**CRAIG:** Hi, I'm Craig Smith. And this is Eye on AI.

**CRAIG:** This week. I speak to Peter Schrammel, one of the founders of Diffblue, an automated unit-test writing software company. Peter spoke about the increasing automatic generation of code and how he sees such automation increasing the productivity of developers.

**CRAIG:** Before we begin. Let's take a moment to thank our sponsor ClearML, an open-source ML ops solution. You can give them a try at clear.ml. Tell them Eye on AI sent you.

**CRAIG:** Now here's Peter. I hope you find the conversation as interesting as I did.

**CRAIG:** Peter, introduce yourself, give some background on your education and your particular area of study. And then we'll talk about automated code generation and unit testing and all of those cool things.

**PETER:** So, I studied in Vienna, at the TU, in Austria, in computer science. I started my career at Siemens working on R F I D systems, radio frequency identification. So, I worked from the firmware up to system integrations.

**PETER:** And the one thing that I noticed back then is that gaining confidence in the correctness of complex systems is really difficult. So testing, you always miss something. So, there must be better ways of doing that. So, I started a PhD in systems verification at Inria in Grenoble in France.

**PETER:** And there, I worked in the area of formal methods for system design, rigorously specifying the behavior of a system. And then constructing an implementation that is mathematically proven. This is mainly used for safety critical systems. So, think nuclear power plants or airplanes and stuff like that, automated tools around it are also used in hardware development and increasingly also for operating systems, to make sure that they're secure, don't crash and so on. And then when I came as a research assistant to Oxford in the United Kingdom, I worked on techniques for software verification, programs written in C, so still low-level code. And this is based on a formalized semantics of the programming language.

**PETER:** And then we are able to check whether C program satisfies certain specified properties. And this is known as model checking and at a lower level, these techniques are based on solving logical constraints.

**PETER:** This machine based logical reasoning is one of the oldest areas of artificial intelligence. And these techniques can also be used for automatically generating test cases. For example, you can ask the system, what are the inputs that you need to feed in the program so that a certain branch in the code gets executed. And then you get the inputs and can construct a test that executes the program.

**CRAIG:** and then you've applied this to automatic unit test generation.

**PETER:** Exactly. So, this was the idea when we founded Diffblue with Professor Daniel Kroening from Oxford to apply this and essentially bring expertise in program analysis to a larger audience of developers.

**PETER:** And the key idea was to change the way software is developed by increasing the level of automation to free up developers from chores like writing unit tests that machines can supposedly do much better, so that developers then can focus on more difficult and creative tasks in software development.

**CRAIG:** Yeah. And unit tests, most of the audience we’re speaking to would understand unit tests, but can you explain what a unit test is and traditionally how it's created.

**PETER:** So, unit tests are tests that are written while the software is developed before or during the development or slightly afterwards, depending on the methodology that is used.

**PETER:** And it tests a unit of code, which is generally defined as the smallest part that you can reasonably test, think of a clause in Java or a couple of clauses that interact with each other. And so, you write this test that checks the behavior of just this small part so that when you compose the parts together, you can rely on them.

**CRAIG:** And something called test led development, it seems from talking to people that very few people really do test led development. There's a lot of talk about it. And a lot of people don't even write the unit tests as they're writing the software. Is that fair to say it?

**PETER:** Yeah. Companies are at different levels with respect to that and have different approaches to unit testing. So, test driven development, TDD, is one methodology where you essentially start writing the test before you write any code. And essentially the tests, the unit test becomes the specification for the bit of code that you are going to write.

**PETER:** Of course, the problem is, sometimes you need to implement a bit in order to understand what are the requirements, and then you can only write a test afterwards. So usually in practice it becomes a bit of a more iterative approach. Instead of a purist approach of writing all the tests upfront and then writing the implementation.

**CRAIG:** And what about not writing tests at all or writing tests for larger blocks of code?

**PETER:** Yeah. So, when you start with a software project, when it's still small, it's easy to write just end to end tests to essentially test the behavior end to end. And then as it grows bigger, you realize you have so many end-to-end tests that execute slowly, that you can't reasonably develop at a fast pace anymore because the tests are just too slow to execute.

**PETER:** And then you realize, ah, actually I should have written unit tests. And this happens in many projects. This is what we also see with our customers that there are large pieces of code that are just lacking these unit tests that would enable them to move at a faster development speed.

**CRAIG:** Yeah. And when you say unit on a typical computer screen, how many lines of codes does that contain?

**PETER:** A typical unit test is usually between 10 and 20 lines of code, depending on …

**CRAIG:**  The test itself or the unit …

**PETER:** The test itself. The unit can, depends, be a few hundred lines usually.

**CRAIG:** And then there's legacy code that was written before this unit testing protocol became a normal way of doing things. Is there a lot of that legacy code out there that has no unit tests?

