웹소프트웨어 신뢰성

Test Oracles and Automated Testing

- ✓ **Instructor: : Matt Staats**
- ✓ **Institution:** 한국과학기술원
- ✓ **Dictated:** 김나정, 김민겸, 김성도, 문혜린, 박현서

[0:00]

Hi. So, my name is Matt Staats. And I'm your substitute teacher for today.

So, I'm not really a teacher. I don't do this very often.

And I think it's kind of silly the way most teaching gets done and so, for engineering.

I'm not a mathematician.

I'm not a physicist.

I'm not trying to teach you how to do something so, that later on you can do it yourself, especially with something like this which is really, you know, a pretty far out there research paper, if I can say so, myself.

So, instead, what I expect today is a discussion.

All of you will talk and answer questions in English.

And so, I have some slides here that go with this paper that I did, but it's not really designed for a classroom.

So, the goal is, I have a list of questions here that I would like you to think about.

And so, we're going to go through until I'm confident[?1:03] that everybody has thought about these questions in the context of this work.

So, are you guys familiar with Socrates? He is a Greek philosopher.

And he had Socratic Method of teaching, and which he would basically just ask questions and probe someone's thinking to trying get them to understand some sort of concept or to think about some sort of concept.

I'm not very good at it.

But I think it is the best way to do this typo work.

Okay, so, who read the paper?

Seriously, who read the paper?

Who understood paper?

Like, Okay.

All right, so, what do you guys think a test Oracle is?

Pablo, what is a test Oracle?

(Student Speaking)

Okay. Right answer. All right.

Jamin, give me an example of a test Oracle.

(Student Speaking)

Right. Okay, that's very academic.

Give me a concrete example.

Okay, so, you worked in industry, correct?

When you tested your software, what was the test Oracle?

(Student Speaking)

[3:05]

So, your test cases were written in what language? Was it JUnit? Was it?

(Student Speaking)

Okay, so, your test Oracle is an expected result.

(Student Speaking)

Okay, and you wrote your test cases by hand.

(Student Speaking)

Manually.

Has anyone ever done automated testing?

Like actually done automated testing with the purpose of detecting faults.

Nobody?

Okay.

(Student Speaking)

And?

(Student Speaking)

Okay, so, your problem was it broke.

All right. So, what is automated testing then since you've done automated testing?

(Student Speaking)

Okay, so, how do you know if it's correct or not?

(Student Speaking)

Okay. So, that's, where do these logical formulas come from?

(Student Speaking)

Okay, so, if I was going to automate the testing of my program, I've got my program, and I've got a tool that tries to generate test inputs.

And then what he's saying is that then you're going to see if these test inputs, these test inputs match up with some formulas.

[6:01]

So, where do these formulas, where would they have to come from?

(Student Speaking)

I mean, okay, so, you've got some tests here.

When you talk about verifying a system, you have some sort of specification conceptually of what the system should do.

And you've got some set of supposedly talented developers that will then go and build the program.

And so, then your big question is, does the program represent the specification?

And what you're saying is an automated testing, you're going to automate this question, correct?

Automate answering the question.

(Student Speaking)

That's what you're supposed to do.

And so, what I'm saying is, where does this specification, we know where the program comes from.

Where does this specification come from?

(Student Speaking)

From where?

(Student Speaking)

Stake holder.

So, what language does the stake holder write the specification in?

(Student Speaking)

Natural language, and so, when you do the automated testing, can you use natural language?

I don't know your name, I'll just point though.

Can you use natural language?

(Student Speaking)

No.

(Student Speaking)

Okay. Okay. So, that's at some point somebody takes this and essentially happens twice, right?

Cause you have it in natural language, and you converted into a formulism which is the program.

And then you have it in natural language, and you converted into a formulism which is your formal specification.

And from there, you can automate your testing process.

Okay. And so, just to go back and talk about some of the things we haven't been

talking about before.

You've got your specification, you've got this program which results after somebody writes it.

We're going to have test inputs.

Maybe you write them by hand.

Maybe you generate them using a model checker.

And then you can do the running part, you can all these pass to the program.

Now this is the simple bit[?8:19].

Everybody gets conceptually how to do this.

There is lots of tools to make these.

And there's lots of tools to automate this.

And of course you have to do this.

It's when you get to the Oracle, and you have this artifact that needs to check.

If the program passes or fails, that's where things sort of fall apart.

And so, basically what this paper was about was nodding that when we talk about testing a program, or really when we talk about verifying a program, you've got all these different pieces.

(Student Questioning)

[9:03]

So, when we talk about verification, verification is the act of answering this question.

Does the program do what the program should do?

And there's a validation which is, I guess verification is does the programs do what the specification says it should do.

And then there's a validation which is does the specification state what the program should actually do.

And testing is just one form of verification.

