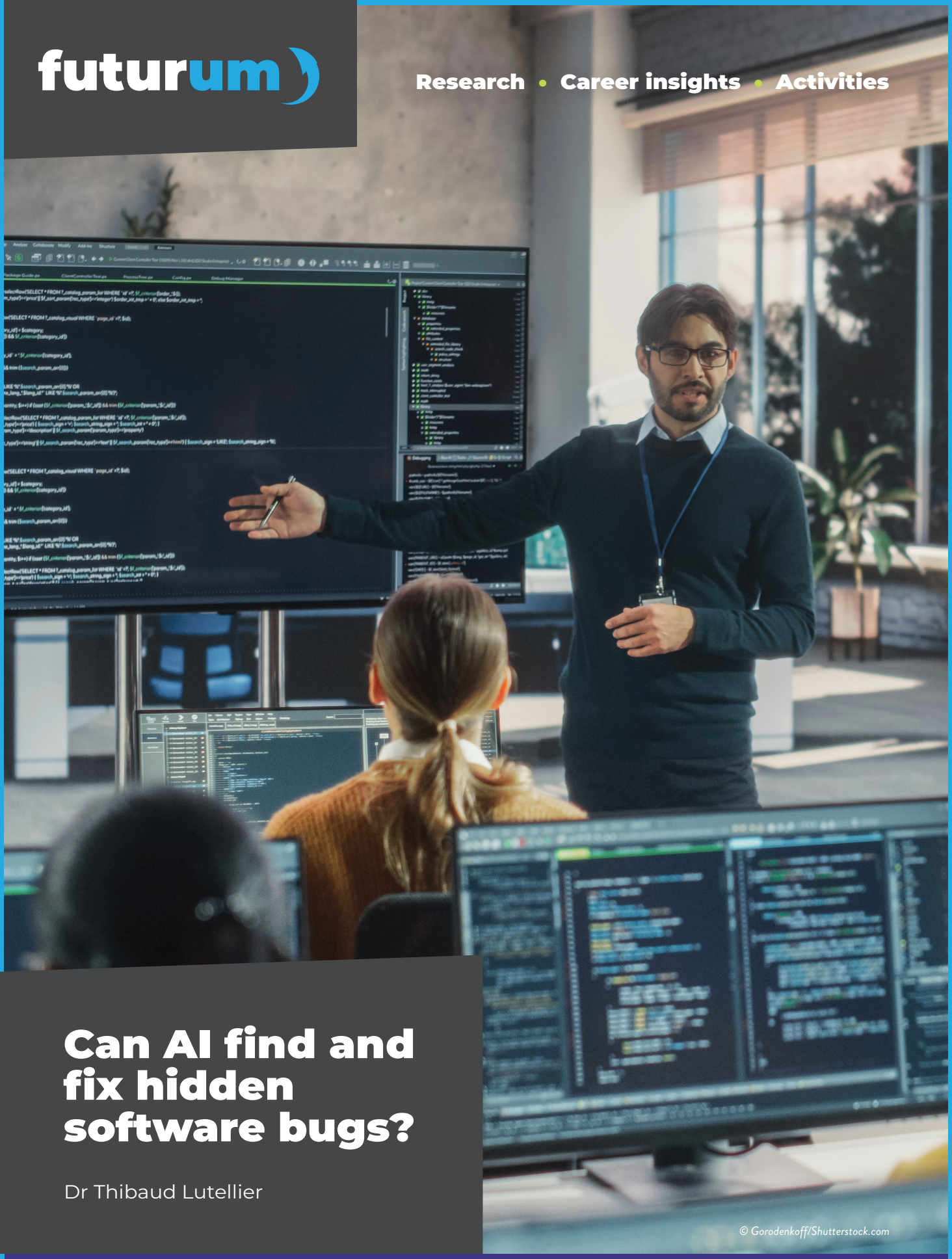
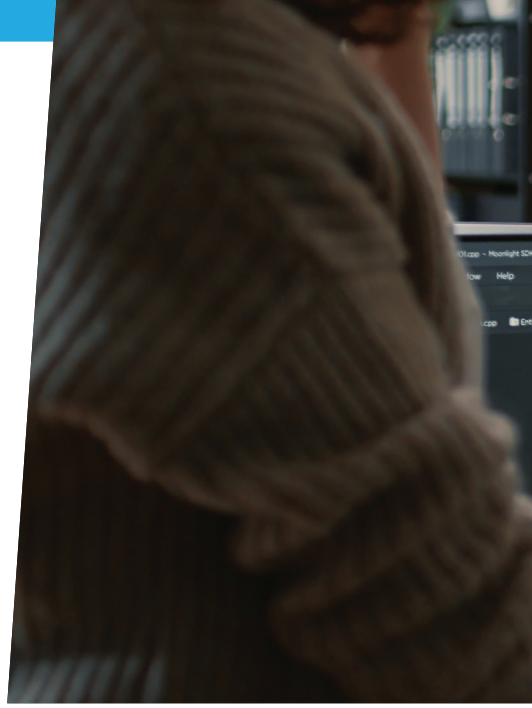
futurum

