Offer #2024-07154

PhD Position F/M [Campagne Allocation Région 2024] "Echo-Debugging": methods and tools for identifying and understanding software bugs

Contract type : Fixed-term contract

Level of qualifications required : Graduate degree or equivalent

Fonction : PhD Position

About the research centre or Inria department

The Inria University of Lille centre, created in 2008, employs 360 people including 305 scientists in 15 research teams. Recognised for its strong involvement in the socio-economic development of the Hauts-De-France region, the Inria University of Lille centre pursues a close relationship with large companies and SMEs. By promoting synergies between researchers and industrialists, Inria participates in the transfer of skills and expertise in digital technologies and provides access to the best European and international research for the benefit of innovation and companies, particularly in the region.

For more than 10 years, the Inria University of Lille centre has been located at the heart of Lille's university and scientific ecosystem, as well as at the heart of Frenchtech, with a technology showroom based on Avenue de Bretagne in Lille, on the EuraTechnologies site of economic excellence dedicated to information and communication technologies (ICT).

Context

Research topic and its scientific and economic context.

Although debugging techniques are varied, it is still difficult to easily identify the source of a program failure. As a result, debugging remains a challenge [3] [7], and the sources of hard-to-detect bugs are numerous [2]. It is difficult to understand why a given code change caused a given bug [5]. In some works, such as [5] and [4], developers have access to an earlier version of the software without the bug, which is an interesting source of information to help them. Without dedicated support, developers have to run the two versions in separate debuggers, manually advance them in parallel and visually compare the executions.

There are already existing techniques that automatically compare two similar executions to produce different results [4]. Generally, these techniques attempt to isolate code fragments that are (suspected to be) responsible for an error. Delta debugging [6] takes two versions of a program and finds the smallest subset of code changes that caused a given test to go from green to red. These techniques are extremely slow and consume energy and memory. For example, a single delta debugging cycle can take up to several hours to isolate the suspect code fragments.

In this thesis, we aim to develop a more time- and energy-efficient solution to this problem. This solution is based on a comparison of executions. It must take into account the specificity of living systems. In living programming systems, developers constantly observe and manipulate executions in progress. Feedback is immediate, and tools are at the heart of liveliness.

In the case of failure, developers use debuggers to find the source and understand the reasons for failures in the program. In living programming systems, debugging is therefore also alive, i.e. developers observe, understand and correct errors in a program without interrupting or restarting execution. Debugging can be enhanced by the introduction of a dynamic layer [9] in which behavioral variations in the program are collected and activated as a set of adaptations. The aim of this dynamic layer approach is to perform unplanned debugging of a running program. Unlike other debugging methods, live debugging allows code to be modified at runtime and variable changes to be made in real time to the state of the program. This approach makes it possible to test temporary fixes, iterate quickly on solutions and observe program behavior without the need for a re-run.

In this thesis, we want to continue the innovation on debuggers and enable the understanding of

comparative program executions. There are many possible applications. These include :

- the implementation of numerous changes within the application, resulting in erroneous execution
- the evolution of a library used in a program: with version n the software works, with version n+1 the software no longer works
- malicious software attacks: the discovery of discrepancies/divergences in software attacks enables security flaws to be identified more quickly.

Assignment

Application development inevitably introduces bugs. Often, it's not clear why a code change introduced a bug. To find this cause-and-effect relationship and debug more efficiently, developers can sometimes rely on the existence of a previous version of the code without the bug. Yet, traditional debugging tools are not designed for this kind of work, making it a tedious operation.

In this thesis, we propose an approach that enables us to debug an application in a live system, such as Pharo or Python, by comparing two executions with different results: one execution succeeds and the other fails.

Based on this hypothesis, we propose in this thesis to answer the following challenges:

- How to detect divergence, i.e. different behavior between two executions of a program?
- How can we reduce the cost in time, memory and energy consumption of detecting divergence(s) on long executions?
- What are the criteria for deciding whether a divergence is normal or unauthorized?
- What about detecting multiple divergences in the same program?
- What abstractions are needed to compare two executions of a program and detect divergences?

To meet these challenges, in addition to a precise state-of-the-art on debugging techniques, the PhD student will study concrete cases of program execution and propose a tool to be integrated into Pharo.

Main activities

Objectives and expected results.

The aim of this thesis is to design debugging methods, techniques and tools to help developers produce the next generation of software systems.

This objective is broken down as follows:

- A rigorous approach to in-live analysis of running programs
- A platform for automated analysis, enabling:
  - detection of program discrepancies
  - classification of discrepancies into normal and unauthorized discrepancies
  - suggest corrections for unauthorized deviations
- The possibility of extending this platform to analyze:
  - programs in other langages
  - scaling up with programs communicating with other programs from other applications (such as client/server applications)
- The design of a new generation of debugging tools integrating the various functionalities mentioned above
- The validation of such debuggers in an industrial context
- The evaluation of the debuggers accuracy, distinguishing between true and false divergences (false positives).

The proposed framework will be evaluated using applications written in Pharo. This choice of language is motivated by the following factors:

- Pharo is a language with a live, interactive environment that meets the criteria of live-programming, debugging and interactive execution.
- We've been developing the official Pharo debugger since 2018, turning it into a robust and extensible tool. This gives us a powerful basis for confidently building debugger prototypes, and for empirically evaluating those prototypes. The debugger is in fact used by the entire Pharo community - over 10,000 people, who are used to seeing research results transferred to their debugger.

Further evaluation will be carried out using other languages such as Java and Python.

Work program with deliverables and provisional schedule.