So, you're going to have, it gets fuzzy when you talk about automating it.

But normally you're going to have a finite number of concrete or pretty close to concrete inputs that you're going to run over the program.

And each one of those will individually be checked again some sort of test Oracle.

So, when you talk about model checking, it's not what you do.

You do some formal proof, some very mathy[?10:01] thing, so, at the end you can say a hundred percent certainty this implies this.

But what testing is just, it's like if you're building something I don't know assembly line?

How do you check if it works?

You pull off something, you pull off like, I don't know, what is something you would build it, the stupid pepedo crap[?10:24] must be built on an assembly line somewhere.

And there's a person whose job it is to eat pepedo, everyone in a million pepedos, he eats one.

And then he says he's fine.

That's what testing is, this is eating pepedo.

And to model check it, you would eat all the pepedo.

You're very fat person.

All right, so, you've got these full artifacts, right?

And so, you can think about potentially how these artifacts relate.

And when you look at some of the testing work, you know you've got a program which is going to implement this specification, and if you have code coverage, you can use how the program is built to generate test inputs.

Or you could design your program for testing or you could have a specification which represents what the program should do.

And from that, you could drive your test Oracle, or et cetera, et cetera.

The point is that we have these four artifacts and testing, and there's a lot of ways to think about how this tool could interact.

But when you actually look at the testing research, you guys have been in this class for what, two months now? Three months?

All right, what have you been studying?

Like give me a topic that you guys have studied in this class.

Come on, you don't remember? This is terrible.

You guys should demand a refund.

Pablo, give me a, that's why you sit{?11:57} in the front.

(Student Speaking)

[12:22]

Right. Okay. So, everything relates to the test inputs in the program basically.

Okay. So, this is kind of what I noticed as well.

It's that when you actually look at what you do in the testing research, almost all of it focuses on the test inputs.

Right. So, why does it focus on the test inputs?

Why do you think that most of the research focuses on the test inputs?

You have a thought[?12:52], or you have guess, one of the two.

(Student Speaking)

I think this is right.

I think when you talk about what you can do automatically what you can build tools to do.

You can build tools to measure code coverage and generate test inputs.

Why are there less tools for generating test Oracles?

(Student Speaking)

Okay, that's a good guess when nothing exists.

Why are there no flying cars?

It is hard. More specific, why do you think there are no tools for test Oracles?

(Student Speaking)

Correct. I think this is right.

So, when you talk about your testing, you sat down, and you wrote your test inputs.

We can automate that, it's all simple, it's random or whatever.

But at some point, you have to say what the system did.

Where did you figure out, how did you know what the system was supposed to do?

How did you know what the system should output?

(Student Speaking)

So, you got what the program should do from what the program did do?

[14:50]

(Student Speaking)

Based on the specification.

And the specification was, in what format?

(Student Speaking)

Right.

So, basically, you read something.

And then you said based on this, I think it should probably do this.

So, this is why it doesn't tend to be automated, right?

Because you are in the [?15:30] to copy something, and turn it into some really big formal math thing, or you do it just point by point.

So, you have a few manual tests with manual test oracles attached.

Okay, right.

So, going back to the automated testing, what type of faults can you detect with the automated testing?

Since you work in Moonji's lab, then you've known this more than anyone.

If you don't know the answer, I will tell them.

And you'll get fired.

(Student Speaking)

System crash.

(Student Speaking)

Right, right.

Exactly, exactly.

So, the type of faults you can detect with an oracle.

This says the just general properties you expect all programs to have.

Don't crash. Don't access memory that's not yours.

You know, don't delete the C drive, these sort of things.

Whereas the stuff that relates to what the program should do, a person has to sit down in any way.

So, anyways, we looked at this.

We said most of the work tends to focus on the test inputs.

And a lot of the work tends to ignore this.

And there is good reasons for that.

And I think in the paper we mention there is good reasons that we didn't, we don't do that.

But there're still problems.

It's one thing to avoid doing something because it's hard to do.

And it's something else to pretend like something doesn't even exist because it's hard to think about.

And when you do a research from this type of prospective, and which you just pretend that something doesn't exist, you tend to get tunnel vision.

You just forget it exists.

And so, you know, if I have my keys here, I got a lot of crap in my pocket.

If I have my keys and I'm walking home and I drop my keys, I have to go find them.

[17:56]

So, one thing to do will be to look under a light post for my keys.

And the reason you might do that because it's easy.

It's easy to look under a light post.

The problem with looking under a light post is extremely unlikely that you actually find your keys.

And so, in some sense, research which does and looks at only one thing, is kind of like looking under the light post.

It's just easy.

And so, what we thought we should maybe broaden our prospective and see what happens a bit, okay.

So, that's kind of what this paper is about.

It's about broadening the prospective and zooming out from the tunnel vision that [?18:39]

Now, you guys all read the paper.