Research • Career insights • Activities



Can AI find and fix hidden software bugs?

Dr Thibaud Lutellier



Can AI find and fix hidden software bugs?

Software bugs can have devastating consequences, so finding and fixing them is a constant job for software engineers. At the **University of Alberta** in Canada, **Dr Thibaud Lutellier** is investigating how AI can help software developers to improve code.

Dr Thibaud Lutellier

Department of Science, Augustana Campus, University of Alberta, Canada

PROFILE

Field of research
Software engineering

Research project
Using AI to detect and fix software bugs

Funder
Natural Sciences and Engineering Research Council of Canada (NSERC)

doi: 10.33424/FUTURUM712

Talk like a ...

software engineer

Annotation – metadata added to code that provides extra information about how it should be used or understood

Code smell – a sign that code may contain a design problem or be difficult to maintain, potentially leading to future bugs

Code refactoring – reorganising and improving the structure of code without changing how it behaves

Jupyter Notebook – an interactive document that combines computer code, text and outputs such as graphs and tables

Software bug – an error in a computer program that causes unexpected or incorrect behaviour

Software is everywhere, from mobile phones and electric cars to banks and hospitals. However, even the most carefully designed software can contain bugs – errors that cause programs to behave in unexpected ways. "A minor bug might simply cause a visual glitch, like an incorrectly positioned image on a webpage," says Dr Thibaud Lutellier, a software engineering at the University of Alberta. "However, a major bug can have real-world consequences, such as disrupting an airport or allowing attackers to steal banking information."