1. The PhD student will work incrementally in 3-4 month cycles on the following tasks: A literature survey on debuggers, program execution comparisons, delta debugging.
2. An initial prototype for program comparison. Today, the developer has to manually advance each execution. We therefore need to provide a tool that enables two programs to be executed and analyzed in parallel.
3. A second task will be to improve this first prototype to identify points of divergence in the executions. Based on the literature, he/she will propose a first version of an interactive visualization to be used for decision-making. Preliminary studies and informal feedback show that too fine a granularity in the comparison of executions results in analysis times that are too costly (time, memory, energy). It will therefore be necessary to provide abstractions that enable relevant comparisons. For example, Time Traveling Queries [8] can be used to query program execution and identify the program states that verify the query.
4. The third task will be to suggest corrections for unauthorized deviations. In addition, the improvement of integrated tools (IDEs) to give immediate feedback to developers will be studied.

Excepting the first phase, each stage will involve the proposal of new methods, the creation of associated tools, empirical evaluations of method/tool combinations and publications. Depending on their level of maturity and the results of their evaluation, the tools will be integrated into the official Pharo distribution debugger.

We will develop open-source tools that we will initially deploy in the Pharo community. We will also collaborate with its industrial consortium to validate our approaches under real-life conditions.

References.

[1] **T. Dupriez, S. Costiou, and S. Ducasse.** "First infrastructure and experimentation in echo-debugging". In Proceedings of the 2020 International Workshop on Smalltalk Technologies, 2020.

[2] M. Perscheid, B. Siegmund, M. Taeumel, and R. Hirschfeld. "Studying the advancement in debugging practice of professional software developers". Software Quality Journal, 25(1):83–110, 2017.

[3] I. Sommerville. "Software Engineering (6th ed.)". Addison-Wesley, 2001.

[4] E. W. Wong, R. Gao, R. Abreu, and F. Wotawa. "A survey on software fault localization". IEEE Transactions on Software Engineering, 42(8):707–740, 2016.

[5] A. Zeller. "Yesterday, my program worked. today, it does not. why?" In ESEC/FSE-7: Proceedings of the 7th European software engineering conference held jointly with the 7th ACM SIGSOFT international symposium on Foundations of software engineering, pages 253–267, London, UK, 1999. Springer-Verlag.

[6] A. Zeller. "Isolating cause-effect chains from computer programs". In SIGSOFT '02/FSE-10: Proceedings of the 10th ACM SIGSOFT symposium on Foundations of software engineering, pages 1-10, New York, NY, USA, 2002. ACM Press.

[7] A. Zeller. "Why Programs Fail: A Guide to Systematic Debugging". Morgan Kaufmann, Oct. 2005.

[8] **M. Willembrinck, S. Costiou, A. Etien and S. Ducasse,** "Time-Traveling Debugging Queries: Faster Program Exploration," *2021 IEEE 21st International Conference on Software Quality, Reliability and Security (QRS)*, Hainan, China, 2021, pp. 642-653.

[9] **Steven Costiou, Mickaël Kerboeuf, Clotilde Toullec, Alain Plantec, Sté□phane Ducasse.** Object Miners: Acquire, Capture and Replay Objects to Track Elusive Bugs. The Journal of Object Technology, 2020, 19 (1), pp.1:1.

[10] Al Bessey, Ken Block, Ben Chelf, Andy Chou, Bryan Fulton, Seth Hallem, Charles Henri-Gros, Asya Kamsky, Scott McPeak, and Dawson Engler. 2010. "A few billion lines of code later: using static analysis to find bugs in the real world". Commun. ACM 53, 2, 66–75.

Skills

- Good programming skills
- Software engineering concepts
  - programming
  - reasonning
- Writing
- Autonomy and ability to work in group (with other team members if needed)
- Good level in English

Benefits package

- Subsidized meals
- Partial reimbursement of public transport costs
- Leave: 7 weeks of annual leave + 10 extra days off due to RTT (statutory reduction in working hours) + possibility of exceptional leave (sick children, moving home, etc.)
- Possibility of teleworking and flexible organization of working hours
- Professional equipment available (videoconferencing, loan of computer equipment, etc.)
- Social, cultural and sports events and activities
- Access to vocational training
- Social security coverage

Remuneration

2100€ gross per month for the 1st and 2nd years

2190€ gross per month for the 3rd year

General Information

- **Theme/Domain :** Distributed programming and Software engineering
  Software engineering (BAP E)
- **Town/city :** Villeneuve d'Ascq
- **Inria Center :** Centre Inria de l'Université de Lille
- **Starting date :** 2024-10-01
- **Duration of contract :** 3 years
- **Deadline to apply :** 2024-04-30

Contacts

- **Inria Team :** EVREF
- **PhD Supervisor :**
  Etien Anne / Anne.Etien@inria.fr

About Inria

Inria is the French national research institute dedicated to digital science and technology. It employs 2,600 people. Its 200 agile project teams, generally run jointly with academic partners, include more than 3,500 scientists and engineers working to meet the challenges of digital technology, often at the interface with other disciplines. The Institute also employs numerous talents in over forty different professions. 900 research support staff contribute to the preparation and development of scientific and entrepreneurial projects that have a worldwide impact.

Warning : you must enter your e-mail address in order to save your application to Inria. Applications must be submitted online on the Inria website. Processing of applications sent from other channels is not guaranteed.

Instruction to apply

CV + cover letter

Defence Security :
This position is likely to be situated in a restricted area (ZRR), as defined in Decree No. 2011-1425 relating to the protection of national scientific and technical potential (PPST).Authorisation to enter an area is granted by the director of the unit, following a favourable Ministerial decision, as defined in the decree of 3 July 2012 relating to the PPST. An unfavourable Ministerial decision in respect of a position situated in a ZRR would result in the cancellation of the appointment.

Recruitment Policy :
As part of its diversity policy, all Inria positions are accessible to people with disabilities.