Why is this paper important?

It's very narcissistic for me to say that.

Why is my paper important?

Tell me.

But it's one and the words.
It's somebody else that wasn't important to.

Why is my paper important, Pablo?

(Student Speaking)

Okay, something else?

(Student Speaking)

Okay, now it works.

(Student Speaking)

Everyone's go.

You go last cause I haven't asked you a question yet.

(Student Speaking)

Jimin.

(Student Speaking)

[21:26]

Okay, Kyuchul.

(Student Speaking)

Why?

(Student Speaking)

Well, of course.

But why do you think it's more important?

(Student Speaking)

Okay, that's fair.

Okay, that's right.

Basically zooming out, trying to convince other people that there was a problem.

So, every time I talk to somebody about that there is always a something different opinion.

Some people hate it.

Some people really hate it.

I got a nasty review when I submitted this.

But basically the idea was you know there's a problem like this.

And then you want to tell other people that there's a problem.

And if this was, if I was not an academic, this would be a blog post.

This whole paper would just be a blog post.

It would be two pages long.

It would have half as much as stuff.

But nobody takes a blog post seriously.

So, what I decide to do is try right, a research paper that pointed out these things.

And so, you end up having to expand it, and make it all very mathy.

And that's what the formalisms come and try to convince somebody to convince people working in the field to move from this to this.

So, same reasons, plus extra crap to make a research paper, all right.

So, now the whole reason, the rationale behind carrying about these things.

You guys have already attached upon.

But, there is really two reasons I see.

One is that if you have these artifacts, then you are not really paying attention to.

They represent unexplored areas.

And by exploring these areas, you can develop techniques and methods that can improve the effects in this testing process.

So, when people looked at symbolic execution, you presented already?

[24:23]

So, you presented symbolic execution.

So, that concept, a symbolic execution is from what year?

(Student Speaking)

That's not, years decades, but it's correct.

1979, there was a paper on symbolic execution.

And so, people have really not looked at that seriously for a long time.

And so, 2001, 1999 I forget when.

(Student Speaking)

Somewhere in there.

People, when back they said we've got all these methods here that can solve some of the problems that we've been trying to do.

By applying them to testing as it exists now, it's easier to cover branches.

It's easier to explore the internal structure of a program, by taking those the improving testing effectiveness.

Things like overlooking the impact of the test oracle, overlooking the impact of the program structure on a testing process is kind of the same thing.

You can go back, look at them, hopefully improve thing.

So, opportunity, this comes from my Ph.D. dissertation, I've got to have a lot of pictures.

And the other reason is, and I think, Gregg said that you guys have been pretty good at this.

When you looked at whether or not something is working effectively or not.

As a scientist, as an empiricist you have to look at all the pieces, right?

So, for example, you do your study, and I don't know study like this.

Well you have students come in and do something.

The fact that students are using a tool as opposed to professional developers or just like normal people over the street.

Is going to impact your results?

So, you have to be honest and upfront and say "I have students, they have these characteristics because of that the world may not seem like my study."

Now before I mention that they weren't, and then you guys know this, they weren't not only, not developing techniques based around test oracles and programs and interrelationships.

They were just ignoring them.

Just entirely not discussing it in the context of their study, and so, because of that you have an uncontrolled factor.

And so, in the end of the paper, where should say we use test oracles and blah blah blah with the specification blah blah blah.

That was missing.

Which mean that we didn't understand how there other factors can result in misapplication of methods.

We didn't understand how these other factors, impact the effect in these things we have been proposing.

Which is a serious issue in a scientific research.

So, there was work on stomach ulcers back in like to 80s.

[27:20]

And they develop this method of curing stomach ulcers.

And they would stick it to down your throat.

And they would pump warm water into a tube.

And then this would curse you, also, it decreased by some statistically significant amount.

And so they conclude it.

I mean it's not a great idea, right?

And it laid there for two hours.

And then you get up, and you're supposed to be better.

And what they found later on is that if you just had somebody come in and pump down cold water, you will get the same of act.

So, it ended up being placebo really if you just had somebody take a pill after going to the doctor, they will be less stressed, and their stomach would feel better.

And so, it's similar thing that you know we never proposed to developers that they stick it to tube down their throat and pump water in their stomach.

But we proposed some pretty extreme things without really considering how other aspects may impact the effectiveness.

Okay.

So, these were the problems.

This is basically with the paper.

I've already touched on this, but the big question I had is "how do you persuade a

group of people that are older than you, that they think smarter than you, that did all the research in the area they overlook something.

And the way you do that I think, as you go back to the stuff they did a long time ago, that they used to originally motivate the work in the area back in the 70s, 80s.

And you say, you made a mistake here, and the nice part about this very old work is that most of these people are no longer in the field.