**PETER:** There is a lot of such code out there that does not have a lot of unit testing.

**CRAIG:** And if you write code and I think we talked about Cobol when I was doing an article on the resilience of Cobol and the effort to get away from Cobol. If you write code and it works, why would it suddenly not work? Can bugs develop in software that's been written and tested end to end and works, and then suddenly something isn't working anymore? Or do problems develop when people start changing units within a large program.

**PETER:** Yeah. So usually, problems start when code needs to be maintained.

**PETER:** So, software is not static that you develop it once deploy it and then it runs forever. So usually, you need to keep pace with the technology and upgrade it. There is some functionality that needs to be added or changed depending on the business. So, software is hardly ever completely static.

**PETER:** And so that you can make these changes with confidence, you need a certain level of testing. Otherwise, you are going to introduce bugs.

**CRAIG:** Yeah. And if you introduce a bug like in this legacy code that has no unit test, how do you find the bug?

**PETER:** Yeah, usually in the worst case, the bug gets found by the customer.

**PETER:** When something doesn't work. And then of course, this is really the worst case in terms of reputation, cost to fix it, et cetera.

**CRAIG:** And in that case, this is something I've always wondered. If you find a bug, you report it, does the developer then look at the log file and can see exactly where in the program, in the millions of lines of code that make up the program, the bug exists? Or how do they find the bug in the code base?

**PETER:** Yeah, this depends very much on how much logging and observability there is in the system. In a large distributed system with many services, you need to have really good logging and introspection into the system so that you understand what is going on so that you can actually trace and hunt down the actual location of the bug and understand what is going on.

**CRAIG:** But unit testing, if you unit test the entire code base as it's being developed, if you make a change, how does the unit test help identify a bug?

**PETER:** When you make a change and you have a reasonable number of tests, then the assumption is that at least one of the tests is going to fail when you make changes.

**PETER:** And then of course, you need to check whether this failure is due to the change that you're intended to make, or whether it's something that you inadvertently broke while making the change.

**CRAIG:** In the traditional way, do you have, is there a programmer and then a test programmer and they sit side by side and work that way?

**CRAIG:** Or is it one programmer has to write a test or write the unit and then write the test, run the test, or is there some way to spread that across more than one person?

**PETER:** So usually, the developer who makes the change to the software, implements a new feature also writes the unit test. So, it's not that one person works on tests and the other one on the implementation.

**CRAIG:** And when you're developing software from scratch the same thing, it's the same developer that's doing both the test and the coding of the program.

**PETER:** Yes. So, for unit tests it is the developer who does both. Yes.

**CRAIG:** So yeah, it seems that automating that would be tremendous. How does AI play into this? I know that you guys use reinforcement learning, and I'm curious to hear how that works.

**CRAIG:** I also want to talk to you about some of the new, large language models that are now writing code, either auto completing code or writing unit tests. And how that compares to what you're doing. So first explain how you guys are using reinforcement learning to  write tests.

**PETER:** Yeah. So, at Diffblue we have a product called Diffblue Cover that writes unit tests fully autonomously.

**PETER:** And so, the tests that it produces, they reflect the current behavior of the code. So, they are meant to be used as regression tests. So that, when you make a change later, you can see that something has changed, which might be a bug. So, this is the intention of how this is used. At Diffblue, we take an approach from the programming language perspective.

**PETER:** Since our background is in program analysis, and software verification, we are not machine learning experts at origin. So, we look at the problem from a programming language perspective. Almost all algorithms that do something in this area, they essentially boil down to solving a search problem in a humongous search space.

**PETER:** We've experimented with multiple algorithms that might be suitable, but what we currently use is reinforcement learning, as you say, to perform the search.

**CRAIG:** And I'm sorry, you're searching a library of unit tests or what are you searching?

**PETER:** So, we are searching for a suite of tests, so essentially a couple of tests, that test a certain function or method in the code.

**CRAIG:** And where do those tests reside? Are there libraries of them or

**PETER:** No, the test code is written by our tool. So, we take the user's software and look at the various units, what we would consider units like in Java clauses and these clauses have certain methods. And then we would look at all methods essentially, and write for each method as many tests as necessary to test the functionality of that method.

So essentially reverse engineering, the specification from the implementation, this is what we are doing. And we are writing down the specification in the form of unit tests.

**PETER:** In the case of Java, it's just a bunch of methods that exercise this particular method on the test. And this code is what we write. The search that happens in our algorithms is essentially in the space of all possible methods that test this code or might test this code. So, we are essentially searching the space of all possible methods that could be written for methods that test particular parts of a method on the test.

**CRAIG:** And then who actually writes the test.

**PETER:** So, the test, our tool automatically writes the tests.  So full Java code is produced by our test. You can compile it and run it to execute the tests.