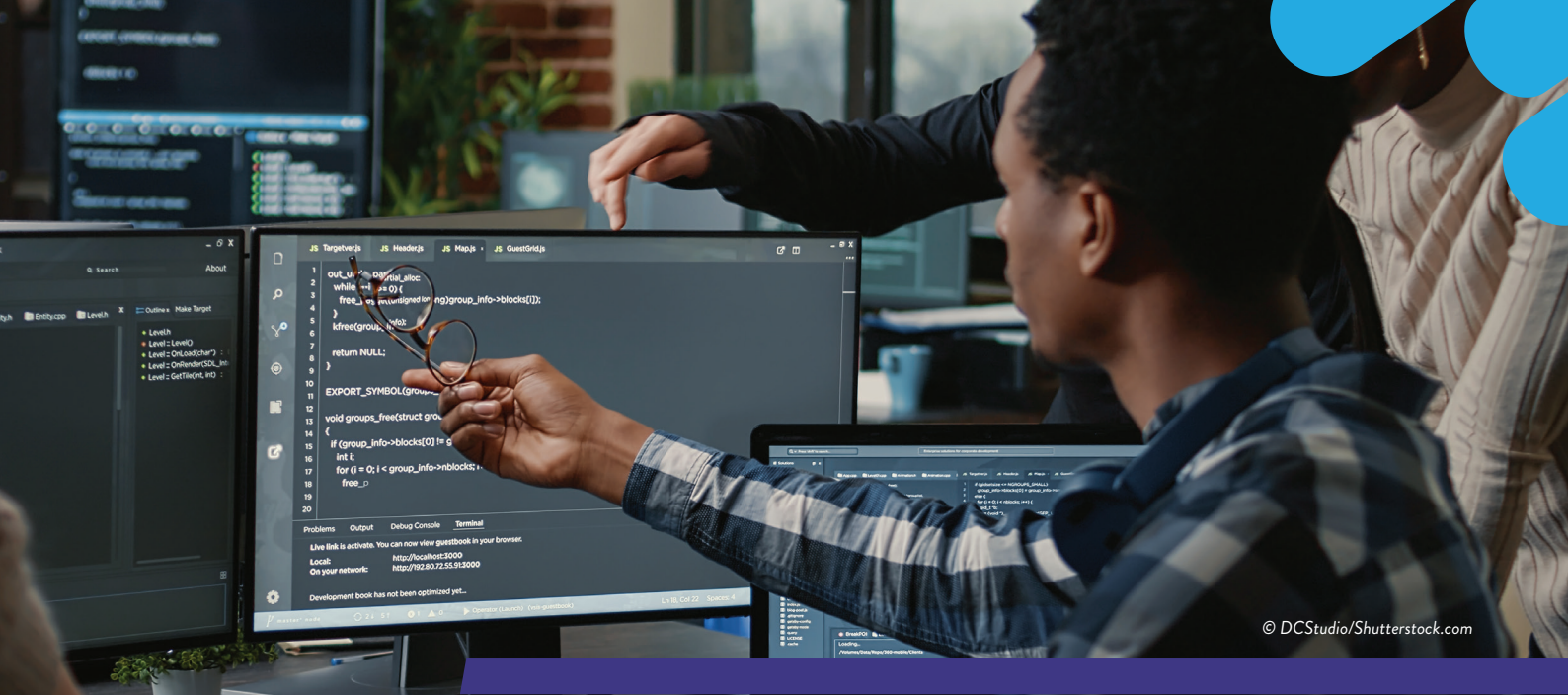
What are Jupyter Notebooks?

Many data scientists, computer scientists and researchers use Jupyter Notebooks to write and run code. Unlike traditional software development, where code is written in a file and executed from beginning to end, Jupyter Notebooks combine code, documentation and outputs such as graphs and tables in a single interactive document.

"Instead of one long script, the code is broken down into individual blocks called cells," explains Thibaud. Users can run a cell and immediately see the results it produces. This makes Jupyter Notebooks particularly useful for analysing large datasets, as researchers can quickly see how changing a variable affects their results without rerunning the entire code.

Why are Jupyter Notebooks prone to bugs?

Unfortunately, the flexibility of Jupyter Notebooks comes with a drawback. Users can run cells in almost any order, so they might run one cell, go back and change an earlier cell, then skip another cell entirely. This means the notebook may appear to



work even though some parts of the code are missing or out of date. For example, a programmer might accidentally delete a cell containing an important variable, yet the notebook continues to work because that variable is still stored in the computer's temporary memory from an earlier run. Once the notebook is closed and reopened, the variable disappears and the code suddenly breaks. "This chaotic environment makes it remarkably easy to introduce subtle, hard-to-detect bugs that standard software tools aren't built to find," says Thibaud.

How is Thibaud detecting bugs in Jupyter Notebooks?

Traditional bug detection tools were designed for conventional software, so they struggle to accurately analyse Jupyter Notebooks with independent cells that can be run in any order. To address this challenge, Thibaud developed JupOtter, an AI-powered bug detection system specifically designed for Jupyter Notebooks. "Instead of analysing the entire project as a single unit, JupOtter uses AI models to examine code at the cell level," he explains. "It treats individual cells as self-contained blocks of logic while using AI to track how data flows and changes between those cells, catching hidden bugs that traditional systems completely miss."

Why is it important to refactor code?

Finding bugs is only one part of software maintenance. Developers also spend a significant amount of time refactoring code – reorganising and improving a program's internal structure without changing how it behaves. "It is essentially a deep-cleaning and reorganising process for a program's codebase," says Thibaud.

Modern software systems can contain millions of lines of code and are often worked on by large teams of developers. Without regular maintenance, code can become disorganised and overly complex. Developers call this 'spaghetti code' because the different parts become tangled together, making the software difficult to modify and increasing the risk that a small change in one area will unexpectedly break something elsewhere.