So, you don't insult anybody.

Very important.

So, you go back to there, and then you demonstrate that things were wrong.

I'll give you a sec.

Did you get the joke?

Anyway, physics joke, strange theory.

Okay.

So, this is part of it.

This allows you to go and take things from a very high level and write a very short paper that tries to cover a lot of stuff.

[29:43]

And in the other bit, it's nice to actually provide evidence, that what you think happens actually happens.

Because this early work of actually got abandoned, and those people left the field in part, because after you did this theory stuff, at the end of it you couldn't say anything.

We are going to go through that in a bit, but you really couldn't say much a value.

So I you want to do empirical studies.

Of course there are problems with empirical studies too.

In soft engineering, you tend to have very little data available to you.

But at least you want to try to provide some evidence.

So I do both of these things, not on this paper, but overall.

Ok, so let's talk about the theory of testing which is really what this paper is about.

Ok. All right, so, way back in is 1983, no, 1981, I don't know, anyway is a long time before I was born.

Good enough and Gerhart, decided they wanted to define some testing was like a very new research area.

They don't even doing it for five years or so.

And really in soft engineering they don't even seriously talking about it for 10 years.

Before, they did really talked about proving it correctly, and it was so simple that you can almost go through a big chunk of the input.

So, anyway, way back when they said, what is testing?

Testing is bla bla bla. What do we want in a testing?

Ideally, we would like to select the set of test inputs such that for every fault in the system, if it exists, we can detect that exists, and we can kill it.

This is the goal, right?

Now, of course the problem we're saying you are going to detect all faults through testing is that theoretically they reduce the whole thing problem.

So if there's an infinite loop, in there you need to detect infinite loop through testing, in order to do that you need to have a test which can detect infinite roof.

It's a whole thing problem, you have to demonstrate the thing that whole [?32:00], which is hard to do or impossible to do with the final number of the test inputs.

So somebody said that these people said this work by this guy is impossible.

But all of them liked the idea.

You should have a formal foundation for testing, just like you have formal foundation for physics, and for linguistics, and for all these other things.

And so that came up with sum.

And one of them, in particular, by Gourlay, the finder concept of test coverage criteria, and power.

Who knows what power is?

Did he probably didn't talk about it in this class.

Ok, so power says that if you have two methods of selecting test inputs, A is more powerful than B, if whenever B is guaranteed the detective fault, A will be guaranteed the detective fault.

[33:00]

So, branch coverage is more powerful than statement coverage.

Because of all scenarios where statement coverages guarantee the detective faults, branch coverage is guaranteed the detective faults.

What's the problem with that?

Let's think for a second.

All right, so, the first question to ask is when its statement coverage guaranteed the detective fault.

Under what circumstances?

Pablo?

(student speaking)

is statement coverage guaranteed to the detective faults.

(You have to be able to get the all statement?)

As soon as possible.

Assume you can cover every statement.

What false are you guarantee to detect?

(student speaking)

You are guaranteed that will execute.

How will you know if it's wrong?

How can you sure to know that it's wrong?

(student speaking : I cannot be sure)

You can! Most of the time.

So if you have the statement that something like zero divided by zero, that will crash every time.

And as long as you execute that statement, you will going to detect the fault.

But very few programs are guaranteed to fail under any circumstances.

It's a bad program like, you know, if access memories that's not yours that every time then it will fail every time.

So the problem with this is that if B is statement coverage, B is not really guaranteed to detect many faults.

And, so A, branch coverage is guaranteed to detect the same fault that B is, but really it's really crappy guarantee, right?

It's sort of like saying every time it rains [?35:37], my hair turns red, you know, like, it never rains[?35:39], it's a meaningless statement.

And so this is being not meaningless, but close to meaningless..

And so somebody says that, they say, ok, so we like what they're saying in here, it's wrong but we like it.

The concept of coverage criteria here that he defines kind of make sense.

And so this is the lane why you [?36:03]

So she says, this is wrong and here's why.

And then she goes to these big lists of all these theoretical methods of comparing coverage criteria, and she finds that from the air 1983 and 1986, all of them have sort of a similar problem.

The end of B is really weak, mathematically correct but practically insignificant guarantees.

Because, and she is having to define very strange made up coverage criteria.

🔊[36:33]

So even come up with examples in which they would be somewhat useful.

Ok. So she says, crap all this others crap, well, it should do, instead is compared coverage criteria probabilistically.

So A is probabilistically better than B if an average A will[?36:53] better results than B.

So what is that mean? What is that mean to be probabilistically better?

(Student Speaking)

How would you measure it?

(Student Speaking)

So what will you say measure individually, what do you mean?

(Student Speaking)

So, I don’t know how to say this, what are you doing for your project?

(Student Speaking)

What are you doing for your project in this class?