**CRAIG:** And why reinforcement learning? Is it because I don't typically, I don't think of reinforcement learning using, being used in search, is it actually writing tests and running them?

**CRAIG:** And then it comes up with the five best tests and the user picks, which one they want to use, or how does that work?

**PETER:** So, we usually compare it with the way AlphaGo works. So, the automatic system for playing the game Go and that one also uses reinforcement learning. So, it identifies areas of this huge search space where potentially there are potential moves to win the game, and then they spend more time in these parts of the search space to actually make a selection of which move to make next. And then they repeat. And what we do is, in some sense, similar. We come up with potential tests, then we evaluate them to see what is the best test we currently have. And repeat this operation until we have a full test suite.

**CRAIG:** And that's being done in the background.

**CRAIG:** And the Diffblue Cover that customers are using then is being updated as you run more, as you train it more and more, is that right?

**PETER:** Not quite. We realized that there are additional constraints that we need to consider. For example, everything needs to run behind the customer's firewall for certain customers that we have. So, it cannot connect to any service that we host because the code is just not allowed to leave the premises. Another requirement that we have is determinism. So, each time you ask the system to get a test for the same code, it must return the same test. Then we also had a requirement that the customer should install the system and get results within one or two minutes at most.

**PETER:** And we also have a very small time budget. So, for each method on the test, we are creating tests for, we are aiming at an average time of one second. So essentially, we need to do the learning on one method within one second. So, we cannot do a large number of iterations. So, we are limited in that sense as well.

**PETER:** And all these additional requirements have shaped the solution that we have.

**CRAIG:** But the learning takes place on Diffblue servers.

**PETER:** So, in our reinforcement learning loop, we predict what potentially good tests are. And this prediction is based on a pre-trained model solution. It's a very small model that we ship with tools. And then we measure how good these tests are, pick the best test. In reinforcement learning, you have a reward function and that is based on various criteria, like what is the coverage of the test, but also what is the aesthetics? So, this is very important because the coding style, the idioms that are used, there are certain frameworks that require certain ways of testing. There is mocking. So, there are all sort of things that make this, the reward function quite complicated.

**PETER:** It's not just, increasing coverage. There are also other things, other metrics in the game here. And so, we use various techniques from program analysis and aesthetics. So, it's important that the tests produced, that they look as if a human has written them.

**PETER:** Because when one of these tests then fails. When a software developer makes a change, then they need to look at it and it needs to look familiar. If it was just some machine generated code, then it will be difficult for them to debug the problem and fix it. So, the test, they need to pass the Turing test for tests, as we say, the human shouldn't be able to tell whether this test was written by a machine or a human.

**CRAIG:** That's so interesting. How does this relate to what's going on in large language models and transformers, and I think you and I exchanged emails. There's a new paper out about something called TiCoder, which purports to write unit tests using a large language model.

**PETER:** So that there are a variety of technologies that one can use to solve such a complex problem.

**PETER:** And no one fits all solutions. And so, some techniques are better for certain parts, other techniques are better for other parts of the problem. Particularly with software, the functionality of software is, in essence, mathematically defined by the programming language semantics.

**PETER:** So, there is not really a need to reverse engineer everything from data. One knows what the behavior is. You can just use the semantics of course. And for some cases it's better to reverse engineer from data, but in other cases, it's better just to rely on the semantics, to do the logical reasoning. So, we use a combination of multiple techniques.

**PETER:** For the functionality, that one is mostly mathematical. There are other aspects like, as I said, the coding style, the idiomatic aspects that are more preference based where more fuzzy algorithms are more suited. So initially when we started Diffblue, there were only a few comparable tools.

**PETER:** And they were hardly known by anyone. So now there are more tools, like you mentioned, these GPT based tools, like Github Copilot and also others. So, the use case that they focus on is essentially interactive. So, it's essentially very powerful code completion and one of the reasons why this is a good use case for these kinds of techniques is the accuracy.

**PETER:** They claim that they are 50% accurate on average. And so, this is not a problem for auto completion. Cause you can look for a couple of suggestions and you just pick the one that you want. So, for the interactive use case, this is not a problem. And the accuracy will certainly increase as these tools get better and better.

**PETER:** At Diffblue, we have focused on fully autonomous unit testing. So, there is no interaction with a human, so it needs to produce reasonable results without asking the human what is good.

**CRAIG:** That's interesting. Could Diffblue Cover draw on a large language model and then run the  reinforcement learning on the unit test written by a large language model.

**PETER:** It certainly could. We are looking at how we can leverage these language models to improve our tool.

**CRAIG:** I'm interested in where coding is going generally with automated code generation right now. Unit tests are one thing. AlphaCode writes complete programs, but they're very basic programs. In watching this space, do you think we'll reach a day where people can write programs in natural language and let the computer code it and write unit tests at the same time. It seems to me over time these systems are going to be capable of writing complete complex programs. Do you think that's possible.

