing that event sequence and plus e1 is direct to avoid division by 0 era.

(Student asking question)

Okay, okay, yes.

Can you explain P2 again? What is the purpose of P2?

P2? Umm...

So, there may be some cases where the event sequence share the same variable, specific variable such as e1, event 1 writes variable A and e2 or may be event that occurs later could read variable A which was written by e1.

I think the author ‘s hypothesize that that kind of sequence could reveal more eras since they share same variable reading or by writing, do it.

Okay, is it correct? Yeah.

Do you guys understand what I mean?

(Student speaking)

Okay, and there are two types of input generator policy.

First one is default.

If this generator will use just randomly chosen variable, values for event parameters or form field values such as using 0 for integrate value or using form field values.

It's something like just dummy generator.

And, next one is dynamically collected constants, which means some event handles could, might contain already defined constant within those event handles and this generator will use them to make, make variate the values of event parameters.

Actually I don't understand, I don't understand why it is meaning, useful in, revealing task cases.

Okay.

(Professor speaking)

Anyone want to answer that?

Anyone have a suggestion?

## 🔊[15:00]

(Professor speaking : You don't want to answer that?)

Okay, next time.

(Professor speaking : One of the things that trying to do is, as mentioned in an earlier slide, pure random tends to run into problems because it chooses nonsense equal values.)

Yes.

(Professor speaking : So if you use values that are known to…)

Already used…

(Professor speaking : …known to have been use in already in some sets that you hope not to run into that problem, so I think that's what the examples are.)

Okay.

And by combining three types of prioritization function and two types of input generator, they obtain four types of algorithms.

The first one 'events' to randomly choose the test cases and randomly generate the test values which is most naïve one and the 'all' uses every information that considers in this paper.

Experiment is…

First, they gave the constraints to make the coverage computation and locating source lines easier.

At first, they used only non-compacted JavaScript code.

There is some cases when to minify the sizes of JavaScript file.

They sometimes actually they compressed the JavaScript code like, for instance, removing the line break or spaces but the authors just didn't use that such of code to make the locating source lines easier.

And next, there are sometimes web application could generate JavaScript code dynamically by using server site codes for instance PHP could just echo some dynamically generated script code, but the author also ignored such a case to make it easier to make the coverage computation easier.

And they also assume that the user can modify the server state, as I mentioned before, the state of web application is represented as the local browser's states plus the server site state.

So I think it is because of that reason but, like accessing the server site code is not always possible, so it might be the drawback of this work, maybe.

## [18:02]

There are Threats of Validity and first is Representativeness.

The author said that the tested application might be selected with bias.

And next is Size.

Maybe the size of tested web application might not be comparable to real-world web applications.

And here is the Benchmark.

They claim that the result by initial algorithm is significantly lower, but they say the initial algorithm is just loading each application's main page.

So I'm not sure to compare the performance to initial algorithm is fair or not.

Yes, initial is now passed generation algorithm.

It is just loading the main page, so I'm not sure, but okay.

And after 100 test cases, most of cases, code coverage is similar for the algorithms except initial.

You see in the coverage section, most of coverage has converged at their 100 test cases.

And errors discovered during the execution, they claim that the events and all algorithms found significantly more HTML errors than initial.

And here is the another graph.

Actually there are two groups of in the result of web application Benchmarking, I mean testing.

In the first group, the algorithm that used many features such as coverage or all algorithm converge much faster than other algorithms.

You see that for the case in AGX4, the all algorithm converged its maximum coverage only after executing only five or four test cases which is much faster than others, but there are also group of cases where four of algorithms just converge in same manner, similar manner.

In this paper they didn't explained why.

And here are Research Questions.

First one is 'Does the feedback-directed algorithm that uses only default strategy, which randomly chooses the test case and randomly generates the test inputs useful?

## 🔊[20:56]

The result was 'Yes'.

This is the case in events algorithm and event algorithm achieve 69 coverage, which is much better than initial algorithm which covered 38 average 38 percent.

And also event usually discovered more HTML validity errors than initial.

Okay.

And question two is 'With the same number of tests, what level of code coverage is achieved by each of the test generation algorithms?

Okay, according to the result, events achieved 69, and const also 69 and coverage and all achieved over 70%. So we can see that the algorithms that used more features showed bit more, a bit better performance, okay?

And the third question is ‘What is time required for test generation algorithms to reach the maximum coverage?’

They claim that it took only two minutes to generate 100 test cases.

And the last, ‘How many HTML validity errors and runtime errors are detected by the different algorithms?’

The proposed algorithms tends to discover many HTML errors, but runtime error is at most one.

And they explained that because the applications are heavily tested already, but I’m not sure whether this is true or not.

The conclusion is their contributions are proposing a framework for automated feed-back directed testing for JavaScript and they also proposed four types of practical algorithms for test generation.

And even the simplest algorithm events achieved much better coverage than the initial algorithm and extended algorithms achieved slightly better coverage than events algorithm.

And this is end of my presentation.

**[23:39]**