You are answering the question, what was the question you were asking?

(Student Speaking)

Your project question, yeah.

(Student Speaking)

Versus?

(Student Speaking)

And so you want to find out if using an applications specific in variant is always better?

(Student Speaking)

Usually better? Or can be better?

(Student Speaking)

Right.

(Student Speaking)

So it's just usually. You want evidence, right?

So probabilistically better. And usually better. It’s the same thing.

And so what were you doing for your project is an empirical study comparing A and B.

That's what you're doing, this is all you’re doing.

So probabilistic, you know, she goes on to give some formal definitions, and then she says she'll never be able to computer in practice, cause you can’t.

You can only do it for your individual cases.

And so, the whole field dies right here. This is like 1989 or something, I don't know.

The whole concept of the [?40:25] dies, because when [?40:26] with cycle, and when somebody would suggest an idea about how you can theoretically talk about different methods of testing.

And then somebody would refute that idea.

And then somebody else would come back and say I'll fix it and et cetera, and eventually the whole thing falls apart.

And it's not nobody really does this, and hasn't for many years.

So, [?40:52] call this, I cycle of failure but there's like a foot note in a text book by [?40:59] talking about [?41:00] and he says,

Yes, I am ok this ideas but it doesn't really work in practice.

But, the thing is, this is where all those definitions that you guys have been studying, all those core concepts will lay down by this work here.

Even though really what they were all these goals they had never actually were achieved.

But they have the nice definitions that came out of it.

And so you can see like this is where the initial concept of coverage criteria comes from.

And this is where the concept of limited effect somebody finally says you will not detect all faults with testing, it will be impossible.

The concept the [?41:45] so [?41:46] means that something always grows or stays the same.

And so when we talk about branch coverage, and you have a stronger coverage criteria, you almost always need more test cases.

[42:00]

And if you have a set of test cases and you add test cases to it, you always improve coverage or stay the same.

And so this is being important for a lot of [?42:11]methods detesting.

So for example 재민 has done work on concurrency testing in software.

And so you've got this coverage criteria, right?

And so how do you find tests to satisfy this?

(Student Speaking)

Right, so basically you just run for a while. And want you stop, you say you've done as good as you can.

Because you know that if you run longer, is not going to get worse.

And so actually having somebody say it sounds very simple. But actually having somebody say that give us this word that we used to talk about testing.

Partition testing is, you know, if you have all these branches, you have this enormous set of potential test inputs, you can divide them.

Along what branches they satisfy.

And so you end up meeting the pick one from each set, and this results from mathematical stuff ends up being important.

But it also gives you a way to think about actually generating test, I mean one from everything.

So there is a concept in linguistics called some hypothesis.

And it's states that the way you think about the world relates to the world you have in your head.

And so, I think in English, and most of you think in Korean Pablo, do you think in English or Spanish?

(Student Speaking)

So this is Spanish going by, ok.

[43:50]

So, because of that, it shapes the way we think about the world.

And so you know, '~습니다's, '~입니다's, '~이에요's, '반말' and '존댓말', you guys think about things differently than the way I think thinks up here.

And these words are the things we used to talk about 'testing'.

So, they should shape the way we think about testing.

And a big part of the reason, I don't think there's any work in Test Oracles or specification, or programs, it's because we never talked about it.

Back, when we were laying down, the words in the textbooks that the students read to develop JPF, and concurrency testing…

All that crap came out of here.

And so, I thought we should go back, just stick this in here, and maybe people will start thinking about test oracles because it'll be in those books.

Maybe I have a footnote in the book.

So, I went back to the big famous one.

This one has like 500 citations or something.

And it's where the concept of coverage criteria came from, and it's the one if you want a paper that references these definitions, this is the one you cite.

1983, it's the same year I was born, "A mathematical Framework for the Investigation of Testing".

Alright, so, we are going to fix this.

Now, we are going to talk about 'Framework'.

It's very boring, but we are going to talk about it.

You've got basically 3 artifacts here that they are concerned with.

You've got the specification S, the set of programs P, like all programs that you can possibly write, and the set of test T.

And they define predicates, which relate these things.

So, you have a correctness predicate, which is going to be true if P is corrects with respect to S.

You have an OK predicate, which will be true if you say that test T passes over program P with respect to S.

And this is basically the test oracle, right?

It's what determines if you pass or fail.

And then you have this relationship between the 2.

If it's correct, your test should pass.

[46:10]

And the problem is that it's sort of a 3 legged stool right here.

It's kind of wabbly, because there's no test oracles.

And so, there's a few problems coming in.

One is that there's no notion of partial correctness.

So, if it's correct, your test pass.

Your program is not correct, if your program is not correct, what can you say about your test oracle?

Mathematically.

(Student Speaking)

Why?

You know the answer I want, but you don't know why.

That's important.