**PETER:** So when talking about the future of coding I think we, shouldn't not only look at coding in the narrow sense of programming something. I think it looked, we need to look at the software development as a whole.

**PETER:** And the main challenge there is, essentially, how do I teach a computer to perform a certain task that provides value to me or my customer? And so, this is the difficult part. How do I instruct the computer to do that? And we go back in history, we were programming on the machine code level, and then there were higher and higher abstractions and compilers that perform the automation to allow us to be more and more productive on higher levels.

**PETER:** The problem still there is how do we actually teach the intent to the computer? And so, in the real world, software requirements are usually vague. And so, they are not usually written down in English language, it still remains vague to a certain degree.

**PETER:** So, to resolve all these ambiguities in English written specification, there needs to be some incremental refinement, some conversation between the human and the machine, and I'm convinced that the machine can help the human in asking questions about how they should behave in a certain situation - the machine might ask, but yeah, but what should happen in that case?

**PETER:** Have you thought about that? And then that way, I think it is possible to develop a common understanding between the human and the machine, what the software should be doing, and then the machine can implement it. So, I think the difficulty is the interface between the human and the machine to make this really work.

**PETER:** So, where we are at the moment is that we can automate certain manual activities and we can support humans in performing some of the more challenging activities. And then finally, we'll automate the latter. So, it will certainly come in stages as a gradual development rather than a revolution.

**CRAIG:** I'm trying to think of ambiguity in language in talking about programs. Fairness, right?

**CRAIG:** Fair is a very ambiguous word. But over time, it seems that a system would learn the various meanings of fair and could query the human developer about the various meanings and nuances of fair.

**PETER:** Yes, certainly.

**CRAIG:** So, what you're saying is that it's gradual, it's iterative and incremental, but eventually we'll get there.

**CRAIG:** Or do you think that natural language is never going to be precise enough for a computer to understand that and write code appropriately.

**PETER:** Yes. I'm not even sure that is meaningful or even desirable to have full automation. The purpose of AI or software, more generally, is to serve humans.

**PETER:** As I said, to provide some value for a human. So there needs to be some, at least one iteration for the human to confirm what the machine has understood is actually what the human intended.

**CRAIG:** I had an interesting conversation with a guy who was at Intel.

**CRAIG:** He's now got his own startup, but he was working on, I think he called it machine programming. His ambition or his vision was to eventually get to the point that his mother, who was a businesswoman, could talk to a computer and have the computer write a program to do what she wanted, some simple tasks.

**CRAIG:** And I know that Microsoft’s Power apps do something through natural language. And it may not be the, at the level of a real program, but at least can create macros and that sort of thing from natural language. So, on that journey from where we are today to complete program writing from natural language, how far along that journey do you think we are?

**CRAIG:** Is it 10% of the way there? Or 20% of the way there, or we were not even at 1%,

**PETER:** This is, it's hard to predict. It's certainly not next year in two years. It's probably a few decades out.

**CRAIG:** Can you put this in context, given the current shortage of software developers or coders, how automated unit testing can speed development. For example, what percentage of the average software developer's time is spent writing unit tests? That could be done automatically.

**PETER:** Yeah. So, we have done a survey, I think two years ago. And so, developers said that they spend roughly 35% of the time testing software. So, there are some significant gains to be made just by automating a part of this.

**PETER:** And in some sense, every business is a software business. So, there is an extremely high demand for software developers. That demand will only increase. So, it seems there are no limits to software that will need to be written. And with the salaries increasing and the current shortage, many companies just need to do more with the resources they've got.

**PETER:** And it's also why these low-code and no-code frameworks are becoming more popular. People are able to implement certain software functionality without being a full-blown software developer. A prediction by Gartner, they expect that 65% of all app development will be done by no code in 2024? I'm not sure what this number is based on. I would certainly expect that this will not displace software developers from their job, but more software now can be written by people other than software developers that account for such an increase.

**CRAIG:** And the 35% that you mentioned, if everyone used automated unit test software, that would in effect save 35% of their time and they could write a third more software.

**PETER:** Exactly. Iit will not happen that software developers are fired because now an automation tool replaces them because there always will be more software that needs to be written.

**CRAIG:** That’s it for this episode. I want to thank Peter for his time. If you want to learn more about unit test automation, visit Diffblue.com and give them a try. I believe they have a community edition that you can use for free. I also want to thank our sponsor, ClearML, which provides a suite of machine-learning tools for AI developers. Check them out at clear.ml

**CRAIG:** As always, you can find a transcript of our conversation today on our website, eye-on.ai.

**CRAIG:** And finally, remember: AI is about to change your world. So, pay attention.