Is AI better at refactoring than humans?

Thibaud and his team have been comparing how humans and AI refactor code by analysing software changes made by developers and AI coding agents. They found that human developers often make structural improvements, such as moving related pieces of code together or breaking large sections into smaller, reusable components. In comparison, AI agents spend most of their effort modifying annotations.

They also discovered that AI agents perform over 60 times more refactoring actions than humans. However, making more changes does not necessarily improve software quality. The team found that AI agents often repeatedly rewrite code to make it appear cleaner or to update annotations. While these changes may look productive, they can create a false sense of progress and increase development costs without providing meaningful improvements.

In addition, despite making many more edits, AI agents did not consistently produce better code. AI introduced code smells – indicators of design flaws or potential bugs – at rates similar to human developers. "In fact, one AI agent was found to introduce a statistically significant increase in code smells after refactoring," says Thibaud. "Ultimately, we found that high-volume AI modifications do not correlate to genuine code improvement, meaning human developers still need to closely audit AI changes."

What role will AI play in the future of software engineering?

"There is a fear that AI is going to replace software engineers and developers," says Thibaud. "Based on our research and the current trajectory of autonomous coding tools, I think this is unlikely. Instead, AI will fundamentally shift what a software engineering job entails."

As AI agents become capable of generating larger amounts of code, software systems will become bigger and more complex. However, more code does not automatically mean better software. Developers will still need to ensure that AI-generated code is reliable and free from errors. The role of software engineers will shift away from writing every line of code and towards reviewing and testing code produced by AI systems, opening exciting new opportunities in the field.

About *software engineering*

Software engineering is the design, development and maintenance of computer software. The field combines problem-solving and technical knowledge to create the programs and applications that power everything from smartphones and video games to hospitals, banks and spacecraft.

Why is software engineering an appealing career?

Software engineering is an exciting field because it allows you to turn ideas into real-world tools and technologies. "It is one of the few fields where you can build something incredibly powerful out of nothing but an idea and a computer," says Thibaud. "Software touches every domain, so you get the chance to work in almost any discipline you can think of."

What challenges will future software engineers be addressing?

The field is evolving quickly as AI becomes increasingly capable of generating code. While these tools can help developers work more efficiently, they also present new challenges. As AI systems produce larger amounts of code, software projects will become more complex. "The challenge won't be generating code," says Thibaud. "It will be managing the growing amount of technical debt and keeping increasingly complex software systems running smoothly over time."

What does a career in software engineering involve?

"People often think that software engineers and developers spend all their time writing code, sitting alone in front of their computers," says Thibaud. "This is far from true!" Instead, they work closely with product managers to understand what software should do and decide which features to prioritise. They also spend time reviewing other people's code to search for logic errors, security weaknesses and potential bugs. And testing software is an important part of the job, helping to ensure that programs behave as expected.

Pathway from school to *software engineering*

At school, computer science and mathematics will help you build the foundation for software engineering by developing computational thinking, logic and problem-solving skills

"Software engineers spend a lot of time explaining technical concepts to clients and discussing complex code with collaborators, so strong writing and speaking skills are essential," says Thibaud. "Don't neglect subjects that will help you develop communication skills."