If Correct is false, what can you say about this formula?

(Student Speaking)

Because?

(Student Speaking)

OK. You know the word 'antecedent'?

In this logical operation, this is the antecedent.

And if the antecedent is false, you can't say anything about the implicate.

I think that's the word.

So, anyways, the program is always false, right? Cause you are testing it.

So, this ends up being meaningless.

It's not a good relationship to base things upon.

The other bit is that this test oracle, this OK predicate is fixed, mathematically.

In the paper, you know it's like one of these really dense papers where they don't even put spaces.

There's no figures…

They don't even have law tag or word or anything.

And so, early on in the paper, he defines this, and then he never talks about it again.

So, the whole framework is basically designed to let you ignore that this even exists.

And the whole thing ends up being about the test inputs.

So, you can't vary the test oracle, you can't even really talk about it because it's fixed.

And so, we go, and the first thing we do in the paper is, we fix this.

And again, this is not exciting, but you know, we add a set of test oracles, we add another predicate that allows you to relate the test in the programs and the specifications.

And then we have, say, if it's all correct for all tests, and then it's correct.

And then you also, have this set of oracles here which can vary.

[48:43]

So, from this, then comes from the rest of the paper.

What did you guys think after section 3?

Oracle comparisons, applications to previous work, etc…

Just refresh yourself for a second.

So, as far as I'm concerned, this paper, the interesting part of this paper ends after section 3.

Section 4 and Section 5 are filler to make it 10 pages long.

The reason is, all we cared about is talking about how this was wrong.

Once you find the stuff, then you can go on and you can show some things.

One part of this is we formalize concepts related to test oracles in section 3.

That part is nice.

So, for example, in a lot of work related automate testing, they always assume that the test oracle is correct.

(student name)'s study is basically looking at program invariance.

And some of the program invariance, they are going to be incorrect.

In the study, people are going to have to sit down, in front of program invariance, which are incorrect, and try to make use of them.

That's something that work like that tends to just assume a person's perfectly capable of.

So, we talk about that.

The test oracle might be wrong.

It's going to be different levels of power.

And then the other bit, where all these definitions come in, just gives the way you talk.

And the other bit is we explore the implications of test oracles on previous work.

So, this relates the opportunity and the danger I talked about earlier.

Here are things we could do that we've kind of overlooked.

This is somewhat of what Pablo's doing, basically.

Correct?

Pablo : Basically.

Right?

You are going to have a fancy way of doing things related to test oracles before people would just not even bother with.

And this explores what happens when these things try to flapping around and changing and how that impacts the effectiveness of the testing process.

And this is kind of work, (student name)'s work comes in.

I don't want to go through it, because it's quite boring even to write.

So, this is the ‘Theory in practice’ part.

This is where we talk about all that stuff.

I think section 4 and section 5 of my own paper are boring and unnecessary.

So, why should I bother to do this?

Why do you think I sat down to write down this?

Why do you think other people were interested to read it?

[51:50]

(Student Speaking)

What do you mean?

(Student Speaking)

Right.

(Student Speaking)

OK. You’re right.

I mean, that’s kind of what I’m going for.

It’s trying to show value.

Now, you are convinced to the value, based on section 4 and section 5.
Be honest.

(Student Speaking)

OK. Who agrees with the proof of concept idea, that you a proof of concept to demonstrate the value of what I argue for in this paper?

Anyone? OK. Why?

OK. What? Give me an example of proof of concept.

What would convince you?

(Student Speaking)

So, I agree that you need some sort of proof of concept, and I think it needs to be

actually stronger than what is in the paper, because what is in the paper…

I mean, I end up defining things that no one would ever define in real world.

I think I even had a reviewer, say, I don't believe that you can't define it, but I can't do it either right now.

So, it ends up begin very, very, very fairly.

[54:20]

So, instead, to actually argue that this thing works, I need the appropriate way that steals him and move him to empirical studies of real systems and show exactly what I argue that as the test oracle changes, the total effectiveness of the testing process will also, change.

That way, you get sort of a 2 prolonged approach.

You've got your theory bit, which is very fun, it fits in 10 pages, and then you have actual charts and graphs and things showing that… at least for these systems, it does matter.

So, in the paper, I defined this concept of complete adequacy criteria.

And basically, in adequacy criteria, from here on, says that your testing process is good enough if for some program, some specification, and some set of tests, you've satisfied some criteria.

In this case, for branch coverage, it would be for this program, and don't care about the specification, this set of tests, needs to make all branches evaluate to true and false.

But it does really care about what you do in the test oracle.

You could have a testing process which satisfies branch coverage, but only catches faults when the program crashes.

And that's different than a testing process which satisfies all branches and verifies that every single output is correct.

That's all we wanted to say.

What I have here are 4 systems from the avionics domain.

So, I'm hired by a web-science, but my background is actually in critical avionic systems, airplanes, basically, not web stuff.

Not even close.

But, what we did is we did an automated testing, so…

Do you know what Karnaugh example based test generation' is, (student name)?

Does anybody?

Student : Excuse me?

Karnaugh example based test generation.

Do you know what that is?

(Student Speaking)

[54:20]

It's not something you would do in web-testing, but if you have a small system like this, you can actually run a model checker.

And a model checker will examine every state of the system.

And you can tell the model checker, for every branch; this branch cannot be made true, this branch cannot be made true, this branch cannot be made true…

And the model checker will disagree.

It'll give you a Karnaugh example, saying you are wrong, this property doesn't hold, here's a Karnaugh example, demonstrating this branch can be made true.

So, what you can do is you can trick it.

And they're giving you a set of test that satisfies some coverage criteria.

So, what we do is we trick this tool and they're giving us 100% branch coverage. OK?

So, they should be really good tests.

That's the important part.

They should really good tests.

And then we use an Output-Only Oracle, that checks all the outputs for the system to see if they are correct.

So, it should be a pretty good oracle.

This is what people talk about or think about in academia when you talk about an oracle.

Somebody actually sat down and stated that every single output is correct.

And so, the fault finding, for branch coverage with this type of oracle, basically this whole criteria sucks.

You will catch half the faults, 15% of the faults, right?

So, this system here is the thing that makes sure that the landing gear is going to come out.

That's really bad.

That's really bad.

If you move to an oracle which looks at the internals and outputs with branch coverage, now it jumps to no lower than 83%, which is not great.

There's 1 20 chance that you won't land, but it's better.

The whole point is to illustrate that moving this in practice can also, change the effectiveness of the testing process.

[58:34]

And then the other thing is that…

I just love these figures.

So, you can look at how vary the number of test inputs impacts the effectiveness of the testing process.

And this is a big part of what all the work with random testing does, and like JPF.

Those show that you can get more, more, more test cases, and the assumption is that as this goes up, fault finding goes up.

But the problem is, you've got no automated way of getting a test oracle, normally, right?

So, you're going to imagine a tool. You guys worked in industry.

You have a tool, and it gives you a bunch of test cases, and then what?

What would you do with those test cases?

I mean, you can run and see if it crashes.

But for actual semantic faults, what would you have to do?

(Student Speaking)

No. It didn't work.

Come on. You and I did a study.

Alright, so, you can do automated invariant generation, and then a person cleans up the invariance.

The alternative, the other alternative would be for a person to sit down for every test case and write what should happen at the end of that test case.

And so, this goes up, the amount of effort for a person also, goes up.

[1:00:07]

So, if you move from one hundred test cases to two hundred test cases, it's going to take like this would be, I'd better take it two hours to generate test Oracles for a hundred cases.

It's four hours. Sounds like half a day.

(Student Questioning)

Yes, the accesses it would be your size Oracle.

So, if person's going to use an automated testing approach, there's kind of like two things they have to work on.

For every test case they have to write down what the tests should do and they have to write down a certain percentage or a certain amount of effort needs to go into constructing a test Oracle.

So, if I, for every test case, specify one output, that's not as much work.

As if I, for every test case, specify a hundred percent of the outputs.

All right. So, both of these are access of effort, essentially and as you go this way, you have to go this way from pointing with should do nothing, you have more to do.

And the point is that, when you look at what the most effective, so, this is the contour map.

So, why is testing access the Oracle Z is the number of faults. So, this dark area, you detect no faults.

This light area, you detect all the faults, and these lines represent constant levels of fault finding.

You're basically walking up for fault finding hill, and so, you're going to see this line here, represents a certain constant level of effort.

Okay, so, you could do a hundred and twenty five tests with a very simple Oracle or you could do almost no tests with an Oracle that captures half of the state, and you will get better fault finding here.

[1:02:06]

The lighter part. Here, same function doesn't matter.

Here, it does matter.

But not a whole lot.

Here, it's actually better the focus on having a strong test Oracle instead of a lot of test cases.

And so, before when I talk about comparing methods, the probalistically better.

Method A versus method B they always assumed that the test Oracle is fixed, and so, you do this empirical study, and you've got one kind of test Oracle, and you don't tell anybody what it is this normally.

Like very rare to see a study which somebody says where it is.

Normally we have to walk up to them at the conference and ask.

And so you assume that that's going to whole but depending on what system you have, what the testing process is, what you're… if you have really strong invariance, then you might be [?1:03:04] scenario like this.

If you have really crappy invariance, and so, the guy just going to write this kind of scenario. But you don't know from the paper.

So, just highlighting that this thing as fluctuate can really change conclusions.

There was work in medicine and they find out they give this type of medicine for your heart and it works well.