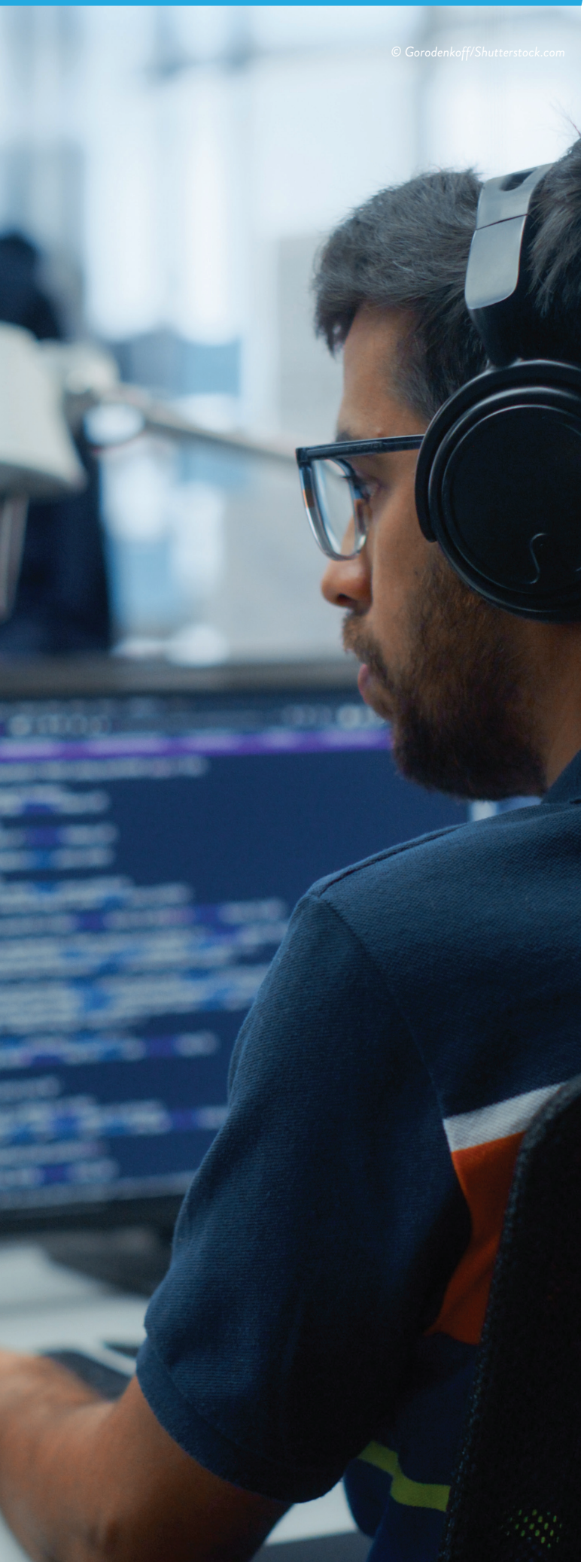
At university, a degree in software engineering, computer science or computer engineering could lead to a career in software engineering.

Explore careers in *software engineering*

Software is found in every aspect of our lives, so there are software engineering career opportunities in every sector!

The best way to explore software engineering is by learning through practice. As well as watching tutorials, try building something you are interested in. "If you like video games, try to create a small computer game, or a mod for a game you like," suggests Thibaud. "If you are part of a local club or organisation, try to build a website for them from scratch."

Online resources can help you learn software engineering skills. Organisations such as freeCodeCamp (**freecodecamp.org**) and Codeacademy (**codecademy.com**) include beginner-friendly courses related to programming and computer science.



Meet *Thibaud*

I have loved computers since I was a kid. As a teenager, I loved playing video games. I also enjoyed reading and playing table tennis and soccer with friends.

When I went to university, I initially studied chemistry for two years but I was not great at it. After that, I realised my favourite course during those two years was a course in computer programming. So I switched my degree to computer engineering. When I moved from France to Canada to study for my master's degree, I was fortunate to have a fantastic supervisor and lab mates who really made me fall in love with research in software engineering.

The part of my job as a software engineer that I enjoy the most is understanding a problem, breaking it down into small pieces and solving it. The most challenging thing is that software engineering is a field that moves extremely fast, meaning we always have to keep ourselves up to date with new technologies, hardware and changing user habits. To stay relevant and effective, software engineers must continuously adapt, step out of their comfort zones, and learn new technologies and programming languages.

In my free time, I love having fun and unwinding with my family and our pets (we have a cat and a dog). I also really enjoy playing board games and video games, hanging out with friends, reading and painting.

Thibaud's *top tips*

1. Don't be afraid of taking a non-linear path or changing direction in your education – learning what you don't like to do is still learning, and very important.
2. Many skills are transferable. When you learn something, think about how it can apply to whatever career you want to pursue.
3. Find mentors, professors or peers who challenge and support you. This will completely transform your career and make your work exciting – we can't succeed alone!

Software engineering

with Dr Thibaud Lutellier

Talking *points*

Knowledge

1. What is a software bug?
2. What is code refactoring?

Comprehension

3. What are some possible consequences of software bugs?
4. Why are Jupyter Notebooks more difficult to debug than traditional code?