There should black and it doesn't work very well at all.

So, they have to get a different kind of medicine, you know for black people to not have heart attacks.

As it is kind of same thing black guy there, so, white guy there. They need different medicine.

Right, so, you see the point, the optimum point very depending on where you at.

We moved from, the whole point of this paper is to move from this type of conceptual frame work to this type of conceptual frame work.

To this one, was wrong and hopefully I've convinced you they're having model, a view of testing which is focused on entirely what you're testing puts are, which doesn't consider how you figure out the program works or not.

How your program is structured at.

We haven't talked about this much but how your program is structured.

What your specification is, what a person has to actually do to make this work and practice is wrong.

And we try to move to, you never going to have a model of the world that's correct.

But you're going to at least have this model that's not as wrong as the previous one and that's what we're trying to do here.

Take away message would be testing effectiveness is influenced by a large number of factors.

There've been benefits to looking another things which a, Pablo, get your hopefully be, be a reaping the benefits though and they're dangers.

And looking at things the wrong way.

[1:05:00]

You overlook things at matter.

Things at impact the effectiveness of the process.

Alright, so, do you guys have any questions?

I mean I guess, the final question I have on here that I haven't asked is how can you improve.

So, when you think about software testing and what's out there now.

How can things get better and what are the hurdles to doing so.

So, I'm going to ask that question.

Pablo, what is the major hurdle in moving from this to this?

Method of thinking, method of doing things.

(Student Speaking)

It's implicit, right. So, there's an Oracle in the sense that if the program crashes, it's a failure.

Right, I mean it's more, it's just, it's not such as model. You're right. There's an Oracle, it's implicit.

But when you read the research papers, and you've read a lot of papers on testing cause we've been working together for a while.

How often do they talk about the test Oracle?

Never.

So, it's a method of thinking. So, it's implicit.

And in the paper, it's implicit, too cause they have some, they have papers that talk about fault finding.

But it's not at the forefront.

It's not something explicitly addressed even though something that does exist.

(Student Speaking)

Right.

Right. But what is the hurdle?

In moving from doing research that things like this to research that things like this.

(Student Speaking)

Right. It's true. It's true.

And the reason is, nobody's pointed out that I'm stupid yet.

I mean another reason is like what would you like me to add.

There's always extra factors to consider.

(Student Speaking)

[1:08:00]

Yeah. It was my advisor, yeah.

(Student Speaking)

Right.

(Student Speaking)

Alright. So, what do you mean by overcomplicated?

(Student Speaking)

To actually do it, so, the hurdle would be cost.

I think that's a good answer, what do you think?

(Student Speaking)

It's… that's an excellent point. The answer is no to the best my knowledge.

The problem is, with testing in general is that you kind of have like…

Alright, so, you have a program, and you want to know if the program's correct or not.

So, you write a test Oracle and you don't know if the test Oracle is correct or not.

So, kind of Oracle for your Oracle and you don't know that one.

So, it's like a, it's turtles all the way down, type of scenario.

I mean the answer, so, the closest thing that I can think of is that there was a suggestion for what they called 'inversion programming' a long time ago.

[1:11:00]

And you know what you have like the space shuttle or whatever an aeronautic system like a really important system though have backups.

And so, on the shuttle for example, there would be three things, four things I think.

They world all work together, and then they would vote on the result.

Because in space you get weird results sometimes because you get space radiation that make sure bits flip.

And so, they would vote every time that they have competition.

And so, that way, you don't know the certain probability that one thing fails but the probability of four things failing a one's becomes less.

And so for software they suggested that what you could do is have three programs, be written by three different teams and then you vote.

And that outcomes the answer and the idea would be the same as hardware.

And that likelihood that all three of them would be wrong in the same way was not high or two of them would be wrong in the same as the high.

And actually what happens is that assumption was wrong and so the programs means that being if you have somebody.

You have three different groups of people write a program they get it wrong for the same reason or some tricky part of math logic and they do the tricky part wrong every time.

And so, the test Oracles you could imagine applying, you're going to write the test Oracle three times, you know three different guys sit down. Tell me what does means.

This English part, in formal whatever and they do it.

But the properly all get it if it's going to be wrong the properly all get it wronger will be close to wrong.

And so, one thing I have kind of wonder is that if you didn't have people write in an add program and they get the program wrong all the time.

So, you maybe write in a program and then you do concrete testing.

And then you do automatic invariance in those all three different, very different formats. Right?

It's like writing a paper in German and Spanish and English.

It seems like one of them should be right or two of them should be right.

But the answer is no.

And but it's a very good question.

It's a fun thing to think about.

So, what do you think, what's the hurdle?

You're working with test Oracles now. Why can't we not just move from there to there.

(Student Speaking)

This class ends by the way.

All right, well okay, the class is over.

Those were two good answers though.