Analysis

5. What are the differences in how human developers and AI agents refactor code? And what do these differences reveal about the ways humans and AI approach problem-solving?
6. AI agents perform many more refactoring actions than human developers, but why might this not lead to higher quality software?

Evaluation

7. As AI becomes more involved in software development, the role of software engineers may shift from writing code to reviewing and improving AI-generated code. How do you think this change will affect the skills needed by future software engineers?
8. Software is essential in critical areas such as healthcare, banking and transport. How much responsibility do you think humans should have when AI is used to create software that affects people's lives?

More *resources*

- This article explains how Thibaud is using AI to detect software bugs: **ualberta.ca/en/augustana/news/2023/01/harnessing-ai to-catch-bugs-lutellier.html**

Activity

Thibaud's research highlights the opportunities and challenges of using AI in software development. AI tools can help developers detect bugs, write code and improve software more quickly. However, AI-generated changes do not always improve software quality, meaning human judgement and review remain essential.

Choose an area where AI is being used (e.g. software development, healthcare, education, transport, finance). Split into two groups for a debate. Group 1 will argue in favour of using more AI in your chosen area, while Group 2 will argue against increasing the use of AI. Before beginning the debate, conduct some research into the topic and gather your ideas and arguments:

- Research the benefits of using AI in your chosen area, including how it could improve efficiency, solve problems or support human decision-making:
  - How is AI currently being used in this area?
  - What tasks can AI complete faster or more accurately than humans?
  - How could AI help people make better decisions or solve difficult problems?

- Research the challenges of using AI, including potential risks, mistakes, security concerns or over-reliance on AI systems:
  - What could happen if AI makes a mistake?
  - Are there situations where human judgement is more important than AI?
  - What risks could come from relying too heavily on AI?

- Can you find real-world examples where AI has been helpful or caused challenges? How do these examples support your argument?

Conduct a respectful debate about whether or not AI use should be increased in your chosen area. During the debate, discuss the following:

1. What are the biggest benefits and risks of using AI in your chosen area?
2. Why are human knowledge and judgement still important when using AI?
3. What tasks should AI be responsible for and what tasks should humans be responsible for?

After the debate, individually or in groups, reflect on the following:

1. How did considering both sides of the debate affect your opinion of AI?
2. How do you think AI will change the way people work in the future?
3. As AI becomes more common in different industries, how do you think the skills people need for future careers will change?

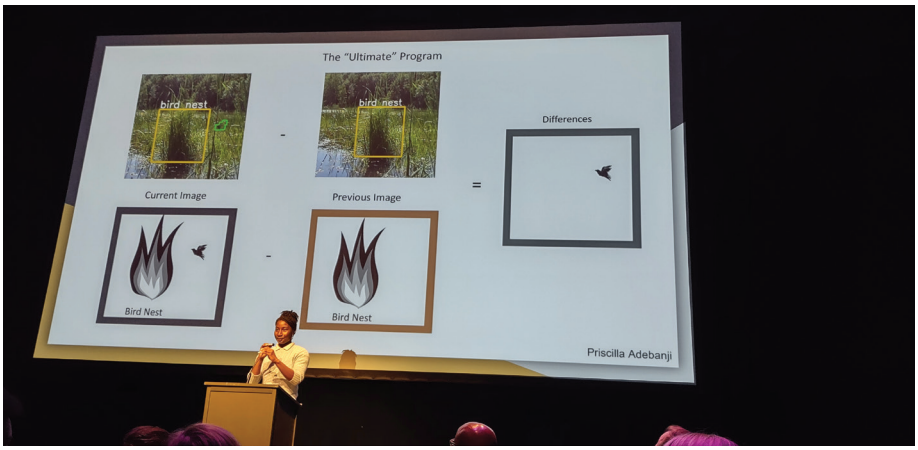


Photo *montage*

Top: *Thibaud is developing AI tools to detect and fix software bugs.*

Middle and bottom: *Thibaud (bottom) and his students Priscilla (left) and Cora (right) present their work to fellow computer scientists.*

+44 117 909 9150
info@futurumcareers.com
www.futurumcareers.com

futurum